Instytut Informatyki Teoretycznej i Stosowanej
Polskiej Akademii Nauk

Rozprawa doktorska pt.

# Wyjaśnialność i bezpieczeństwo systemów inteligentnych

mgr inż. Katarzyna Filus

Praca napisana pod kierunkiem dr hab. inż. Joanny Domańskiej, prof. IITiS PAN

Gliwice, 2023

*Niniejszą pracę doktorską dedykuję mojemu zmarłemu Tacie -*
*dr hab. inż. Zdzisławowi Filusowi, profesorowi Politechniki Śląskiej.*

*Tato, dziękuję Ci za zaszczepienie we mnie umiłowania wiedzy,*
*ciągłego wspierania mojego rozwoju oraz utwierdzania mnie w przekonaniu,*
*że najlepsze co można zrobić to po prostu być sobą.*

# Spis treści

**1 Wstęp** **13**

**2 Analiza tematu** **21**
2.1 Algorytmy Sztucznej Inteligencji . . . . . 21
2.2 Wyjaśnialność systemów inteligentnych . . . . . 26
2.2.1 Interpretowalność modeli uczenia maszynowego . . . . 26
2.2.2 Metody wyjaśnialności dla modeli uczenia głębokiego . 28
2.2.3 Intuicyjność metod a praktyczna wyjaśnialność . . . . 29
2.3 Bezpieczeństwo systemów inteligentnych . . . . . 30
2.3.1 Podwójnie dualna natura algorytmów sztucznej inteligencji . . . . . 31
2.3.2 Tradycyjne zagrożenia systemów informatycznych . . . 32
2.3.3 Zagrożenia dedykowane algorytmom sztucznej inteligencji . . . . . 41
2.4 Testowanie i ocena działania algorytmów uczenia maszynowego 44
2.4.1 Testowanie metod tradycyjnych . . . . . 44
2.4.2 Testowanie metod uczenia głębokiego . . . . . 46
2.4.3 Automatyczne oznaczanie danych na potrzeby testowania algorytmów uczenia głębokiego w wizji komputerowej . . . . . 47
2.5 Relacja wyjaśnialności i bezpieczeństwa systemów inteligentnych . . . . . 48

**3 Bezpieczeństwo** **51**
3.1 Analiza krajobrazu luk bezpieczeństwa bibliotek uczenia głębokiego i numerycznych na przykładzie TensorFlow . . . . . 51
3.1.1 Proponowane rozwiązanie i metodologia badań . . . . . 52
3.1.2 Wyniki . . . . . 58
3.1.3 Wnioski . . . . . 69

3.2 Wykorzystanie metryk statycznych analizatorów kodu do wykrywania luk bezpieczeństwa w kodzie napisanym w językach C/C++ . . . . . 70
3.2.1 Proponowane rozwiązanie i metodologia badań . . . . . 72
3.2.2 Wyniki . . . . . 79
3.2.3 Wnioski . . . . . 86
3.3 Wykorzystanie autorskiej modyfikacji Losowych Sieci Neuronowych do wykrywania zagrożeń bezpieczeństwa . . . . . 88
3.3.1 Proponowane metody dla Losowych Sieci Neuronowych 93
3.3.2 Wykrywanie ataków sieciowych opartych na infrastrukturze botnet z wykorzystaniem Losowych Sieci Neuronowych . . . . . 99
3.3.3 Wykrywanie podatności w kodzie z wykorzystaniem Losowych Sieci Neuronowych . . . . . 100
3.3.4 Wyniki . . . . . 102
3.3.5 Wnioski . . . . . 105
3.4 Wykorzystanie fizycznych markerów do automatycznego tworzenia zbiorów danych oraz testowania sieci głębokich . . . . . 108
3.4.1 Proponowany system do testowania . . . . . 109
3.4.2 Metodologia badań . . . . . 112
3.4.3 Wyniki . . . . . 117
3.4.4 Wnioski . . . . . 119
3.5 Netsat - intuicyjny atak na pośrednie warstwy sieci głębokich . 122
3.5.1 NetSat - Network Saturation Attack . . . . . 124
3.5.2 Metryka Niepodobieństwa - Dissimilarity Metric (DM) 126
3.5.3 Metodologia badań . . . . . 128
3.5.4 Wyniki . . . . . 130
3.5.5 Wnioski . . . . . 131

**4 Wyjaśnialność 135**
4.1 Network Activation Mapping - metoda wyjaśnialności dla głębokich sieci konwolucyjnych . . . . . 135
4.1.1 Proponowana metoda wizualizacji . . . . . 136
4.1.2 Metodologia badań . . . . . 137
4.1.3 Wyniki . . . . . 140
4.1.4 Wnioski . . . . . 149
4.2 Wykorzystanie intuicyjnych metod filtrowania do stworzenia wyjaśnialnego systemu lokalizacji . . . . . 150
4.2.1 Lokalizacja w środowiskach wewnętrznych . . . . . 152
4.2.2 Proponowany system lokalizacji z uwzględnieniem prostych w interpretacji metryk . . . . . 154

4.2.3 Metodologia badań . . . . . 159
4.2.4 Wyniki . . . . . 161
4.2.5 Wnioski . . . . . 165

**5 Podsumowanie** **167**

**A Lista publikacji** **207**

# Streszczenie

Systemy inteligentne są stosowane w wielu obszarach życia człowieka. Chociaż oferują wysoką dokładność oraz efektywne rozwiązywanie problemów, ich praktyczne stosowanie jest ograniczone ze względu na liczne zagrożenia związane z nimi oraz ich niską wyjaśnialność. Te aktualne problemy skutkują niskim zaufaniem, którym społeczeństwo darzy sztuczną inteligencję oraz ograniczonym stosowaniem tego typu systemów w dziedzinach, w których bezpieczeństwo jest krytyczne. Z tego powodu konieczne jest zaproponowanie metod umożliwiających poprawę bezpieczeństwa i wyjaśnialności obecnych systemów, a także uwzględnieniem tych aspektów w procesie projektowania nowych rozwiązań.

**Celem niniejszej pracy doktorskiej jest poprawa bezpieczeństwa i wyjaśnialności systemów inteligentnych.** Aby zrealizować cel pracy, w sposób kompleksowy zbadano zagadnienia dotyczące bezpieczeństwa i wyjaśnialności. Zaproponowano szereg metod mających na celu poprawę obu tych aspektów oraz zaprezentowano wyniki potwierdzające ich skuteczność i porównano je ze stosownymi metodami. Ze względu na sieć wzajemnych połączeń między bezpieczeństwem oraz wyjaśnialnością, część zaproponowanych metod ma pozytywny wpływ na oba rozpatrywane zagadnienia.

W przypadku bezpieczeństwa w sposób kompleksowy zbadano różne aspekty bezpieczeństwa systemów inteligentnych: tradycyjne zagrożenia informatyczne (cyberataki, podatności oprogramowania) oraz zagrożenia bezpośrednio związane z algorytmami sztucznej inteligencji.

W domenie cyberataków zaproponowano metodę wykrywania ataków opartą na autorskich metodach inicjalizacji oraz trenowania Losowych Sieci Neuronowych. Proponowana metoda inicjalizacji ma na celu zapewnienie neutralności sieci przed rozpoczęciem procesu jej trenowania oraz lepszą interpretowalność tego procesu, skutkując poprawą wyjaśnialności. Metoda trenowania ogranicza liczbę kosztownych operacji wykonywanych w procesie trenowania. Proponowane metody pozwalają uzyskać lepsze wyniki dokładności w kontekście wykrywania cyberataków w stosunku do bazowego rozwiązania.

W obszarze podatności oprogramowania przeprowadzono obszerną anali-

zę znanych podatności wiodącej biblioteki uczenia głębokiego - TensorFlow - oraz przetestowano adekwatność dostępnych statycznych analizatorów kodu w wykrywaniu tych podatności. Jako że dostępne narzędzia wykazują niską skuteczność w wykrywaniu podatności tego typu oprogramowania, zaproponowano metody wykrywania podatności oparte na tradycyjnych algorytmach uczenia maszynowego oraz selekcji cech, a także system hybrydowy wykorzystujący autorską modyfikację Losowych Sieci neuronowych i cechy mieszane opisujące kod programu. Wykazano, że zaproponowane rozwiązania poprawiają dokładność w stosunku do bazowych rozwiązań.

W pracy zaproponowano również metody z zakresu testowania głębokich sieci neuronowych. Zaproponowano metodę umożliwiającą tworzenie zbiorów danych na potrzeby kompleksowego testowania w rzeczywistych warunkach, a także metody umożliwiające testowanie sieci pod kątem zagrożeń dedykowanych - ataków adwersarza. Pierwsza z metod umożliwia automatyczne tworzenie oznaczonych zbiorów danych i testowanie głębokich sieci neuronowych bezpośrednio na pojazdach sterowanych automatycznie (ang. Automated Guided Vehicles, AGV). Zaproponowany atak adwersarza umożliwia testowanie sieci wizyjnych i jest niezależny od klasyfikatora sieci. Uzyskane wyniki pokazały, że próbki wygenerowane za pośrednictwem proponowanego ataku skutkują wyższą szkodliwością niż próbki wygenerowane za pośrednictwem powszechnie stosowanych ataków. Zaproponowano również prostą w interpretacji metrykę umożliwiającą praktyczną ocenę stopnia szkodliwości ataków adwersarza. W pracy opisano zalety proponowanej metryki w stosunku do dostępnych metryk. Prostota interpretacji i wykorzystania metryki wywiera pozytywny wpływ na aspekt wyjaśnialności.

W zakresie wyjaśnialności zaproponowano metodę interpretacji działania głębokich sieci konwolucyjnych. Metoda obejmuje wizualizację aktywacji sieci oraz jej ogólnego skupienia na danym obrazie. Metoda jest niezależna od klasyfikatora sieci oraz nie wymaga wyznaczania wartości jej gradientów. Zaprezentowano, że różne warianty metody mogą być stosowane do wizualnego nadzoru ekstrakcji wzorców oraz stronniczości. Zaprezentowano również, że metoda może być stosowana do badania wpływu ataków adwersarza na działanie sieci, co ma pozytywny wpływ na aspekt bezpieczeństwa. Proponowana metoda jest prostsza od dostępnych metod, a jednocześnie informatywna, co skutkuje polepszeniem wyjaśnialności działania sieci. Zaproponowano również system inteligentny do lokalizacji użytkowników z telefonami w przestrzeni. Rozwiązanie wykorzystuje siłę sygnału Bluetooth oraz autorskie metryki pozwalające odfiltrować niewiarygodne odczyty pozycji użytkowników. Zaproponowane metryki są sformułowane w taki sposób, aby były proste w interpretacji nawet dla nietechnicznych użytkowników systemu inteligentnego. Wywiera to pozytywny wpływ na aspekt praktycznej wyjaśnialności.

# Abstract (English)

Intelligent systems are used in many areas of human life. Although they offer high accuracy and effective problem-solving, their practical application is still limited due to their security issues and low explainability. These current problems result in a low level of trust that society places in artificial intelligence, which limits the use of such systems in safety-critical domains. Therefore, it is necessary to propose methods that would improve the security and explainability of existing systems, as well as to consider these aspects in the process of designing new solutions.

**The aim of this doctoral thesis is to improve the security and explainability of intelligent systems.** To achieve this goal, different issues related to security and explainability of such systems have been thoroughly investigated. A series of methods has been proposed to improve these two aspects. The results confirming their effectiveness have been presented and compared with alternative existing approaches. Due to the interconnection between security and explainability, some of the proposed methods have a positive impact on both of these aspects.

In the case of security, different aspects of intelligent systems security have been comprehensively examined: both the traditional information technology threats (cyber attacks, software vulnerabilities) and the threats directly related to artificial intelligence algorithms.

In the domain of cyber attacks, an attack detection system based on novel methods for initialization and training of Random Neural Networks has been proposed. The proposed initialization method aims to ensure the neutrality of the network prior to the beginning of its training process and better interpretability of the process, resulting in improved explainability. The training method limits the number of expensive operations performed during the training procedure. The proposed methods allow to achieve better accuracy results in the context of detecting cyber attacks compared to the baseline solution.

In the area of software vulnerabilities, an extensive analysis of known vulnerabilities within the leading deep learning library, TensorFlow, has been

conducted, and the adequacy of available static code analyzers in detecting these vulnerabilities has been tested. Since the available tools show low effectiveness in detecting vulnerabilities in this type of software, vulnerability detection methods based on traditional machine learning algorithms and feature selection have been proposed, as well as a hybrid system using an original modification of Random Neural Networks and mixed features characterizing the program code. It has been demonstrated that the proposed solutions improve accuracy compared to the baseline solutions.

The thesis also presents methods from the field of deep neural network testing. A method has been proposed that allows to create datasets for comprehensive real-world testing of deep vision networks. Also, a method has been created that allows to test networks' resiliency to dedicated threats - adversarial attacks. The first method makes it possible to automatically create labeled datasets and test deep neural networks directly on automated guided vehicles. The proposed adversarial attack allows testing of vision networks and is independent of the network's classifier. The results showed that samples generated through the proposed attack result in higher harmfulness than samples generated through commonly used attacks. A metric that is straightforward in interpretation has also been proposed to enable practical evaluation of the degree of harmfulness of adversarial attacks. The thesis describes the advantages of the proposed metric over available metrics. The simplicity of interpretation and use of the metric has a positive impact on the explainability aspect.

In the area of explainability, a method for interpreting the operation of deep Convolutional Neural Networks has been proposed. The method allows to visualize the activation of the network and its overall concentration on a given image. The method is independent of the network's classifier and does not require the calculation of the values of its gradients. It has been presented that different variants of the method can be used for visual examination of pattern extraction and bias. It has also been presented that the method can be used to study the impact of adversarial attacks on network operation, which has a positive impact on the security aspect. The proposed method is simpler than the available approaches, and at the same time informative, which results in improved explainability of network operation. Also, an intelligent system for locating users with smartphones has been proposed. The proposed solution uses Bluetooth signal strength and original metrics that can be used to filter out unreliable readings of users' positions. The proposed metrics are formulated in such a way as to be straighforward in interpretation even for non-technical users of the intelligent system. This has a positive impact on the practical explainability of the system.

# Rozdział 1

# Wstęp

Sztuczna inteligencja (ang. Artificial Intelligence, AI) to ogólna dziedzina, w której tworzy się algorytmy samodzielnie uczące się sposobu wykonywania określonych zadań. Zadania te obejmują szeroką gamę czynności wykonywanych przez ludzi, między innymi kategoryzację, tłumaczenie tekstów oraz rozpoznawanie obiektów na obrazach. Celem algorytmów sztucznej inteligencji jest więc automatyzacja procesów wykonywanych wcześniej przez ludzi. Z drugiej strony celem metod sztucznej inteligencji jest także rozwiązywanie problemów, z którymi ludzie mają trudności: analiza danych dużej wymiarowości, czy też wymagających dużych nakładów obliczeniowych lub znajdowania nietrywialnych zależności w danych. W skład tej ogólnej dziedziny wchodzi uczenie maszynowe, a także poddziedziny uczenia maszynowego - uczenie głębokie oraz uczenie płytkie. Szeroko rozumiana sztuczna inteligencja znajduje zastosowanie w wielu różnorodnych obszarach, między innymi transporcie [1], przemyśle [2], medycynie [3] oraz rolnictwie [4]. Wraz ze wzrostem popularności inteligentnych domów i miast [5], rośnie zapotrzebowanie na rozwiązania inteligentne. Same rozwiązania tego typu można zdefiniować jako systemy wykorzystujące algorytmy sztucznej inteligencji, analizę danych oraz automatyzację do zwiększenia efektywności i ułatwienia podejmowania decyzji. W zależności od specyfiki problemu, wykorzystuje się różne rodzaje algorytmów sztucznej inteligencji.

Algorytmy należące do grup uczenia płytkiego i głębokiego są równie istotne wewnątrz dziedziny sztucznej inteligencji i służą do rozwiązywania problemów związanych z różnymi zagadnieniami i formą danych. Metody płytkiego uczenia maszynowego także obejmują szereg metod: od najprostszych, takich jak drzewa decyzyjne i maszyny wektorów nośnych, do modeli wykorzystujących wzmocnienie gradientowe. Algorytmy te są wykorzystywane głównie do analizy danych tabelarycznych, zawierających wartości ze zbioru liczb rzeczywistych, naturalnych, a także binarne i kategoryczne.

Algorytmy uczenia głębokiego natomiast wykorzystują różnego rodzaju sieci neuronowe: perceptrony wielowarstwowe, sieci konwolucyjne, sieci typu Transformer oraz sieci rekurencyjne. Stosuje się je głównie do przetwarzania obrazów i języka naturalnego, ale także danych serii czasowej [6]. Są to dane trudne w przetwarzaniu i automatycznym rozumieniu, charakteryzujące się dużą wymiarowością i abstrakcyjnymi relacjami wewnątrz danych. Możliwe jest również łączenie ze sobą algorytmów z grup uczenia głębokiego oraz płytkiego, na przykład poprzez wykorzystanie metod uczenia płytkiego do przetwarzania i analizy cech wyekstrahowanych za pośrednictwem głębokich sieci neuronowych.

Istotnym aspektem w kontekście sztucznej inteligencji jest jej wyjaśnialność (ang. explainability). Wyjaśnialność można zdefiniować jako zdolność do dostarczenia przez dany system opisu wniosków/podjętej decyzji w sposób zrozumiały dla człowieka. Wyjaśnialność jest jednym z kluczowych aspektów sztucznej inteligencji, zarówno z perspektywy społecznej, jak i technologicznej [7]. Brak transparentności systemu decyzyjnego utrudnia jego testowanie oraz identyfikację i poprawę ewentualnych błędów. Uniemożliwia to zdobycie pełnego zaufania społeczeństwa i przemysłu do sztucznej inteligencji [7]. Z tego też powodu powstała poddziedzina sztucznej inteligencji nazywana Wyjaśnialną Sztuczną Inteligencją (ang. Explainable AI, XAI). Najbardziej powszechnym obszarem badań wewnątrz tej dziedziny jest wyjaśnialność działania samych algorytmów sztucznej inteligencji. Algorytmy sztucznej inteligencji można w tym kontekście podzielić na metody typu białej skrzynki oraz czarnej skrzynki. Przykładem metody białoskrzynkowej jest drzewo decyzyjne. Wizualizacja struktury drzewa i dokonywanych w nim decyzji w transparentny sposób wyjaśnia jego działanie. Z drugiej strony w przypadku metod typu czarnej skrzynki interpretacja dokonywanych przez nie decyzji i wyznaczenie cech dominujących w procesie ich podejmowania nie jest trywialne. Przykładem algorytmów czarnoskrzynkowych są głębokie sieci neuronowe. Pomimo że to właśnie nieliniowość oraz duża liczba hierarchicznych przekształceń danych umożliwiają sieciom głębokim ekstrakcję abstrakcyjnych cech o wysokiej użyteczności, to elementy te w dużym stopniu zmniejszają ich interpretowalność. Co więcej, interpretowalność maleje wraz ze wzrostem głębokości sieci. Z tego powodu trudno jest wyjaśnić, dlaczego sieć dla konkretnych danych wejściowych zwróciła taki wynik, a nie inny. Trudno jest także wyjaśnić jaką część danych wykorzystała w procesie decyzyjnym. Obok dużej złożoności obliczeniowej oraz pamięciowej, niska interpretowalność modeli głębokich jest największą wadą tych algorytmów. Pomimo że istnieją pewne mechanizmy interpretacji działania głębokich sieci neuronowych, to algorytmy te są w dużym stopniu niezrozumiane, co wywiera negatywny wpływ na zaufanie społeczeństwa do sztucznej inteligencji i

ogranicza możliwość jej wykorzystania w dziedzinach, w których krytycznym aspektem jest bezpieczeństwo. Przykłady takich dziedzin to autonomiczne pojazdy, przemysł, czy też medycyna. Większość praktycznych systemów informatyki dąży do tego, aby możliwe było wyjaśnienie podejmowanych przez nie decyzji i zwracanych rezultatów. Z jednej strony umożliwia to efektywną analizę zagrożeń, a z drugiej wyjaśnia przyczyny ewentualnych incydentów. Z tego powodu duży nacisk kładzie się obecnie na rozwój domeny Wyjaśnialnej Sztucznej Inteligencji. Głównym celem jest minimalizacja konieczności wyboru między możliwościami modelu a jego interpretowalnością. Z tego powodu kluczowe jest dostarczanie nowych mechanizmów wyjaśniających działanie dostępnych algorytmów oraz projektowanie nowych algorytmów, tak aby można było w transparentny sposób interpretować ich decyzje.

Drugim aspektem XAI wydaje się być praktyczna wyjaśnialność rezultatów, która faworyzuje algorytmy i metryki o prostych w interpretacji i intuicyjnych rezultatach oraz obsłudze. Prostota interpretacji i intuicyjność jest szczególnie istotna w przypadku wykorzystywania rezultatów metryk przez osoby mniej związane z techniką. Praktyczna wyjaśnialność może przejawiać się możliwością tworzenia analogii między działaniem i rezultatami algorytmów oraz zjawiskami świata rzeczywistego, stałym zakresem wartości wyników, czy też możliwością wizualizacji. Prostota obsługi może polegać na przykład na braku konieczności ustalania nietrywialnych progów wartości koniecznych dla działania algorytmu. Aspektowi praktycznej wyjaśnialności poświęca się zdecydowanie za mało uwagi w domenie sztucznej inteligencji, pomimo że ma kluczowy wpływ na możliwość praktycznego wykorzystania metod w procesie podejmowania decyzji. Z tego powodu jest to jak najbardziej aktualny problem i konieczne jest tworzenie metod i metryk właśnie w takim duchu.

W kontekście sztucznej inteligencji warto również zwrócić uwagę na drugi istotny problem tej dziedziny - bezpieczeństwo. Zagrożenia związane z bezpieczeństwem systemów inteligentnych można podzielić na zagrożenia tradycyjne wynikające z przynależności systemów inteligentnych do szerszej grupy systemów informatycznych, a także zagrożenia dedykowane. Systemy inteligentne są bardziej podatne na tradycyjne zagrożenia związane z bezpieczeństwem niż inne systemy. Zagrożenia te obejmują podatności oprogramowania na zagrożenia i ataki sieciowe. Specyfika systemów inteligentnych sprawia, że w wielu przypadkach koszt operacji wykonywanych przez systemy oparte na algorytmach uczenia maszynowego dużej skali jest tak wysoki, że obliczenia są wykonywane za pośrednictwem chmur obliczeniowych. Wymiana informacji między urządzeniem docelowym a chmurą wymaga systemu łączności. Systemy te najczęściej korzystają z łączności bezprzewodowej, która to jest szczególnie narażona na zagrożenia natury sieciowej [8, 9]. Im więcej informacji

wymienianych między poszczególnymi komponentami systemu połączonymi za pośrednictwem sieci Internet, tym więcej danych, które mogą być potencjalnie przechwycone, zmodyfikowane i wykorzystane. Algorytmy sztucznej inteligencji są coraz częściej wykorzystywane przez urządzenia internetu rzeczy (ang. Internet of Things, IoT). Urządzenia te są często ograniczone pod względem możliwości pamięciowych i obliczeniowych, więc tym bardziej opierają swoje działanie na obliczeniach w chmurze. Niejednokrotnie z powodu prostoty, niskiego kosztu oraz kupowania niestandaryzowanych urządzeń z niesprawdzonych źródeł, aspekt bezpieczeństwa jest zaniedbywany i nie jest zapewniany akceptowalny poziom zabezpieczeń tych urządzeń. Staje się to krytyczne w takich przypadkach jak monitorowanie dzieci za pośrednictwem niań elektronicznych wykorzystujących sztuczną inteligencję do wykrywania pewnych zachowań dziecka [10]. Naruszenie zasad bezpieczeństwa może w tym przypadku skutkować przechwyceniem zdjęć dziecka przez przestępców seksualnych. W krytycznych sytuacjach przechwycenie urządzeń połączonych z siecią internet może skutkować nawet zagrożeniem życia, tak jak było to w przypadku podatności oprogramowania połączonych z internetem rozruszników serca w roku 2017 [11]. Ponadto algorytmy uczenia maszynowego dużej skali są naturalnym celem ataków sieciowych typu powódź HTTP, które wykorzystują poprawnie skonstruowane zapytania skierowane do systemu w celu uniemożliwienia korzystania z usług prawowitym użytkownikom. Jako że operacje wykonywane za pośrednictwem takich algorytmów są kosztowne, tym łatwiej jest doprowadzić do blokady usług (ang. Denial of Service, DoS) wspieranych przez nie systemów.

Wiele ataków i blokad działania usług jest możliwych z powodu podatności oprogramowania (ang. software vulnerability). Takie podatności mogą mieć wpływ na jeden lub więcej elementów triady CIA (ang. Confidenciality, Integrity, Availability - Poufność, Integralność, Dostępność do informacji). Większość produktów związanych z uczeniem maszynowym oraz obliczeniami numerycznymi dużej skali jest tworzona w językach takich jak C/C++ ze względu na ich szybkość i wysoką kontrolę nad zasobami. Jednocześnie ta natura rodziny języków C/C++ sprawia, że najbardziej powszechnymi i niebezpiecznymi podatnościami oprogramowania są te związane z właśnie tymi językami. Potwierdzają to raporty firmy Veracode [12] oraz ranking zagrożeń organizacji MITRE - CWE TOP25 [13]. Wszystkie te aspekty sprawiają, że należy zwrócić szczególną uwagę na bezpieczeństwo systemów inteligentnych właśnie z perspektywy zagrożeń tradycyjnych. Należy także proponować nowe rozwiązania bezpieczeństwa dla obszarów nierozerwalnie związanych ze sztuczną inteligencją, między innymi Internetu Rzeczy i aplikacji napisanych w językach C oraz C++. W tym miejscu interesującym zagadnieniem jest dualność krajobrazu bezpieczeństwa związanego ze sztuczną inteligencją. Cho-

ciaż algorytmy te są podatne na zagrożenia, to mogą one oferować lepszą dokładność niż alternatywne rozwiązania do wykrywania podatności oprogramowania i ataków sieciowych. Rozwiązania inteligentne wykazują olbrzymi potencjał w tych dziedzinach [14, 15, 16, 17]. Z tego powodu, konieczne jest proponowanie nowych efektywnych rozwiązań opartych na uczeniu maszynowym do wykrywania tych problemów. Sztuczna inteligencja i cyberbezpieczeństwo pozostają w wielowymiarowej relacji i cechuje je szereg wzajemnych zależności [18]. Z tego powodu, praca z rozwiązaniami sztucznej inteligencji mającymi na celu poprawę bezpieczeństwa pozwala poznać jeszcze jeden wymiar tego obszaru, co wywiera pozytywny wpływ na kompleksowe poznanie dziedziny na styku tych dwóch obszarów [19].

Kolejnym aspektem bezpieczeństwa systemów inteligentnych są zagrożenia dedykowane sztucznej inteligencji. Z jednej strony algorytmy uczenia maszynowego umożliwiają automatyzację wielu procesów, sztuczna inteligencja otwiera również nowe możliwości manipulacji i ataku, a także stawia nowe wyzwania w zakresie prywatności i ochrony danych. Konieczne jest zatem uwypuklenie roli cyberbezpieczeństwa w tworzeniu godnych zaufania i niezawodnych rozwiązań w zakresie sztucznej inteligencji [7]. Istnieje szeroka gama zagrożeń dedykowanych systemom opartym na algorytmach uczenia maszynowego [18]. Zagrożenia te wynikają między innymi z olbrzymiego zapotrzebowania i zależności od danych w każdej fazie życia produktów opartych na uczeniu maszynowym. Mogą one być związane zarówno z danymi treningowymi (zniekształcone dane, zmodyfikowane zmienne, etykiety), jak i z danymi wykorzystywanymi w fazie predykcji (np. zniekształcenie danych w celu spowodowania błędnych predykcji). Również same modele uczenia maszynowego mogą podlegać manipulacji mającej na celu nieprawidłowe działanie systemu. Dodatkowo, w kontekście uczenia głębokiego, jednymi z najbardziej nagłośnionych zagrożeń dedykowanych algorytmom sztucznej inteligencji stały się tzw. ataki adwersarza (ang. adversarial attacks).

Ataki adwersarza wykorzystują niezauważalne lub ledwo zauważalne dla ludzkiego oka perturbacje w danych powodujące błędne działanie systemu [20, 21]. Istnieją różne ataki tego typu, pośród których jedne zakładają dostęp do atakowanego modelu, a inne nie. Istnieją ataki mające na celu jakąkolwiek zmianę predykcji atakowanego modelu lub zmianę na konkretną wartość. Niezależnie od rodzaju danego ataku, stało się faktem, że ataki adwersarzy stały się już powszechne, co odnotowano w raporcie Agencji Unii Europejskiej ds. Cyberbezpieczeństwa (ang. European Union Agency for Cybersecurity, ENISA) [22]. Ich celem jest podważenie zaufania społeczeństwa i przemysłu do inteligentnych systemów informatycznych. Dlatego kluczowe jest, aby społeczność badawcza związana ze środowiskiem uczenia maszynowego dołożyła wszelkich starań celem poprawy bezpieczeństwa tych systemów. Dwa klu-

czowe działania w tym zakresie to proponowanie metod obrony przed tymi atakami, a także tworzenie nowych skutecznych ataków - nieodłącznego elementu testowania nowych i istniejących modeli oraz technik obrony przed zagrożeniami. Ataki adwersarza sprawiły, że testowanie jedynie dokładności systemów inteligentnych nie jest już wystarczającym kryterium ich działania i jakości. Bezpieczeństwo nie jest już jedynie opcją, ale koniecznością [18]. Jako że ataki typu białej skrzynki są zazwyczaj skuteczniejsze niż ataki typu czarnej skrzynki, konieczne jest proponowanie nowych typów praktycznych ataków, które mogłyby być włączone do procedur wszechstronnego testowania systemów pod kątem bezpieczeństwa.

Poprawa wyjaśnialności i bezpieczeństwa systemów inteligentnych to jedne z największych aktualnych wyzwań dziedziny sztucznej inteligencji. Aspekty te nie są jeszcze wystarczające zbadane, m.in. ze względu na młody wiek jednej z czołowych dziedzin sztucznej inteligencji, jaką jest uczenie głębokie. Dlatego też wydaje się, że zagadnienia te jeszcze długo będą aktualne i należy dołożyć wszelkich starań w celu ich lepszego zbadania. Jest to o tyle istotne, że aspekty te mają kluczowy wpływ na wiarygodność i zaufanie społeczeństwa do tego typu systemów. Jako że algorytmy sztucznej inteligencji są tworzone właśnie po to, aby służyć ludziom, to poprawa tego zaufania jest fundamentalnym problemem dziedziny. Z jednej strony wiarygodność cierpi w przypadku naruszeń triady CIA systemu, a więc bezpieczeństwa, a z drugiej strony w przypadku braku możliwości rozumienia otrzymywanych rezultatów, a więc z powodu braku wyjaśnialności. Co więcej, aspekty te są powiązanie siecią wzajemnych relacji. Brak wyjaśnialności ma negatywny wpływ na bezpieczeństwo, ponieważ między innymi utrudnia dogłębne zrozumienie systemu oraz wykonywanych w nim decyzji. Wyjaśnialność może natomiast cierpieć w przypadku naruszenia bezpieczeństwa, ponieważ często utrudnia ono wnioskowanie o właściwych przyczynach, dlaczego system inteligentny zachował się tak, jak się zachował [7]. Dodatkowym znakiem aktualności i zapotrzebowania na nowe rozwiązania z obszarów bezpieczeństwa i wyjaśnialności są liczne nowe standardy dotyczące bezpieczeństwa sztucznej inteligencji wprowadzane przez Europejski Instytut Norm Telekomunikacyjnych (ang. European Telecommunications Standards Institute, ETSI), np. ETSI GR SAI 001 V1.1.1, oraz ISO/IEC (np. ISO/IEC DTR 27563), a także wyjaśnialności wprowadzane przez Instytut Inżynierów Elektryków i Elektroników (ang. Institute of Electrical and Electronics Engineers, IEEE), np. IEEE 2894, IEEE 2976. Wszystkie te standardy zostały wprowadzone najpóźniej w roku 2021. Ze względu na aktualność problemów wyjaśnialności i bezpieczeństwa systemów inteligentnych, konieczne jest proponowanie nowych rozwiązań mających na celu poprawę tych aspektów.

**Celem niniejszej pracy doktorskiej jest poprawa bezpieczeństwa oraz wyjaśnialności systemów opartych na algorytmach sztucznej inteligencji.** Aby zrealizować ten cel, w niniejszej pracy zaproponowano szereg nowych metod oraz praktycznych obserwacji wynikających z przeprowadzonych analiz. Poniżej wyszczególniono główne elementy wkładu pracy:

1. Analiza krajobrazu zagrożeń wiodącej biblioteki uczenia głębokiego i numerycznej - TensorFlow - oraz weryfikacja możliwości wykrywania jej znanych podatności przez istniejące analizatory statyczne kodu.

2. Metoda wykrywania podatności oprogramowania za pośrednictwem metryk generowanych przez statyczne analizatory kodu. Wszechstronna analiza przydatności tych metryk do wykrywania podatności kodu.

3. Nowy sposób inicjalizacji wag oraz modyfikacja procedury trenowania za pośrednictwem metody najszybszego spadku Losowych Sieci Neuronowych. Zaproponowaną modyfikację sieci zastosowano do wykrywania ataków sieciowych z wykorzystaniem infrastruktury botnet oraz do stworzenia hybrydowego systemu wykrywania podatności oprogramowania na zagrożenia bezpieczeństwa.

4. System do automatycznego zbierania danych i etykietowania na potrzeby testowania algorytmów uczenia głębokiego w wizji komputerowej.

5. Nowy atak adwersarza na końcowe reprezentacje obrazów wewnątrz sieci konwolucyjnej.

6. Prosta w interpretacji metryka do oceny stopnia szkodliwości ataków adwersarza pod względem podobieństwa etykiet przed i po ataku w postrzeganej przez model przestrzeni klas.

7. Metoda wizualizacji i wyjaśniania działania głębokich konwolucyjnych ekstraktorów cech.

8. Zestaw prostych w interpretacji metryk do filtrowania ruchomych użytkowników systemu lokalizacji opartego na telefonach komórkowych.

Metody te wpisują się w różnorodne aspekty bezpieczeństwa i wyjaśnialności systemów inteligentnych. Aspekt bezpieczeństwa obejmuje zagadnienia poprawy bezpieczeństwa z perspektywy tradycyjnych zagrożeń systemów informatycznych (ataki sieciowe, podatności oprogramowania), ale także kompleksowe testowanie bezpieczeństwa i jakości algorytmów uczenia głębokiego za pośrednictwem automatycznie etykietowanych zbiorów, ataków adwersarza oraz interpretowalnych metryk. Także aspekt wyjaśnialności jest badany

z różnych perspektyw: od stosowania wyjaśnialnych algorytmów uczenia maszynowego do rozwiązywania problemów i badania specyfiki danych, przez projektowanie metryk, tak aby ich wartości były proste w interpretacji, aż do metod wyjaśniania działania algorytmów uczenia głębokiego. W pracy opisano proponowane metody oraz zaprezentowano wyniki potwierdzające ich adekwatność dla rozwiązywanych problemów.

**Zawartość pracy** W Rozdziale 2 opisano zagadnienia teoretyczne kluczowe dla problemów wyjaśnialności i bezpieczeństwa systemów inteligentnych. Przedstawiono tam również literaturę powiązaną z rozpatrywanymi problemami, uwypuklono jej braki, a także umotywowano i usytuowano w niej proponowane rozwiązania. W Rozdziale 3 zaproponowane w ramach pracy doktorskiej metody z dziedziny bezpieczeństwa systemów inteligentnych, a w Rozdziale 4 - z obszaru wyjaśnialności (prace dotyczące zagadnień z powyższej listy). Zaprezentowano również rezultaty uzyskane za pośrednictwem proponowanych metod potwierdzające ich adekwatność w rozwiązywaniu rozpatrywanych problemów. Celem prezentowanych rozwiązań jest poprawa przynajmniej jednego z aspektów sztucznej inteligencji rozpatrywanych w pracy - bezpieczeństwa lub wyjaśnialności. W poszczególnych sekcjach rozdziałów opisano również wnioski i spostrzeżenia wynikające z przeprowadzonych analiz. W Rozdziale 5 przedstawiono zbiorcze wnioski wynikające z pracy, podsumowano pracę oraz nakreślono dalsze kierunki rozwoju w obszarach wyjaśnialności i bezpieczeństwa systemów inteligentnych. Dodatek A stanowi pełną listę publikacji autorki aż do dnia złożenia pracy. W niniejszej pracy dołożono wszelkich starań, aby używać polskich terminów z zakresu sztucznej inteligencji, bezpieczeństwa i wyjaśnialności. W przypadku terminów, dla których nie istnieją jeszcze ustandaryzowane odpowiedniki w języku polskim, wykorzystano oryginalne angielskie nazwy, aby uniknąć niejednoznaczności.

# Rozdział 2

# Analiza tematu

W niniejszym rozdziale przybliżono kluczowe zagadnienia dla dziedziny sztucznej inteligencji oraz jej wyjaśnialności i bezpieczeństwa. Opisano dostępne rozwiązania oraz podkreślono braki w literaturze tematu, w które wpisują się rozwiązania przedstawione w pracy.

## 2.1 Algorytmy Sztucznej Inteligencji

Sztuczna inteligencja to niezwykle obszerna i różnorodna dziedzina. Obejmuje algorytmy skrajnie różniące się działaniem, celem, rozmiarem i naturą. Wspólnym celem algorytmów należących do obszaru sztucznej inteligencji jest automatyzacja analizy danych i wsparcie procesu podejmowania decyzji. Samą sztuczną inteligencję można w popularnonaukowy sposób zdefiniować jako „zdolność maszyn do wykazywania ludzkich umiejętności, takich jak rozumowanie, uczenie się, planowanie i kreatywność" [23].

Tak jak wspomniano wcześniej, sztuczna inteligencja to obszerna i bardzo ogólna dziedzina. Jest to grupa nadrzędna dla domeny uczenia maszynowego. Samo uczenie maszynowe, ze względu na naturę modeli, można podzielić na uczenie płytkie oraz uczenie głębokie.

**Uczenie płytkie** korzysta z prostszych struktur i technik niż uczenie głębokie. Algorytmy te zazwyczaj pozyskują wiedzę na podstawie statystycznych właściwości danych. Najczęstszą formą danych wejściowych dla tego typu algorytmów są dane tabelaryczne z kolumnami różnych typów (kolumnami mogą być np. płeć, wiek, średnia odczytów z sensora w określonym oknie czasowym). W przypadku algorytmów uczenia płytkiego dużą wagę przykłada się do wybrania (i/lub stworzenia) jak najlepszych cech opisujących dany problem. Z tym aspektem związane są pojęcia inżynierii cech i selekcji cech. Inżynieria cech polega na wykorzystaniu wiedzy domenowej do tworze-

nia funkcji przetwarzającej cechy w taki sposób, aby na cechach wynikowych łatwiej było uczyć się algorytmom uczenia maszynowego. Selekcja cech także ma na celu wsparcie procesu uczenia. Zbiory danych z dużą liczbą atrybutów często zawierają cechy niepotrzebne do rozwiązania danego problemu, a jednocześnie wprowadzające „szum" utrudniający efektywne trenowanie algorytmów uczenia płytkiego [24]. Jest to spowodowane koniecznością znajdowania przydatnych wzorców w wielowymiarowej przestrzeni cech. Redukcja wymiarowości skutkuje zazwyczaj bardziej efektywnym trenowaniem modeli (ich wyższą dokładnością) oraz krótszym czasem trenowania. Zastosowanie technik selekcji cech może także przeciwdziałać zbytniemu dopasowaniu modeli do zbioru treningowego, czyli mieć pozytywny wpływ na generalizację. Jeśli chodzi o same algorytmy uczenia płytkiego, to istnieje szeroka gama takich modeli, m.in. drzewa decyzyjne, maszyny wektorów nośnych, czy też naiwny klasyfikator Bayesa. Istnieją także bardziej zaawansowane modele z rodziny modeli zespołowych. Istnieją różnorodne techniki budowania takich modeli. Dwie podstawowe techniki to bagging oraz wzmacnianie (ang. boosting) [24]. Bagging zakłada wykorzystanie wielu słabych modeli, które uczą się osobno na podzbiorach danych treningowych równolegle i decydują o finalnej decyzji na podstawie głosowania. W przypadku wzmacniania, również wykorzystuje się słabe modele, ale w przeciwieństwie do bagging, uczą się one sekwencyjnie tak, aby adaptacyjnie poprawiać predykcje poprzednich modeli w sekwencji. Ogólny opis przedstawiony w tym paragrafie (znacznie skrócony i uproszczony) pokazuje, że dziedzina płytkiego uczenia maszynowego jest niezwykle obszerna i zróżnicowana. To samo tyczy się opisanej w następnym paragrafie dziedziny uczenia głębokiego.

**Uczenie głębokie** jest aktualnie najbardziej rozwijającą się domeną wewnątrz sztucznej inteligencji. W uczeniu głębokim wykorzystuje się głębokie sieci neuronowe, których głębia oznacza obecność wielu warstw transformacji danych wewnątrz modelu sieci (odróżnia to metody uczenia głębokiego od algorytmów uczenia płytkiego). Kolejne warstwy ekstrahują coraz to lepsze, bardziej informatywne, a zarazem abstrakcyjne informacje o danych wejściowych. Siłą uczenia głębokiego jest duża liczba prostych (zazwyczaj nieliniowych) transformacji wewnątrz warstw oraz licznych warstw, które to mają na celu usunąć z danych informacje nieistotne dla rozwiązywanego problemu, a jednocześnie wyłuskać z nich cenne dla tego problemu informacje. W tym miejscu można posłużyć się analogią sita, które przepuszcza pewne elementy (niepotrzebne według sieci informacje), a zostawia elementy pożądane (informacje wykorzystywane do wykonywania zadań, do których sieć została wytrenowana). Głębokość modelu określa się poprzez liczbę jego warstw.

Istnieje wiele różnych typów warstw i sieci. W domenie przetwarzania obrazu, pierwsze sieci wykorzystywały głównie warstwy ściśle połączone, które

są w stanie poradzić sobie z prostymi obrazami o stałym położeniu konkretnych obiektów w obrazie (np. cyfry ze zbioru MNIST [25]). Niemniej jednak, przełom w dziedzinie wizji umożliwił inny rodzaj sieci - sieci konwolucyjne (ang. Convolutional Neural Network, CNN) - i ich połączenie w roku 1989 r. w Bell Labs przez Yanna LeCuna z algorytmem propagacji wstecznej. Sieci konwolucyjne to sieci zawierające warstwy konwolucyjne. Warstwy te wykorzystują matematyczną operację splotu (inaczej konwolucji) i filtry do znajdowania tych samych wzorców w różnych lokalizacjach przestrzennych obrazu (mechanizm przesuwnego okna). Przez to sieci konwolucyjne charakteryzuje tzw. ekwiwariancja translacyjna (ang. translational equivariance) [26]. Mechanizm przesuwnego okna ma również pozytywny wpływ na wydajność sieci i konieczność nauczenia się mniejszej liczby parametrów niż w przypadku ciężkich warstw ściśle połączonych. Dostępnych jest wiele różnych modeli wykorzystujących różnorodne techniki: od starszych modeli z rodziny VGG [27], przez modele z rodziny ResNet [28] do nowszych wydajnych sieci z rodziny EfficientNetV2 [29].

W dziedzinie przetwarzania języka obecnie najczęściej wykorzystywane są sieci typu Transformer [30]. Jest to typ sieci wykorzystujący mechanizm tzw. warstwy skupienia (ang. attention layers). Sieci typu Transformer wyparły w dziedzinie przetwarzania języka sieci typu rekurencyjnego (z warstwami rekurencyjnymi), które są używane do przetwarzania np. danych serii czasowej. Przykładowe warstwy rekurencyjne to proste warstwy rekurencyjne oraz warstwy rekurencyjne mające na celu uczenie się zależności długoterminowych w danych (np. Long Short-Term Memory, LSTM [31] oraz Gated Recurrent Unit, GRU [32]). W dziedzinie języka naturalnego sieci te zostały wyparte przez sieci typu Transformer m.in. ze względu na lepsze wyniki oraz prostsze zrównoleglenie operacji. Później, sieci typu Transformer zostały wykorzystane również w wizji komputerowej (Transformery wizyjne, ang. Vision Transformer, ViT [33]) i aktualnie rywalizują w tej dziedzinie z sieciami konwolucyjnymi.

W porównaniu do sieci CNN, potrzebują albo znacznie więcej danych do uogólnienia [34], albo zaawansowanych schematów uczenia [26]. Pozwalają jednak na lepsze powiązanie przestrzenne odległych obiektów, wykazując przy tym większą odporność na zmiany tekstury i okluzje [35]. Potrafią one uchwycić globalny kontekst w obrazach, ale mają trudności z drobnymi szczegółami [36]. Transformery hierarchiczne [36], które ponownie wprowadzają „przesuwne okno" inspirowane sieciami konwolucyjnymi, zostały wprowadzone jako pierwszy szkielet wizyjny ogólnego przeznaczenia oparty na ViT. Wprowadzono również modele hybrydowe CNN-ViT (np. CvT [37], LeViT [38]). Po sukcesie sieci typu Transformer, niektóre z opracowanych technik zostały wykorzystane do modernizacji sieci CNN [39] - tak powstały sieci z

rodziny ConvNeXt. Te czysto konwolucyjne modele konkurują z sieciami typu Transformer pod względem dokładności, skalowalności i odporności. Różne sieci mogą wykonywać różne zadania wewnątrz swoich dziedzin zastosowania. Dla przykładu, główne problemy wizji to rozpoznawanie obiektów, detekcja obiektów i segmentacja semantyczna. Podstawą wszystkich tych zadań jest klasyfikacja. W przypadku zadań języka naturalnego przykładowe zadania to predykcja zamaskowanych wyrazów, auto-uzupełnianie i streszczanie. W przypadku danych serii czasowych może to być na przykład detekcja anomalii lub odszumianie. Tak jak w przypadku metod płytkich uczenia maszynowego jest to jedynie drobny wycinek dziedziny uczenia głębokiego, która nieustannie rozwija się i poszerza.

Drugi możliwy podział dziedziny uczenia maszynowego to podział ze względu na formę nauki i informacje, na podstawie których model uczy się. Trzy główne grupy to uczenie nadzorowane, nienadzorowane oraz ze wzmocnieniem [24]. Algorytmy uczenia nadzorowanego obserwują dostępne pary wejścia-wyjścia i na tej podstawie uczą się sposobu mapowania informacji wejściowych na wyjściowe [24]. Kluczem uczenia nadzorowanego jest fakt, że w procesie uczenia dostępne są oczekiwane odpowiedzi modelu (nazywane w języku angielskim ground truth). Przykładem może być klasyfikacja obrazów. Algorytmy (głównie głębokie sieci neuronowe) na podstawie obrazów z etykietami uczą się rozróżniać kategorie. W uczeniu nienadzorowanym algorytm uczy się wzorców występujących w danych wejściowych bez wyraźnej informacji zwrotnej dotyczącej poprawności tych wzorców [24]. Przykładowymi typami algorytmów nienadzorowanego uczenia są klasteryzacja i algorytmy zmniejszania wymiarowości danych. Ostatnia kategoria - uczenie ze wzmocnieniem - obejmuje algorytmy, które w procesie uczenia są karane lub wynagradzane według konkretnej strategii. W przeciwieństwie do uczenia nadzorowanego, w tym przypadku model nie dostaje informacji zwrotnej dotyczącej poprawności danego wyniku, a o samej poprawności rozumuje na podstawie przeszłych nagród oraz kar [24].

Wszystkie opisane dotychczas algorytmy uczenia maszynowego należą do grupy popularnych i konwencjonalnych metod stosowanych w szerokiej gamie zastosowań. Algorytmy te są implementowane w najpopularniejszych bibliotekach numerycznych i uczenia maszynowego. Niemniej jednak, oprócz tych konwencjonalnych algorytmów uczenia maszynowego, istnieją także mniej popularne rozwiązania, które również często uzyskują bardzo dobre wyniki w różnorodnych zadaniach. Ich wykorzystanie pozwala spojrzeć na rozwiązywane problemy z innej perspektywy i potencjalnie opracować jeszcze lepsze rozwiązania pod względem dokładności, wydajności, umiejętności generalizacji itp. Przykładem takiego modelu jest Losowa Sieć Neuronowa, która stała się przedmiotem części badań przeprowadzonych na potrzeby niniejszej pracy

doktorskiej. Z tego powodu, postanowiono w dużym skrócie opisać ten rodzaj sieci, ponieważ jest ona znacznie mniej znana niż sieci neuronowe głównego nurtu.

Losowa Sieć Neuronowa (ang. Random Neural Network, RNN) została zaproponowana przez E. Gelenbego w pracy [40]. Jest to sieć neuronowa, w której krążą sygnały dodatnie i ujemne. Sieci te zostały zastosowane do wielu problemów klasyfikacyjnych - zarówno do zadań binarnych, jak i wieloklasowych. Zostały one z powodzeniem zastosowane w dziedzinach segmentacji obrazów medycznych [41], klasyfikacji pojazdów [42], symulacji rozszerzonej rzeczywistości systemów transportowych [43], określania stanu serwerów w dynamicznym środowisku sieci chmurowej [44] i wykrywania ataków [45, 46]. Udowodniono, że mogą one przewyższyć inne metody w wielu zadaniach [44, 47, 48, 49]. Również różnorodne modyfikacje sieci RNN mogą skutecznie wykonywać zadania klasyfikacji [48]. W pracy [44] klasyfikator oparty na wielowarstwowych gęstych klastrach RNN został wykorzystany do określenia stanu serwerów w dynamicznym środowisku chmurowym. Zaproponowano również różnorodne techniki uczenia sieci. W kontekście inicjalizacji wag, w poprzednich pracach początkowe wartości wag były zazwyczaj wybierane losowo [42, 50] lub z użyciem alternatyw wymagających dużych nakładów obliczeniowych [50, 51, 52]. Niemniej jednak, brakuje prac skierowanych na bardziej intuicyjne i zrozumiałe strategie. Różne procedury dotyczące samego trenowania zostały zaproponowane dla tego typu sieci, między innymi oparte na metodzie najszybszego spadku [50] lub Levenberga-Marquardta [53]. Niemniej jednak, podejścia skierowane na uproszczenie tych algorytmów nie są powszechne. W obszarach ograniczonych zasobami, takich jak IoT, korzystne byłoby wprowadzenie mniej wymagających obliczeniowo i pamięciowo rozwiązań. Dlatego też niniejsza praca ma na celu wprowadzenie (1) intuicyjnego sposobu inicjalizacji sieci oraz (2) metody, która pozwala na zmniejszenie liczby kosztownych operacji obliczeniowych wykonywanych podczas trenowania sieci metodą najszybszego spadku. Metoda ta ma również pozytywny wpływ na wykorzystanie pamięci podczas fazy predykcji, ponieważ pozwala przechowywać mniej parametrów modelu. Opis metody wraz z badaniami przedstawiono w Sekcji 3.3. Zaproponowaną modyfikację sieci zastosowano w dwóch zadaniach związanych z poprawą bezpieczeństwa systemów informatycznych: do wykrywania podatności oprogramowania oraz do wykrywania ataków sieciowych z wykorzystaniem infrastruktury botnet.

## 2.2 Wyjaśnialność systemów inteligentnych

Koncept wyjaśnialności (ang. explainability) systemów inteligentnych odnosi się do możliwości tłumaczenia ludziom w zrozumiały sposób ich rezultatów oraz sposobu, w jaki te rezultaty zostały zwrócone (przebieg procesu decyzyjnego). Jest to kluczowy aspekt w kontekście sztucznej inteligencji. Wyjaśnialność jest istotna zarówno z perspektywy społecznej, jak i technologicznej, ponieważ jej niewystarczający stopnień wewnątrz domeny sztucznej inteligencji ma negatywny wpływ na zaufanie społeczeństwa dla rozwiązań inteligentnych [7]. Z tego też powodu powstała poddziedzina sztucznej inteligencji nazywana Wyjaśnialną Sztuczną Inteligencją (ang. Explainable AI, XAI), wewnątrz której istotnym aspektem jest interpretowalność. W uproszczeniu, interpretowalność i wyjaśnialność można rozróżnić w następujący sposób: wyjaśnialność skupia się na zrozumiałych rezultatach metod, a interpretowalność na rozumieniu modeli i ich procesu uzyskiwania rezultatów. Niemniej jednak, pojęcia te są ze sobą ściśle związane i przenikają się (wyjaśnialność można uznawać za szerszy koncept zawierający interpretowalność, stąd wykorzystanie właśnie tej nazwy w tytule pracy). W pracy zwrócono uwagę na podstawowy aspekt wyjaśnialności - interpretowalność modeli uczenia głębokiego. Zaproponowano również szereg metod praktycznej wyjaśnialności, co rozumiane jest w pracy jako metody, których rezultaty i działanie można w prosty sposób wytłumaczyć lub nawet wizualizować, co uprasza w dużym stopniu wykorzystanie i odbiór tych metod. Są to metody, których kluczowym aspektem jest intuicyjność oraz prostota interpretacji wyników.

### 2.2.1 Interpretowalność modeli uczenia maszynowego

Ważnym aspektem algorytmów uczenia maszynowego jest ich interpretowalność. Trudno jest matematycznie zdefiniować interpretowalność. Także w przypadku niematematycznych definicji możliwy jest wybór między różnymi opcjami. Jedna z definicji, przedstawiona w pracy [54], brzmi następująco: „Interpretowalność to stopień z jakim człowiek potrafi zrozumieć przyczynę podjętej decyzji". Alternatywną definicją może być: „Interpretowalność to stopień z jakim człowiek potrafi konsekwentnie przewidywać wyniki modelu" [55]. Pod względem interpretowalności, istnieje podział algorytmów uczenia maszynowego na „czarne skrzynki" (ang. black box) i „białe skrzynki" (ang. white box). Do grupy algorytmów typu „biała skrzynka" należą takie algorytmy jak regresja liniowa oraz drzewa decyzyjne. W przypadku tej grupy algorytmów, reguły klasyfikacji są przejrzyste i dobrze zdefiniowane, przez co w prosty sposób można wyjaśnić dlaczego zwróciły daną predykcję. W wielu przypadkach możliwe jest nawet wykonanie tych samych obliczeń co

w modelu ręcznie. Ta prostota umożliwia również niejednokrotnie naturalną wizualizację podejmowanych w modelu decyzji i jego reguł. Przykładem jest graficzna reprezentacja drzew binarnych, w której to kolejne węzły drzewa zawierają ustalone wartości cech zbioru danych, na których podstawie dokonywana jest decyzja dotycząca przejścia w drzewie aż do osiągnięcia etykiety (etykiety te znajdują się w liściach drzewa). Na podstawie cech znajdujących się w węzłach drzewa można w prosty sposób wnioskować o istotności poszczególnych cech w procesie podejmowania decyzji. Odwrotnością grupy algorytmów „białej skrzynki" są algorytmy typu „czarnej skrzynki". Są to takie modele jak sieci neuronowe, czy też algorytmy uczenia zespołowego (np. drzewa wzmacniane gradientowo, lasy losowe). Algorytmy te cechują się zazwyczaj lepszą dokładnością i głębią zrozumienia danego problemu. Niemniej jednak dzieje się to kosztem właśnie interpretowalności ich decyzji. Przykładem mogą być sieci konwolucyjne w zastosowaniach wizyjnych, takich jak rozpoznawanie obiektów na obrazach. Sieci te osiągają bardzo wysoką dokładność w tym zakresie, ale w przypadku rozpoznania konkretnego obiektu na obrazie wejściowym trudno jest wytłumaczyć na podstawie jakich cech obrazu dana predykcja została zwrócona. Wysoka skuteczność głębokich sieci neuronowych jest rezultatem olbrzymiej liczby przekształceń danych dokonywanych przez liczne warstwy sieci, co pozwala uzyskać wysoce abstrakcyjne i informatywne reprezentacje nawet wysokowymiarowych danych wejściowych. Niemniej jednak, te liczne przekształcenia matematyczne sprawiają, że nie da się w prosty sposób ustalić na jakiej podstawie sieć zwróciła taką odpowiedź a nie inną. Nasuwa się więc pytanie: skoro algorytmy typu „czarnej skrzynki" dają tak dobre rezultaty, to jaki jest cel rozpatrywania takich aspektów jak interpretowalność? Możliwość interpretacji i inspekcji procesu decyzyjnego jest najbardziej istotna w przypadku błędnych predykcji w celu wyjaśnienia przyczyn takich wyników. Jest to istotne dla tworzenia odpornych na błędy systemów o wysokiej jakości. Są to kluczowe właściwości z perspektywy bezpieczeństwa, w której to inspekcja błędów jest zazwyczaj wymaganiem. Z drugiej strony ludzie analogicznie do prób zrozumienia własnego procesu analizy świata i decyzyjnego, również dążą do zrozumienia działania swoich tworów. Niska wyjaśnialność w dużym stopniu wpływa na niskie zaufanie społeczeństwa w stosunku do rozwiązań inteligentnych i poprawa tego aspektu wydaje się być kluczowa dla dalszego rozwoju dziedziny sztucznej inteligencji i rozpowszechnienia jej algorytmów w codziennych rozwiązaniach [7].

### 2.2.2 Metody wyjaśnialności dla modeli uczenia głębokiego

Podjęto wiele prób wizualizacji wewnętrznych reprezentacji obrazu nauczonych przez sieci konwolucyjne. Reprezentacje uzyskane przez warstwy konwolucyjne nazywane są konwolucyjnymi mapami cech. Odzwierciedlają one reakcję sieci na określone wzorce w danych - wyrażoną poprzez aktywację określonych regionów. Najprostszym sposobem wizualizacji aktywacji sieci konwolucyjnej dla danego obrazu jest oddzielna wizualizacja map cech. Takie podejście zastosowano np. w pracy [56]. Podejście to nie jest praktyczne, ponieważ zwykle z ostatniej warstwy konwolucyjnej uzyskuje się kilka tysięcy map cech. Jako alternatywę można zastosować różne metody wizualnego skupienia (ang. visual attention). Mają one na celu wyjaśnienie, które regiony obrazu miały największy wpływ na decyzję sieci. W przypadku rozpoznawania obiektów na poziomie obrazu są to regiony, które spowodowały zwrócenie określonej klasy jako predykcji. Można je wizualizować na podstawie błędu propagacji wstecznej [27, 57]. Metoda Class Activation Mapping (CAM) [58] wykorzystuje warstwę Global Average Pooling (GAP) do tworzenia map uwagi dla określonej klasy. Grad-CAM - Gradient-weighted Class Activation Mapping [59] - jest rozszerzeniem CAM dla sieci niewykorzystujących GAP. Eigen-CAM [60] wykorzystuje składowe główne wyuczonych reprezentacji z warstw konwolucyjnych do wizualizacji. Autorzy LayerCAM [61] ponownie rozważają związek między gradientami sieci a mapami cech. Uncertainty Class Activation Map (U-CAM) [62] to wizualizacja zależna od klasy oparta na szacunkach pewności opartych na gradientach. Full Resolution Class Activation Maps (F-CAM) [63] to ogólna metoda skalowania CAM. W przeciwieństwie do opisanych mechanizmów, proponowany mechanizm NAM koncentruje się na zdolności konwolucyjnego ekstraktora cech do dostarczania informacyjnych reprezentacji obrazu, a nie na wyjaśnianiu decyzji podejmowanych przez klasyfikator CNN. Daje to ogólny przegląd aktywacji sieci na danym obrazie i wydaje się być przydatne do inspekcji sieci konwolucyjnych, gdy planowane jest ich użycie w różnych zadaniach (np. do wykrywania obiektów). Wydaje się to również korzystne dla analizy w przypadku ataków adwersarzy, ponieważ może potencjalnie pokazać, czy atak niszczy wewnętrzne reprezentacje obrazu, a nie tylko zmienia przewidywania. W przeciwieństwie do większości sposobów wizualizacji jest to metoda, która nie wykorzystuje gradientów sieci.

### 2.2.3 Intuicyjność metod a praktyczna wyjaśnialność

Drugim aspektem wyjaśnialności systemów inteligentnych jest ich praktyczna wyjaśnialność, co w pracy odnosi się do tworzenia i wykorzystania metod oraz technik, które są łatwo zrozumiałe, intuicyjne i użyteczne dla ludzi, w tym również użytkowników nietechnicznych, w celu zrozumienia i wyjaśnienia zachowania i decyzji systemów inteligentnych. Obejmuje ona szereg strategii mających na celu uczynienie systemów sztucznej inteligencji bardziej przejrzystymi, interpretowalnymi i godnymi zaufania społeczeństwa. Oto kilka kluczowych aspektów i elementów, które przyczyniają się do praktycznej wyjaśnialności sztucznej inteligencji

Pierwszym aspektem jest tworzenie intuicyjnych metryk (zarówno tych wykorzystywanych w systemach inteligentnych, jak i metryk do oceny działania tych systemów) oraz metod. Tego typu metryki są z zasady łatwe do zinterpretowania i zrozumiałe dla ludzi (również dla tych mniej związanych z techniką). Metryki te powinny zapewniać jasny opis wydajności modelu, jego zachowania i procesu przetwarzania danych. W tym obszarze na potrzeby pracy zaproponowano dwie metryki. Pierwsza z nich - **metryka do opisu szkodliwości ataków na sieci neuronowe** - kładzie nacisk na prostotę wykorzystania, intuicyjność rezultatów oraz ich informatywność. Metryka jest również bardzo wszechstronna i oprócz oceny działania modeli podczas ataków, może być również wykorzystywana wraz ze standardową metryką ogólnej dokładności do dokładniejszej oceny dokładności modeli. Druga metryka, a w zasadzie **zestaw metryk do filtrowania ruchomych użytkowników** na potrzeby systemu lokalizacji użytkowników z telefonami w pomieszczeniach, kładzie nacisk na intuicyjność rezultatów i możliwość ich wizualizacji nawet dla nietechnicznych użytkowników, którzy mogą być docelowymi użytkownikami tego typu systemu. W tym miejscu można wyszczególnić drugi aspekt praktycznej wyjaśnialności - możliwość wizualizacji rezultatów metod. Do tego obszaru można również zaliczyć **zaproponowaną wizualną metodę interpretowania działania modeli sieci konwolucyjnych** (Network Activation Mapping). Także **przedstawiona nowa metoda ataku na sieć neuronową** jest intuicyjna i prosta w interpretacji, ponieważ jej celem jest saturacja wewnętrznych reprezentacji możliwymi wzorcami w celu „zakrycia" prawdziwych wzorców, a więc „oślepienie" sieci neuronowej. Kluczowym aspektem wszystkich prezentowanych metryk i metod jest prostota działania, interpretacji oraz szeroko rozumiana przyjazność dla użytkownika.

Kolejny aspekt praktycznej wyjaśnialności obejmuje intuicyjne formy inicjalizacji wag modeli uczenia maszynowego. Inicjalizacja wag modeli uczenia głębokiego jest istotna z punktu widzenia efektywnego uczenia tych algorytmów. Przykładem może być popularna metoda inicjalizacji wag Glorota z

wykorzystaniem rozkładu jednostajnego (ang. Glorot uniform) [64], wprowadzona w celu zapewnienia efektywnej propagacji informacji wewnątrz sieci neuronowej do przodu oraz wstecz. Oprócz samego aspektu wydajności, w tym obszarze ważne wydaje się być również wykorzystanie technik inicjalizacji, które promują neutralność modelu i zmniejszają jego stronniczość na początku trenowania. Celem jest zapewnienie, aby model nie zaczynał swojego procesu trenowania od obecności silnych uprzedzeń, które mogą wpływać na jego proces uczenia się. Stronniczość (ang. bias) jest poważnym problemem sieci neuronowych i jego obecność wywiera wpływ na działanie rzeczywistych systemów opartych na algorytmach sztucznej inteligencji. Jako przykład można podać problem znaleziony w firmie Amazon. W 2015 roku firma odkryła, że ich system do rekrutacji pracowników był stronniczy względem kobiet. Narzędzie było wytrenowane na danych historycznych, w których zdecydowaną większość pracowników branży technicznej stanowili mężczyźni, przez co byli faworyzowani w procesie rekrutacji [65]. Tak jak w tym przypadku, stronniczość jest zazwyczaj wprowadzana przez dane wykorzystane w procesie trenowania, niemniej jednak nieodpowiednia inicjalizacja wag również może ją wprowadzać. Z tego powodu w pracy postanowiono skupić się na tym aspekcie i **zaproponowano metodę inicjalizacji wag dla modelu Losowej Sieci Neuronowej**, która zapewnia neutralność sieci na początku trenowania.

Intuicyjność i prostota metod jest również ważnym aspektem projektowania metod zorientowanych na użytkownika. W tym obszarze priorytetowo powinno się traktować potrzeby i poziom zrozumienia interesariuszy końcowych, zwłaszcza tych nietechnicznych, tak aby końcowe wartości i ich prezentacja były dostosowane do doświadczenia i wiedzy odbiorców. Takie podejście do projektowania systemów inteligentnych wydaje się kluczowym aspektem dla wzrostu zaufania społeczeństwa do sztucznej inteligencji, co powinno być celem rozwijającej ją społeczności [7].

## 2.3 Bezpieczeństwo systemów inteligentnych

Sztuczna inteligencja (AI) zyskała w ostatnich latach ogromną popularność, ułatwiając zautomatyzowane podejmowanie decyzji w szerokim zakresie zadań i obszarów zastosowań. Obecnie można zauważyć powszechne łączenie różnych technologii (np. Internetu Rzeczy, robotyki, czujników itp.) oraz rosnącej ilości i różnorodności danych w celu wykorzystania sztucznej inteligencji na dużą skalę [18]. W kontekście cyberbezpieczeństwa sztuczna inteligencja jest stosunkowo nową domeną o wysokim stopniu złożoności, co stwarza nowe wyzwania. Choć niewątpliwie przynosi ona korzyści, to nie należy po-

mijać faktu, że sztuczna inteligencja i jej zastosowanie, może narazić osoby i organizacje na nowe, czasem nieprzewidywalne i trudne do przezwyciężenia zagrożenia. Są to zagrożenia, które nie są w pełni zrozumiane i znane ze względu na młody wiek tej technologii [18]. Jest to szczególnie istotne w przypadku zautomatyzowanego podejmowania decyzji we wdrożeniach o krytycznym znaczeniu dla bezpieczeństwa, takich jak pojazdy autonomiczne lub inteligentna produkcja. Sztuczna inteligencja otwiera również nowe ścieżki w zakresie metod i technik ataków oraz stwarza nowe wyzwania w zakresie ochrony danych. Z drugiej jest źródłem nowych możliwości automatyzacji poprawy bezpieczeństwa (np. poprzez wykrywanie zagrożeń za pomocą metod uczenia maszynowego) [18]. Z tego powodu, dziedzina na pograniczu bezpieczeństwa i sztucznej inteligencji jest jednym z kluczowych wyzwań dzisiejszej informatyki. Potwierdza to aktualność i ważność zagadnień, wokół których koncentruje się tematyka niniejszej pracy doktorskiej.

### 2.3.1 Podwójnie dualna natura algorytmów sztucznej inteligencji

Krajobraz bezpieczeństwa sztucznej inteligencji jest niezwykle trudny do zmapowania [18]. W celu wyjaśnienia dlaczego tak się dzieje, w pracy postanowiono wprowadzić zagadnienie podwójnie dualnej natury algorytmów sztucznej inteligencji. Pierwszym aspektem dualności jest fakt, że systemy inteligentne są z jednej strony systemem informatycznym takim jak każdy inny, przez co dotykają je te same zagrożenia jak tradycyjne systemy (np. ataki sieciowe, podatności oprogramowania), a jednocześnie, ze względu na naturę algorytmów sztucznej inteligencji, systemy te są narażone również na dodatkowe, dedykowane zagrożenia (np. modele uczenia głębokiego na ataki adwersarza). Natura algorytmów sztucznej inteligencji sprawia również, że systemy na nich oparte są nawet bardziej narażone na tradycyjne zagrożenia bezpieczeństwa, ponieważ często wymagają one implementacji skomplikowanych metod matematycznych (przez co prościej jest wprowadzić podatności oprogramowania w kodzie) oraz dużej liczby zasobów (przez co np. konieczne jest wykonywanie obliczeń w chmurach obliczeniowych i częsta wymieniana informacji - może to zwiększać podatność na ataki). Z tego powodu w pracy podjęto analizę dwóch kluczowych aspektów bezpieczeństwa systemów inteligentnych:

1. bezpieczeństwa systemów inteligentnych jako tradycyjnych systemów informatyki,
2. dedykowanych zagrożeniach systemów inteligentnych.

Można również zdefiniować drugi aspekt dualności bezpieczeństwa w kon-

tekście systemów inteligentnych. Z jednej strony systemy inteligentne są w dużym stopniu narażone na zagrożenia bezpieczeństwa. Z drugiej, są one coraz częściej wykorzystywane do tworzenia rozwiązań na potrzeby poprawy bezpieczeństwa innych systemów (np. do wykrywania ataków, czy też podatności) [19]. Nie znaczy to jednak, że nie powinno się wykorzystywać tych systemów do budowania rozwiązań wspierających bezpieczeństwo innych systemów. Znaczy to, że aby zapewnić coraz to lepsze bezpieczeństwo systemów informatycznych (zarówno inteligentnych, jak i tradycyjnych) powinno się równolegle pracować nad poprawą bezpieczeństwa algorytmów sztucznej inteligencji oraz rozwijać i proponować nowe rozwiązania z zakresu wykorzystania algorytmów inteligentnych do wykrywania zagrożeń bezpieczeństwa, ponieważ algorytmy te wykazują nieocenioną wartość ze względu na swoją moc znajdowania wzorców (czasem bardzo skomplikowanych) w danych. Również, algorytmy uczenia maszynowego mogą posłużyć do poprawy bezpieczeństwa samych siebie - na przykład w przypadku opracowania dokładnych algorytmów detekcji podatności oprogramowania oraz ataków sieciowych, jako że modele uczenia maszynowego także są budowane za pośrednictwem kodu (kod ten może zawierać podatności), a systemy je goszczące często przesyłają między sobą informacje (są podatne na ataki sieciowe). Dodatkowo, analiza obu tych aspektów pozwala lepiej zrozumieć problemy i wyzwania domeny bezpieczeństwa systemów inteligentnych poprzez kompleksowe podejście do tego tematu. Z tego powodu, w pracy zaproponowano rozwiązania z zakresu:

1. poprawy bezpieczeństwa systemów informatycznych za pośrednictwem algorytmów uczenia maszynowego (wykrywanie ataków i podatności w kodzie),
2. poprawy bezpieczeństwa algorytmów uczenia maszynowego.

Zarysowana w tej sekcji podwójna dualność jest jedynie uproszczeniem krajobrazu bezpieczeństwa sztucznej inteligencji na potrzeby przeanalizowanych w pracy aspektów. Oprócz wykrywania zagrożeń oraz bycia ich celem, algorytmy sztucznej inteligencji mogą być również wykorzystywane do samego generowania zagrożeń (np. ataków) [19]. Tak naprawdę, dziedziny sztucznej inteligencji i bezpieczeństwa łączy sieć wzajemnych zależności [18].

### 2.3.2 Tradycyjne zagrożenia systemów informatycznych

W niniejszej sekcji opisano tradycyjne zagrożenia bezpieczeństwa systemów informatycznych. Są to ryzyka obecne we wszystkich systemach informatycznych, niezależnie od tego, czy wykorzystują one algorytmy sztucznej inteligencji, czy też nie. Do tych zagrożeń zaliczono ataki sieciowe i podatności oprogramowania.

### Ataki sieciowe

Szybki rozwój technologii związanych z siecią Internet sprawia, że równie szybko rozwija się krajobraz zagrożeń związanych z tą dziedziną. Główne zagrożenia związane z sieciami to ataki. Istnieje szeroka gama ataków celujących w różne słabe punkty systemów informatycznych. Europejska Agencja ds. Bezpieczeństwa Sieci i Informacji (ang. European Union Agency for Cybersecurity, ENISA) przewiduje, że w najbliższych latach zagrożenia bezpieczeństwa będą trudniejsze do wykrycia i zbadania z powodu ciągle rosnącej złożoności powierzchni ataków [66]. Niektóre z zagrożeń cyberbezpieczeństwa z zakresu ataków to ataki fizyczne, ataki sieciowe, ataki wykorzystujące oprogramowanie i ataki dotyczące szyfrowania danych [67].

Ataki fizyczne obejmują między innymi zakłócanie kanałów przesyłu danych lub ataki mające na celu obciążenie urządzenia i wyczerpanie jego baterii [67]. Kolejnym atakiem tego typu jest stała blokada usług (ang. Permanent Denial of Service lub phlashing) obejmująca fizyczne zniszczenie atakowanego urządzenia lub załadowanie BIOSu z wykorzystaniem złośliwego oprogramowania [68].

Ataki dotyczące szyfrowania danych obejmują ataki oparte na analizie kryptograficznej oraz ataki kanału bocznego (ang. Side-Channel attacks). Ataki kanału bocznego wykorzystują wiedzę dotyczącą fizycznej implementacji systemu w celu zdobycia informacji. Ataki dotyczące danych [68] mają na celu zaburzenie integralności danych oraz spowodowanie ich niespójności, nieautoryzowane dostępy do danych i ich naruszenia (ang. Data Breach).

Ataki sieciowe obejmują ataki mające na celu zbieranie informacji przesyłanych za pośrednictwem internetu oraz modyfikację tych informacji. Przykładem może być spoofing RFID (ang. Radio Frequency Identifiction). Atak polega na podmienieniu przez atakującego RFID prawidłowego urządzenia w celu przechwycenia kierowanego do tego urządzenia ruchu sieciowego. Ataki sieciowe mogą także obierać za cel zmianę routingu w sieci lub spowodowanie przesyłania wewnątrz sieci jedynie fragmentarycznych informacji. Kolejnym przykładem są ataki typu wormhole, w których tuneluje się pakiety między węzłami za pośrednictwem łącza o niskiej przepustowości.

Niezwykle istotnymi atakami sieciowymi ze względu na swoją powszechność są ataki typu blokada usług (ang. Denial of Service, DoS) oraz rozproszona blokada usług (ang. Distributed Denial of Service, DDoS). Ich celem jest zablokowanie zasobów i usług serwera dla prawowitych użytkowników. Przykładem tego typu ataków jest Powódź (ang. flood) TCP SYN [69]. W ataku tym wykorzystywany jest mechanizm handshake protokołu TCP. Zapytania są wysyłane szybciej niż serwer jest w stanie je przetworzyć, co prowadzi do blokady usług. Podobnym atakiem tego typu jest Powódź UDP [70]. W

tym przypadku, losowe porty docelowego urządzenia są „zalewane" datagramami UDP. Urządzenie szuka aplikacji odpowiadających tym portom i po nieudanej próbie odsyła komunikat o niepowodzeniu. Wiele takich akcji powoduje blokadę usług. Kolejnym atakiem DDoS jest Powódź HTTP [71]. W ataku tym wykorzystywane są zapytania HTTP POST i GET. Efektywność tego typu ataku polega na zrozumieniu specyfiki i działań wykonywanych przez dany serwer. Ataki wykorzystujące HTTP POST są konstruowane w taki sposób, aby uruchamiać na serwerze atakowanym złożone i obciążające operacje, np. obliczenia. Ataki opierające swoje działanie na HTTP GET są prostsze w konstrukcji i lepiej skalowalne na większą liczbę atakujących urządzeń. Opisane ataki są klasycznymi atakami blokady usług, ale istnieją także mniej standardowe procedury - np. ataki typu slow-rate, takie jak Slowloris [72]. Atak ten wykorzystuje fragmentaryczne zapytania HTTP blokujące dany port jak najdłużej się da (wysyłane są nagłówki bez kontynuacji wiadomości). Atak tego typu jest trudny do wykrycia dla systemów wykrywania ataków, ze względu na wykorzystanie częściowych, a nie zniekształconych zapytań. Do grupy ataków DDoS zalicza są też ataki wzmacniające (ang. Amplification Attacks), które nazywa się również atakami metodą odbicia lustrzanego (ang. Reflection Attacks). Ataki typu DDoS są szczególnie istotne w popularnej w ostatnich latach domenie Internetu Rzeczy (ang. Internet of Things, IoT). Proste, często niezabezpieczone lub słabo zabezpieczone urządzenia IoT są często przechwytywane przez kryminalistów w celu włączenia ich do specjalnych sieci, nazywanych botnetami. Sieci botnet są wykorzystywane do przeprowadzania ataków DDoS na wyznaczone cele, najczęściej bez wiedzy ich użytkowników. Czyni to je jeszcze bardziej niebezpiecznymi niż inne typy ataków. Z tego powodu konieczne jest dalsze badanie ataków tego typu i proponowanie rozwiązań będących w stanie wykrywać ataki przeprowadzane z wykorzystaniem infrastruktury botnet i DDoS w celu poprawy bezpieczeństwa sieci internet. W obszarze IoT istotna jest również wydajność tych rozwiązań, aby możliwe było ich wykorzystywanie nawet na ograniczonych urządzeniach Internetu Rzeczy. Z tego powodu, w pracy **zaproponowano metodę do wykrywania ataków sieciowych wykorzystujących infrastrukturę botnet**, która wykorzystuje specjalną modyfikację Losowych Sieci Neuronowych ograniczającą liczbę kosztownych operacji koniecznych do trenowania oraz pamięci koniecznej do przechowywania jej parametrów w stosunku do bazowego rozwiązania.

### Metody wykrywania ataków sieciowych

Wiele różnych podejść zostało wykorzystanych do stworzenia systemów wykrywania ataków. Wykrywanie oparte na wzorcach (ang. signature-based)

[73, 74] wykorzystuje znane wzorce ataków do ich identyfikacji. Ten typ wykrywania nie ma silnej mocy generalizacji i nie jest efektywny w wykrywaniu nowych typów ataków [75]. Systemy oparte na wykrywaniu anomalii działają na zasadzie identyfikacji wzorców definiujących ruch normalny i nieprawidłowy. Dlatego też są one bardziej adaptacyjne niż systemy bazujące na wzorcach. Takie systemy można podzielić na trzy główne grupy: oparte na wiedzy, statystyczne i oparte na uczeniu maszynowym. Systemy oparte na wiedzy wykorzystują zazwyczaj zestawy reguł i skończone maszyny stanów. Techniki oparte na statystykach [76] bazują np. na analizie szeregów czasowych i ich właściwości. Podejście oparte na uczeniu maszynowym obejmuje między innymi klasteryzację [77], algorytmy genetyczne [78]. W dziedzinie wykrywania ataków, powszechne są również podejścia oparte na uczeniu głębokim [14, 15, 17, 79]. Wykrywanie ataków można traktować jako zadanie klasyfikacji: wieloklasowej lub binarnej. W przypadku systemów wykrywania ataków, klasyfikacja binarna oznacza zazwyczaj oznaczenie ruchu sieciowego jako normalnego lub nieprawidłowego. Ataki mogą być również klasyfikowane do wielu klas w jednym modelu. Nie jest to jednak zalecane, zwłaszcza dla uczenia płytkiego, ponieważ wzrost liczby klas w zadaniu klasyfikacji wieloklasowej może prowadzić do spadku dokładności i wymaga zazwyczaj stosowania większych modeli [80]. Większość prac w powiązanej literaturze stosuje do wykrywania ataków algorytmy z kanonu standardowych modeli uczenia maszynowego (drzewa decyzyjne, lasy losowe, maszyny wektorów nośnych, czy też sieci rekurencyjne). Niestandardowe rozwiązania oferują inne spojrzenie na problem, prezentując potencjał do poprawy wyników. Z tego powodu konieczne jest testowanie i wprowadzanie nowych algorytmów w obszarze wykrywania ataków sieciowych. W niniejszej pracy **zaproponowano metodę do wykrywania ataków opartą na autorskiej modyfikacji Losowej Sieci Neuronowej**. Bazowa wersja metody na przestrzeni lat okazała się efektywnym rozwiązaniem o szerokim spektrum zastosowania [42, 43, 44], również w obszarze bezpieczeństwa [45, 46].

### Luki bezpieczeństwa

Lukę bezpieczeństwa/podatność można zdefiniować za pośrednictwem standardu ISO/IEC 27000:2018 [81] jako „słabość zasobu lub kontroli, która może zostać wykorzystana przez jedno lub więcej zagrożeń". Niewykryte luki w zabezpieczeniach pociągają za sobą znaczne koszty utrzymania [82] i mogą powodować potencjalne naruszenie zasad bezpieczeństwa [83].

Common Weakness Enumeration (CWE) [84] i Common Vulnerabilities and Exposures (CVE) [85] to dwa najważniejsze standardy wykorzystywane do identyfikacji i opisu luk bezpieczeństwa. Oba standardy są utrzymywane

i rozwijane przez MITRE - amerykańską organizację non-profit. CWE definiuje typy podatności, a CVE definiuje konkretne instancje w produkcie lub systemie. Przykładem typu CWE może być CWE-119 [86] - Niewłaściwe ograniczenie operacji w granicach bufora pamięci (ang. Improper Restriction of Operations within the Bounds of a Memory Buffer). Instancją CWE-119 jest na przykład CVE-2022-26763 [87] - podatność występująca w niektórych produktach firmy Apple. Ze względu na błąd dostępu do bufora pamięci poza granicami, złośliwa aplikacja mogła wykonać dowolny kod z uprawnieniami systemowymi. Problem ten został rozwiązany dzięki ulepszonemu sprawdzaniu granic bufora pamięci.

Oprócz standaryzacji, w celu ułatwienia wiedzy na temat oprogramowania, powstały organizacje zajmujące się bezpieczeństwem, takie jak Computer Emergency Response Team Coordination Center (CERT/CC) [88], Open Web Application Security Project (OWASP) [89] i SANS Institute [90]. Organizacje społeczne i rządowe tworzą publiczne repozytoria podatności (np. National Vulnerabilities Database (NVD) [91] i CVE Details [92]), rankingi (np. CWE TOP25 [13], OWASP Top10 [93]) oraz wytyczne dotyczące tworzenia bezpieczniejszych aplikacji (np. OWASP Secure Coding Practices Guide [94]). Pomimo tych wysiłków, luki bezpieczeństwa są nadal powszechne i niosą ze sobą poważne konsekwencje. Według danych Veracode [95], ponad 85% aplikacji przeskanowanych za pomocą ich platformy zarządzania bezpieczeństwem oprogramowania od 1 kwietnia 2017 r. do 31 marca 2018 r. zawierało co najmniej jedną podatność. Według jedenastego tomu corocznego raportu State of Software Security (SOSS) firmy Veracode [12], aplikacje oparte na językach C++ i PHP najczęściej spośród wszystkich badanych zawierały błędy o wysokiej i bardzo wysokiej wadze. Było to 59 procent dla aplikacji napisanych w języku C++ i 53 procent dla aplikacji napisanych w języku PHP. Przyspieszenie procesu wytwarzania oprogramowania, ograniczone zasoby testowe i brak wiedzy na temat bezpieczeństwa uniemożliwiają znalezienie i naprawienie wszystkich luk oraz zapobieganie ich wykorzystaniu. Dlatego tak ważne jest wprowadzenie nowych metod, strategii i wytycznych, które pomogłyby w poprawie bezpieczeństwa oprogramowania. Równie istotna jest analiza oprogramowania napisanego w językach z rodziny C/C++ z powodu ich szerokiego zastosowania do budowania podstawowych elementów systemów informatycznych (systemów operacyjnych, maszyn wirtualnych), a także wydajnych bibliotek uczenia maszynowego. Z tego powodu na potrzeby pracy **przeprowadzono dogłębną analizę znanych luk w oprogramowaniu TensorFlow** (biblioteka uczenia głębokiego napisana w C++) oraz **sprawdzono możliwość wykrywania tych podatności za pośrednictwem dostępnych narzędzi open-source do analizy statycznej**. **Zaproponowano również dwa rozwiązania z zakresu wykrywania po-**

**datności w oprogramowaniu C/C++.**

### Analiza i wykrywanie luk oprogramowania

**Główne podejścia do analizy podatności** Tradycyjne podejścia do wykrywania podatności można podzielić na dwie główne grupy: analizę statyczną i analizę dynamiczną [96]. Możliwe są także podejścia hybrydowe wykorzystujące zalety poszczególnych podejść i mitygujące ich wady. Analiza statyczna (znana również jako analiza kodu) jest zwykle przeprowadzana podczas weryfikacji kodu (testowanie białoskrzynkowe). Pomimo że wielu badaczy podejmuje wysiłki w celu ułatwienia przeprowadzania analizy statycznej w oparciu o wiele różnych podejść, to walidacja bezpieczeństwa oprogramowania stanowi wyzwanie [82]. W przypadku drugiego rodzaju analizy - analizy dynamicznej - niezbędna jest wykonywalna wersja programu. Tego typu analiza polega najczęściej na skanowaniu aplikacji w trakcie jej wykonywania w celu znalezienia luk w zabezpieczeniach. Ponieważ analiza dynamiczna wymaga wystarczającej liczby przypadków testowych do znalezienia luk, jest ona często bardzo czasochłonna [82]. Zróżnicowany charakter tych dwóch rodzajów analizy sprawia, że dobrym podejściem jest wykorzystywanie obu z nich na różnych etapach Cyklu Życia Oprogramowania (ang. Software Development Life Cycle, SDLC) w celu zwiększenia prawdopodobieństwa stworzenia bezpiecznego oprogramowania [97]. Niektórych typów podatności nie da się wykryć przed uruchomieniem programu, a inne wymagają analizy statycznej [95]. Analiza statyczna może być wprowadzona na wczesnych etapach cyklu życia oprogramowania i bada cały kod aplikacji, w przeciwieństwie do analizy dynamicznej, która koncentruje się jedynie na częściach wykonywanego kodu. Dlatego też konieczne jest włączenie analizy statycznej jako kluczowego elementu produkcji oprogramowania. Z tego powodu, w pracy **zaproponowano dwa rozwiązania wykorzystujące statyczną analizę kodu źródłowego do wykrywania podatności oprogramowania**. Jedno rozwiązanie jest oparte na dogłębnej analizie przydatności metryk statycznych analizatorów kodu oraz standardowych algorytmów uczenia maszynowego. W drugim rozwiązaniu zaproponowano natomiast system hybrydowy oparty na mieszanych cechach (tekst, metryki statycznego analizatora kodu) oraz sieci konwolucyjnej i autorskiej modyfikację Losowej Sieci Neuronowej (zastosowanej wcześniej do wykrywania ataków).

**Analiza luk w zabezpieczeniach i cech mogących służyć do ich przewidywania.** Analiza podatności może obejmować zarówno ich przyczyny, jak i charakterystykę oraz konsekwencje. Część prac koncentruje się na konsekwencjach podatności i ocenie ryzyka [98, 99]. Możliwe jest również przeana-

lizowanie przyczyn występowania podatności i jej potencjalnych wskaźników, np. cech uzyskanych z analizy statycznej. Takie cechy mogą być potencjalnie wykorzystane do stworzenia algorytmów przewidywania podatności. W literaturze można znaleźć wiele prac poświęconych ocenie różnych statycznych analizatorów kodu na przykładzie tradycyjnych produktów oprogramowania (np. [100, 101, 102, 103] dla C/C++), jednakże liczba prac poświęconych analizie przydatności generowanych przez nie cech do celów przewidywania podatności jest ograniczona. W [104, 105] przeprowadzono badania empiryczne z uwzględnieniem trzech aplikacji internetowych PHP o otwartym kodzie źródłowym. Autorzy oparli swoje badania na zbiorze danych i dwunastu metrykach wprowadzonych w pracy [106]. W pracy [107] przeprowadzono natomiast badanie empiryczne mające na celu sprawdzenie możliwości przewidywania ryzyka związanego z bezpieczeństwem aplikacji dla systemu Android w oparciu o 21 metryk opisujących kod uzyskanych przy użyciu programu SonarQube i 6 algorytmów uczenia maszynowego.

**Analiza podatności kodu związanych ze sztuczną inteligencją.** Liczne prace dotyczą wykrywania luk w oprogramowaniu w oparciu o modele uczenia maszynowego [16, 108, 109, 110], jednak prace dotyczące analizy podatności w samych produktach AI są bardzo ograniczone. W pracy [111] omówiono zagrożenia bezpieczeństwa platform uczenia głębokiego (TensorFlow, Caffe i Torch) na podstawie ich zależności (tj. bibliotek NumPy dla TensorFlow oraz opencv dla Caffe/Torch). W przeciwieństwie do proponowanego w niniejszej pracy podejścia, artykuł [111] nie bada zagrożeń znajdujących się bezpośrednio w kodzie źródłowym bibliotek uczenia głębokiego. Biorąc pod uwagę wszechobecność modeli uczenia głębokiego w różnych aplikacjach, jest to duża luka w literaturze. Ponadto oprogramowanie związane z uczeniem głębokim jest stosunkowo nowe i można oczekiwać przynajmniej pewnych zmian w praktykach programistycznych w stosunku do starszych, tradycyjnych produktów. Same prace dotyczące tradycyjnych produktów oprogramowania są bardziej powszechne (np. w pracy [112] autorzy przeanalizowali produkty tradycyjne: Linux Kernel, Xen oraz Mozilla). Ograniczeniem istniejących prac dotyczących podatności jest to, że analizują one głównie produkty tradycyjne. Prace te rozpatrują również podatności niezależnie od ich typu CWE lub analizują bardzo ograniczoną liczbę typów podatności. Nie koncentrują się one na ich znaczeniu i dotkliwości. Z tego powodu, na potrzeby pracy zdecydowano się zapełnić tę wyraźną lukę w literaturze i **przeprowadzono dogłębną analizę w produkcie bezpośrednio związanym z uczeniem głębokim i obliczeniami numerycznymi - TensorFlow**. Ze względu na swoją popularność oraz otwarte źródła, TensorFlow jest idealnym produktem

dla takiej analizy. W ramach analizy uwzględniono aż sześć typów podatności, zbadano je i opisano z różnych perspektyw oraz przetestowano możliwość ich wykrywania z wykorzystaniem dostępnych analizatorów statycznych kodu.

**Źródła danych wykorzystywane do statycznej analizy kodu i przewidywania podatności.** Analizę statyczną kodu stosuje się zazwyczaj od wczesnych etapów cyklu życia oprogramowania. Może ona bazować na analizie tekstu, metrykach oprogramowania, a także metrykach związanych z bezpieczeństwem (alertach Automatycznej Analizy Statycznej, ang. Automatic Static Analysis, ASA) [113]. Standardowe metryki oprogramowania mogą być używane do rozróżniania komponentów oprogramowania podatnych na zagrożenia i neutralnych [114]. Takie podejście wykorzystano w wielu pracach naukowych, między innymi stosując do tego celu metrykę złożoności (ang. complexity) [115], metrykę złożoności, sprzężenia i spójności (ang. complexity, coupling, cohesion) [116], a także metryki złożoności oraz opisujące zmiany kodu i aktywność programisty [83]. Alternatywą dla tych rozwiązań jest wykorzystanie alertów narzędzi do statycznej analizy kodu i bazujących na nich metrykach [117, 118]. Kolejnym powszechnym podejściem do przewidywania podatności oprogramowania jest analiza tekstu. W pracy [119] komponenty oprogramowania są reprezentowane jako lista wyekstrahowanych elementów „include" i wywołań funkcji. W pracy [120] wywołania funkcji zostały pobrane z abstrakcyjnych drzew składniowych (ang. Abstract Syntax Trees). Zastosowano również niektóre standardowe podejścia z domeny przetwarzania języka naturalnego. W pracy [16] komponenty oprogramowania były reprezentowane w postaci serii terminów. W podejściu tekstowym, można również uwzględnić kolejność tych terminów. Alternatywnie, komponenty można traktować jako Bag-Of-Words (czyli zbiór tokenów lub jednostkowych terminów) z powiązanymi częstotliwościami [121] lub za pomocą ważenia terminów na podstawie odwróconej częstotliwości występowania [122]. Wykorzystanie cech mieszanych również pojawia się w pracach, ale znacznie rzadziej niż stosowanie jednego typu cech. Przykłady takich prac to [123] (tekst + standardowe metryki oprogramowania), [124, 125] (metryki tradycyjne + metryki dotyczące rozwoju oprogramowania i jego bezpieczeństwa). Ze względu na fakt, że podejścia wykorzystujące mieszane cechy są rzadkością w literaturze tematu, a mogą one oferować zróżnicowany opis elementów kodu, **w pracy wykorzystano mieszane cechy do stworzenia hybrydowego systemu do wykrywania podatności oprogramowania.** Zastosowano w tym celu cechy tekstowe oraz metryki z analizatora kodu SonarQube (generuje on standardowe metryki oraz m.in. alerty ASA).

**Metody wykorzystywane do wykrywania podatności** Współczesne badania są zdominowane przez podejścia opierające się na uczeniu maszynowym i eksploracji danych. Metody te zyskały popularność również w dziedzinie przewidywania podatności oprogramowania [96]. Zasadniczo podejścia te można podzielić na dwie główne grupy: techniki kalkulacyjne i klasyfikacyjne [126]. Techniki kalkulacyjne mają na celu przewidywanie liczby podatności w jednostce systemu, a zadania klasyfikacyjne - występowanie samych podatności. W zadaniu klasyfikacji komponenty oprogramowania są oznaczane jako neutralne lub podatne na ataki [121]. Algorytmy oparte na klasyfikacji mogą bazować na różnych typach cech: metrykach oprogramowania (ang. software metrics, SM) [115, 126], tekście [16, 120, 122, 127], ostrzeżeniach ASA [117, 118] i mieszanych [124, 125]. Do tworzenia algorytmów przewidywania podatności wykorzystywano różne algorytmy: m.in. drzewa decyzyjne [122, 125], lasy losowe [122, 128], drzewa wzmacniane [125], liniową analizę dyskryminacyjną [83], sieci bayesowskie [83], naiwny klasyfikator Bayesa [121], algorytm k najbliższych sąsiadów [122], a także sztuczne sieci neuronowe i uczenie głębokie [16, 129, 130]. Na potrzeby pracy także zdecydowano się wykorzystać różne podejścia oparte na uczeniu maszynowym do wykrywania podatności oprogramowania. **Przetestowano bardziej konwencjonalne algorytmy uczenia maszynowego do wykrywania podatności oprogramowania** na podstawie różnorodnych metryk oprogramowania uzyskanych za pośrednictwem dwóch statycznych analizatorów kodu (Sekcja 3.2). We wcześniejszych pracach, nikt nie wykorzystał cech z tych analizatorów kodu do wykrywania podatności oprogramowania wraz z algorytmami uczenia maszynowego, więc praca stanowi cenny wkład w dziedzinę wykrywania podatności. W Sekcji 3.3 **zaproponowano również system wykorzystujący fuzję danych tekstowych i metryk statycznego analizatora kodu oraz mniej konwencjonalny model uczenia maszynowego - model hybrydowy zbudowany z wykorzystaniem Losowej Sieci Neuronowej oraz sieci konwolucyjnej.** To także stanowi nowość w literaturze powiązanej z wykrywaniem podatności, ponieważ nikt wcześniej nie stosował Losowych Sieci Neuronowych do tego zadania pomimo ich wysokiej skuteczności w wykonywaniu szerokiej gamy zadań [42, 44, 47, 48, 49].

**Wykorzystanie statycznych analizatorów kodu do wykrywania podatności.** Chociaż modele uczenia maszynowego zdobywają coraz większą popularność w obszarze wykrywania podatności, to tradycyjne narzędzia analizy statycznej oparte na regułach są nadal powszechnie używane. W artykule [112] podatności należące do grupy Buffer Overflow (CWE-119) zostały przeanalizowane przy użyciu ODC i dwóch narzędzi analizy statycznej (ang.

Static Analysis Tools, SAT). W [131], przeprowadzono testy porównawcze jedenastu analizatorów SAT dla języka C/C++ w oparciu o zestaw testów Toyoty [132]. Zestaw testowy Toyoty został również wykorzystany w pracy [133], w którym testowano różne narzędzia SAT w celu znalezienia luk związanych z pamięcią. Również prace [100, 101, 102, 103, 134] mają na celu porównanie różnych narzędzi SAT. Ich wyniki wykazały potencjał trzech narzędzi SAT w kontekście wykrywania luk w zabezpieczeniach: CppCheck, FlawFinder i Visual Code Grepper. Jako że dostępne prace skupiają się na testowaniu efektywności analizatorów kodu w znajdowaniu podatności w tradycyjnych produktach oprogramowania, w niniejszej pracy **przetestowano wymienione statyczne analizatory kodu z perspektywy skuteczności wykrywania znanych luk w produkcie bezpośrednio związanym z uczeniem głębokim** (Sekcja 3.1).

### 2.3.3 Zagrożenia dedykowane algorytmom sztucznej inteligencji

W niniejszej sekcji opisano zagrożenia bezpieczeństwa ściśle związane z algorytmami sztucznej inteligencji.

#### Krajobraz zagrożeń

Dedykowane zagrożenia w domenie sztucznej inteligencji są związane z charakterystyką algorytmów wchodzących w jej skład. Algorytmy te opierają swoje działanie na dużych ilościach danych w procesie automatycznego zdobywania wiedzy oraz do podejmowania decyzji. Ze względu na niską wyjaśnialność dużej części rozwiązań inteligentnych oraz młody wiek części najważniejszych technologii składowych, utrudnione jest dokładne zmapowanie krajobrazu tych zagrożeń [18]. Zagrożenia dedykowane algorytmom sztucznej inteligencji obejmują przede wszystkim dane wykorzystywane w różnych fazach życia systemu inteligentnego - od trenowania modeli aż do fazy operacyjnej. Przykładem zagrożeń z tej grupy jest zatrucie danych treningowych i walidacyjnych (ang. data poisoning), mające na celu spowodowanie konkretnego działania systemu w fazie operacyjnej [135]. Wzbogacanie danych zbioru treningowego o specyficzne próbki może na przykład skutkować wytworzeniem stronniczości danego modelu uczenia maszynowego w kierunku jakiejś konkretnej opcji [18]. Również manipulacja etykietami danych może spowodować niepożądane działanie z perspektywy właścicieli systemu w fazie operacyjnej. Zagrożenia związane z danymi dotykają także wstępnego przetwarzania danych wykorzystywanych do predykcji. Algorytmy uczenia maszynowego przystosowują się do formy danych obecnej w procesie treno-

wania. Zaburzenie procesu wstępnego przetwarzania danych najczęściej skutkuje błędnym działaniem systemu. W tym kontekście wstępne przetwarzanie danych obejmuje na przykład odpowiednie skalowanie i normalizację danych. Manipulacje mogą także obejmować parametry modeli uczenia maszynowego, czy też stosowane w nich procesy optymalizacyjne [18]. Zagrożenia mogą celować w dostępne modele uczenia maszynowego wykorzystywane podczas uczenia w trybie uczeń-nauczyciel (ang. student-teacher training). Także takie działania, zorientowane na algorytmy i strukturę modeli, mogą znacząco wpłynąć na działanie docelowego systemu inteligentnego. Jednymi z najbardziej znanych i największych zagrożeń dedykowanych algorytmom sztucznej inteligencji wchodzących w skład zatrucia danych są tzw. ataki adwersarza (ang. adversarial attacks) [22, 135].

### Ataki adwersarza

Wrażliwość sieci konwolucyjnych na specjalnie wygenerowane, nielosowe perturbacje została oryginalnie opisana w pracach [21] oraz [20]. Perturbacje te da się wygenerować w taki sposób, że są niewidoczne dla ludzkiego oka. To sprawia, że zagrożenia związane z atakami adwersarza są poważnym i ciągle aktualnym problemem domeny sztucznej inteligencji.

Istnieją dwa główne podziały ataków adwersarza. Jeden to podział na ataki typu białej oraz czarnej skrzynki, a drugi - na ataki ukierunkowane (ang. targeted) oraz nieukierunkowane (ang. non-targeted).

Pierwszy podział wiąże się z dostępnością do parametrów modelu. W przypadku ataków typu białej skrzynki, atakujący ma dostęp do parametrów modelu [20, 136, 137]. W tym przypadku perturbacje są generowane zazwyczaj za pośrednictwem optymalizacji danych wejściowych w taki sposób, aby zmaksymalizować błąd predykcji lub zmusić go do zwrócenia konkretne predykcji z dużym prawdopodobieństwem. Ataki typu czarnej skrzynki zakładają brak dostępu do parametrów modelu. Do generacji perturbacji są najczęściej wykorzystywane estymacje gradientów modeli atakowanych [138, 139, 140]. Chociaż w przypadku praktycznych ataków, ataki typu czarnej skrzynki są bardziej prawdopodobne, to ataki typu białej skrzynki dostarczają zazwyczaj bardziej szkodliwych perturbacji przez co mają nieocenione zastosowanie w procesie testowania sieci z perspektywy bezpieczeństwa. Ataki te są także wykorzystywane do generowania ataków typu czarnej skrzynki za pośrednictwem metod bazujących na transferze perturbacji [141].

Drugi podział wiąże się z celem ataku. Ataki nieukierunkowane mają na celu zmianę predykcji sieci, ale bez określenia docelowej predykcji. W przypadku ataków ukierunkowanych celem jest zwrócenie konkretnej predykcji. Ukierunkowane ataki mogą powodować poważniejsze konsekwencje niż ataki

nieukierunkowane poprzez oszukiwanie sieci w celu przewidywania wyznaczonych klas docelowych [142]. Ma to szczególne znaczenie w domenach, w których bezpieczeństwo jest priorytetem, takich jak inteligentny transport przemysł i zdrowie publiczne. Niemniej jednak sukces ataków nieukierunkowanych również wiąże się z pogorszeniem jakości usług oferowanych przez systemy informatyczne. Z tego powodu proponowanie nowych ataków nieukierunkowanych jest również istotne z perspektywy testowania głębokich sieci neuronowych.

Przykłady najczęściej wykorzystywanych w pracach naukowych ataków to Fast Gradient Sign Method (FGSM) [20], Projected Gradient Descent (PGD) [136] oraz Carlini & Wagner (C&W) [137]. Niektóre alternatywne ataki obejmują Universal Adversarial Perturbations [143], Generalizable Data-free UAP [144] i Top-k adversarial attack [145]. Niektóre ataki opierają się na pojedynczym kroku optymalizacji (np. FGSM [20]), a inne modyfikują metody jednokrokowe, np. Interacyjny FGSM (nazywany również Basic Iterative Method, BIM) [146]. Metody te można dalej modyfikować za pomocą różnych technik optymalizacji gradientowej, takich jak momentum [147].

Istniejące ataki wykorzystują najczęściej dane wyjściowe sieci do generowania perturbacji [148]. Jedynie nieliczne prace operują na wcześniejszych reprezentacjach danych. Przykładem takiej pracy jest [148], której celem jest kontaminacja wewnętrznych reprezentacji wszystkich warstw sieci głębokiej. Autorzy dokonują tego poprzez uśrednienie aktywacji w każdym konkretnym wymiarze przestrzennym wyjścia każdej warstwy konwolucyjnej. Używają łączonego celu straty biorąc pod uwagę wartości funkcji straty wyznaczone dla poszczególnych warstw. Feature Adversary Attack [149] ma na celu zminimalizowanie odległości głębokich reprezentacji dwóch obrazów wyrażonych przez konwolucyjne mapy cech (zamiast warstwy wyjściowej). TAP [141] działa na głębokich reprezentacjach obrazów. Jego celem jest zwiększenie możliwości przenoszenia ataków między różnymi sieciami - maksymalizuje on normę między oryginalnym a zmodyfikowanym obrazem dla wszystkich warstw.

Według raportu Agencji Unii Europejskiej ds. Cyberbezpieczeństwa (ENISA), ataki adwersarzy stały się już powszechne, a ich celem jest podważenie zaufania społeczeństwa i przemysłu do inteligentnych systemów informatycznych i produkcyjnych. Dlatego tak ważne jest, aby społeczność badawcza poświęciła znaczny wysiłek na rzecz zapewnienia postępów, które mogłyby zwiększyć odporność i niezawodność systemów sztucznej inteligencji zarówno w warunkach normalnych, jak i w przypadku ataku [22]. Z tego powodu, **w pracy przedstawiono nowy typ ataku adwersarza**, który może być wykorzystany do testowania głębokich sieci konwolucyjnych. W przeciwieństwie do większości obecnych w literaturze ataków, atak ten jest niezależny od klasyfikatora sieci, więc potencjalnie może być również stosowany w przypadku

innych typów sieci niż te do klasyfikacji obrazów. **Zaproponowano również nową metrykę opisującą stopnień szkodliwości ataku** wykorzystując pojęcie podobieństwa między znanymi obiektami. Proponowane metody cechują się nieskomplikowanym wykorzystaniem i prostotą interpretacji, co ma pozytywny wpływ na ich wyjaśnialność oraz umożliwia ich wykorzystanie do praktycznego testowania systemów inteligentnych.

## 2.4 Testowanie i ocena działania algorytmów uczenia maszynowego

Testowanie działania algorytmów uczenia maszynowego jest kluczowe z perspektywy oceny jakości ich działania oraz odporności na różnorodne czynniki, co ma kluczowy wpływ nie tylko na jakość oferowanych usług, ale także na bezpieczeństwo. Z jednej strony kluczowe jest testowanie wszystkich algorytmów uczenia maszynowego z perspektywy ich dokładności na danych zbliżonych do tych, które będzie można zaobserwować w fazie operacyjnej danego systemu. Z drugiej strony, szczególnie w przypadku algorytmów uczenia głębokiego, aby zapewnić odporność systemów inteligentnych na celowe modyfikacje danych (ataki), należy testować je również za pośrednictwem dedykowanych ataków i badać jak zachowują się w ich obecności.

### 2.4.1 Testowanie metod tradycyjnych

Pierwszym aspektem testowania algorytmów uczenia maszynowego jest ocena skuteczności ich działania w „normalnych" warunkach, czyli w obecności realistycznych danych, które nie były manipulowane w celu spowodowania błędnego działania systemu (np. za pośrednictwem ataków adwersarza). W celu dokładnego przetestowania każdego algorytmu uczenia maszynowego, potrzebne są duże ilości różnorodnych danych. Różnorodność danych jest istotna, ponieważ pozwala uwzględnić szeroką gamę sytuacji, które mogą wystąpić w fazie operacyjnej systemu inteligentnego. W przypadku nieuwzględnienia tej różnorodności w procesie testowania i ewaluacji systemów na prostych danych można uzyskać myląco wysokie wyniki dokładności, które nie reprezentują stanu faktycznego i zachowania modelu w rzeczywistych warunkach. Ogólnie, liczba danych koniecznych do testowania wzrasta wraz ze skomplikowaniem domeny problemowej, rozmiarem danych oraz skomplikowaniem modelu. Małe, interpretowalne modele wykorzystywane do rozwiązania prostych problemów opisanych niewielką liczbą cech uczących można z powodzeniem testować na mniejszych zbiorach danych. Takie zbiory można oznaczać manualnie stosunkowo niskim kosztem. W przypadku dużych,

trudnych w interpretacji modeli operujących na wielowymiarowych danych, kluczowe jest dostarczenie dużych ilości danych zebranych w różnorodnych warunkach. Manualne oznaczanie takich zbiorów jest niepraktyczne.

Z perspektywy danych testowych, istotne jest również wykorzystywanie danych do testowania, które są rozłączne ze zbiorem treningowym wykorzystanym do uczenia danego modelu. Istnieją różnorodne strategie podziału zbioru danych na próbki treningowe i testowe. Przykładem jest metoda wydzielania (ang. holdout cross-validation), w której cały zbiór jest dzielony na rozłączne zbiory: treningowy i testowy (zazwyczaj w proporcjach 70:30, 80:20, 50:50). Zbiór można podzielić również na trzy podzbiory: treningowy, testowy i walidacyjny. Różne modele są trenowane na zbiorze treningowym, ich umiejętność uogólniania jest w procesie trenowania testowana na zbiorze walidacyjnym, a na końcu model o największej skuteczności oraz mocy uogólniania jest testowany na zbiorze testowym. Ta strategia jest wykorzystywana w przypadku dostępu do dużej ilości danych. Gdy ilość danych jest ograniczona wykorzystuje się zazwyczaj k-krotny sprawdzian krzyżowy (ang. k-fold cross-validation). Jest to wszechstronna technika oceny skuteczności modeli, w której zbiór dzieli się na $k$ rozłącznych, równolicznych podzbiorów i w każdej iteracji model trenuje się na $k-1$ podzbiorach i testuje na pozostałym podzbiorze. Rozwiązanie to ze względu na konieczność wielokrotnego trenowania i testowania modeli jest bardzo czasochłonne i niepraktyczne w przypadku dużych zbiorów danych, przez co metody tej raczej nie wykorzystuje się w przypadku testowania modeli uczenia głębokiego na zbiorach danych dużej skali.

W celu numerycznego opisu skuteczności modeli wykorzystuje się pewne standardowe metody i miary oceny modeli. W przypadku klasyfikacji, która jest kluczowym zadaniem uczenia maszynowego w pracy wykorzystuje się takie miary jak: ogólną dokładność (ang. accuracy) oraz macierze pomyłek (ang. confusion matrices). Dokładność jest definiowana jako stosunek liczby poprawnie sklasyfikowanych przez model próbek do liczby wszystkich próbek. Metoda ta sprawdza się dla zbalansowanych zbiorów danych o małej liczbie klas. W przypadku danych niezbalansowanych może dawać mylące wyniki w przypadku stronniczości modelu w stronę klas większościowych oraz w przypadku dużej liczby klas i niskiej dokładności jedynie dla nielicznych klas (i wysokiej dla wszystkich innych). Macierze pomyłek w sposób pełny opisują dokładność każdej z klas oraz pokazują między jakimi klasami model się myli, przez co są cennym narzędziem inspekcji poprawności działania modelu. Macierze pomyłek mają postać macierzy, które na przecięciu $i$-tego wiersza i $j$-tej kolumny zawierają liczbę próbek sklasyfikowanych przez model jako klasa $j$-ta, a których prawdziwą klasą była klasa $i$-ta. Dlatego też macierz pomyłek dla idealnego systemu klasyfikacji powinna mieć postać diagonalną.

W przypadku znormalizowanych macierzy pomyłek, element pod indeksem $i, j$ zawiera dokładność dla klasy $i$-tej. Wartość ta nazywana jest wartością prawdziwie pozytywną dla danej klasy (ang. True Positive Rate, TPR). W przypadku klasyfikacji binarnej zazwyczaj jedną z klas oznacza się jako pozytywną (i przyjmuje ona wartość liczbową 1), a drugą jako negatywną. Macierz pomyłek ma wtedy rozmiar $2 \times 2$. TPR nazywana jest wtedy czułością (ang. recall), a dokładność rozpoznawania klasy negatywnej (zazwyczaj oznaczanej jako 0) - specyficznością (ang. specificity). W przypadku klasyfikacji binarnej można również użyć takich miar jak precyzja - stosunek wartości prawdziwie pozytywnych do sumy wartości prawdziwie i fałszywie pozytywnych. Wartości fałszywie pozytywne to takie, które model uznał za pozytywne, a w rzeczywistości to wartości negatywne. Inne miary wykorzystywane do oceny skuteczności modeli klasyfikacji to m.in.: miara F1, krzywe ROC (ang. Receiver Operating Curve) oraz pole pod tą krzywą. Miary oceny skuteczności klasyfikacji są wykorzystywane zarówno dla metod płytkiego uczenia, jak i dla algorytmów uczenia głębokiego. W przypadku zadań innych niż klasyfikacja istnieje szereg innych miar i metod wykorzystywanych do oceny skuteczności modeli. Na przykład w przypadku zadań regresji, wykorzystuje się różnego rodzaju błędy (np. błąd średniokwadratowy, czy też średni błąd bezwzględny).

### 2.4.2 Testowanie metod uczenia głębokiego

Testowanie metod uczenia głębokiego jest zazwyczaj bardziej skomplikowane niż testowanie metod płytkiego uczenia maszynowego. Jeszcze większe znaczenie ma w tym przypadku dostęp do dużej ilości danych testowych. Jako że algorytmy uczenia głębokiego ze względu na interpretowalność to algorytmy typu czarnej skrzynki, to trudno jest przewidzieć jakie rezultaty uzyska model na różnorodnych danych. Jest to szczególnie istotne w przypadku algorytmów uczenia głębokiego wykorzystywanych do rozwiązania problemów wizyjnych. Tego typu algorytmy są stosowane w wielu domenach, w których poprawne działanie i bezpieczeństwo algorytmów jest krytyczne (np. inteligentne fabryki, autonomiczne pojazdy itp.). Głębokie sieci wizyjne są bardzo wrażliwe na zmiany w rozkładzie danych i zmiana tego rozkładu (np. spowodowana innym rozmiarem danych treningowych i testowych, innymi warunkami oświetlenia panującymi podczas robienia zdjęć itp.) może spowodować znaczny spadek dokładności algorytmów percepcji [150], a więc spadek jakości działania systemu bazującego na danym algorytmie. Tworzenie dedykowanych danemu zastosowaniu zbiorów danych jest bardzo wymagające i wiąże się zazwyczaj z dużym nakładem czasowym i pieniężnym związanym z manualnym oznaczaniem danych przez ludzi. Z tego powodu konieczne jest

dostarczenie rozwiązań, które umożliwiłyby szybkie zbieranie różnorodnych danych w rzeczywistych warunkach i ich anotację w celu wydajnego i dokładnego testowania algorytmów uczenia głębokiego.

Z drugiej strony, w domenie uczenia głębokiego poważnym zagrożeniem są wcześniej opisane ataki adwersarza. Jako że testowanie dokładności na czystych danych nie jest już wystarczającym aspektem testowania systemów inteligentnych, w praktycznym testowaniu takich algorytmów należy również zbadać wpływ różnorodnych ataków adwersarza na tworzony model. Ma to zapewnić wiarygodne i bezpieczne działanie modelu nie tylko w „normalnych" warunkach, ale także w przypadku intencjonalnej manipulacji obrazów za pośrednictwem ataków adwersarza. Pomimo że w przypadku praktycznych ataków, ataki typu czarnej skrzynki są bardziej prawdopodobne, to ataki typu białej skrzynki dostarczają zazwyczaj bardziej szkodliwych perturbacji. Z tego powodu w procesie testowania powinno się zwrócić uwagę właśnie na ten typ ataków. Ataki te są także wykorzystywane do generowania ataków typu czarnej skrzynki za pośrednictwem metod bazujących na transferze perturbacji [141], co tym bardziej zwiększa ich wartość w procesie testowania. Z tego powodu, w niniejszej pracy **zaproponowano nowy typ ataku typu białej skrzynki**, który ze względu na prostotę generacji może być wykorzystany do praktycznego testowania systemów opartych na wizyjnych algorytmach uczenia głębokiego.

### 2.4.3 Automatyczne oznaczanie danych na potrzeby testowania algorytmów uczenia głębokiego w wizji komputerowej

Tak jak wspomniano wcześniej, do trenowania i testowania systemów wizyjnych konieczne jest dostarczenie olbrzymich ilości danych. W celu stworzenia oznaczonych zbiorów w tej dziedzinie w sposób automatyczny można wykorzystać kilka podejść. W przypadku nieustrukturyzowanych środowisk zewnętrznych zwykle stosuje się symulacje (np. automatyczne etykietowanie symulowanych scen środowisk planetarnych w pracy [151]). Niemniej jednak symulacje zapewniają jedynie uproszczony świat i jego fizykę. Z tego powodu, jeśli to możliwe, systemy oparte na uczeniu głębokim powinny być trenowane na rzeczywistych danych, a platformy robotyczne wykorzystujące te systemy powinny być testowane przed wdrożeniem w rzeczywistych środowiskach.

Duży potencjał do oznaczania obiektów w świecie rzeczywistym za pośrednictwem obiektów umożliwiających ich lokalizację i identyfikację wykazują znaczniki (ang. Fiducial markers). Znaczniki mogą być używane do szacowania pozycji kamery z dużą szybkością, przy niskich kosztach i wysokim

współczynniku wykrywalności znaczników [152]. Najpopularniejszym typem markera jest ten o kwadratowym kształcie, ponieważ można go w prosty sposób wykorzystać do wyodrębnienia pozycji kamery na podstawie czterech narożników [153]. Zazwyczaj znaczniki zawierają unikalne identyfikatory, np. kod binarny [153]. Dostępnych jest wiele systemów znaczników, które mogą być wykorzystywane przez społeczność związaną z robotyką. Znaczniki można zazwyczaj wydrukować za pomocą standardowej drukarki domowej. Przykładami systemów znaczników są ArUco [153], AprilTag [154] i ARTag [155]. Prace wykorzystujące znaczniki do automatycznego oznaczania obiektów na zdjęciach to [156, 157]. Prace te zakładają anotację danych offline. W pracy [156] autorzy stworzyli system do etykietowania małych obiektów na taśmociągach dla ramion robotów wykorzystywanych w zautomatyzowanej produkcji do wykrywania obiektów. Autorzy wykorzystali specjalnie wykonane cokoły trzymające każdy obiekt wraz ze znacznikiem, aby zapewnić pola ograniczające małe obiekty. Autorzy rozszerzyli swoje podejście w pracy [157], w której otoczyli małe obiekty okręgiem znaczników i oszacowali ich położenie (zakładają regularny kształt obiektów, ponieważ ich celem jest głównie produkcja produktów spożywczych). W przeciwieństwie do prac [156, 157], w których dane są najpierw gromadzone, a następnie anotowane offline na komputerze, w niniejszej **pracy zaproponowano inne podejście**. Pozwala ono na gromadzenie, anotację danych i testowanie sieci neuronowych w czasie rzeczywistym bezpośrednio na pojeździe. Nie wymaga ono żadnych dodatkowych stojaków/podestów, a jedynie pojedyncze znaczniki przyklejone bezpośrednio do obiektu. Zwiększa to praktyczność metody w stosunku do dostępnych rozwiązań. Podejście to wydaje się szczególnie przydatne w zastosowaniach, w których nie jest możliwe zbudowanie dodatkowych cokołów lub otoczenie całego obiektu znacznikami (np. w przypadku dużych obiektów przemysłowych, takich jak wózki widłowe). Chociaż głównym celem opracowanego podejścia jest testowanie wytrenowanych sieci głębokich, to uzyskane zbiory danych mogą być również wykorzystywane do ich trenowania.

## 2.5 Relacja wyjaśnialności i bezpieczeństwa systemów inteligentnych

W niniejszej pracy doktorskiej zbadano dwa istotne aspekty dla dziedziny sztucznej inteligencji - wyjaśnialność i bezpieczeństwo. Wybrano te tematy nie tylko ze względu na ich istotność, ale również ze względu na ich wzajemny wpływ. Z jednej strony niska wyjaśnialność algorytmów sztucznej inteligencji (w szczególności głębokich sieci neuronowych) utrudnia testowanie modeli

oraz przewidywanie możliwych rezultatów. Gdy system inteligentny zawodzi, trudno jest ustalić przyczynę. Jest to szczególnie ważne w obszarach, w których bezpieczeństwo jest krytyczne. Przez niską wyjaśnialność i jej wpływ na trudną inspekcję modeli pod kątem bezpieczeństwa, znacząco cierpi zaufanie społeczeństwa do sztucznej inteligencji. Niska praktyczna wyjaśnialność metod wykorzystywanych do opisu szkodliwości ataków adwersarza, które są kluczowe dla dziedziny sztucznej inteligencji utrudnia inspekcję algorytmów pod kątem odporności na ataki tego typu. Z drugiej strony dedykowane zagrożenia sztucznej inteligencji - ataki adwersarza - również wpływają na działanie samych metod interpretowalności. W przypadku analizy skupienia sieci w celu wytłumaczenia jej decyzji, często wykorzystuje się metody wizualne oparte na gradientach sieci. Jako że perturbacje wprowadzone przez ataki nie są widoczne gołym okiem przez człowieka na obrazie, mapy aktywacji (mające na celu wskazanie obszarów obrazu odpowiedzialnych za konkretne predykcje) w przypadku niektórych ataków mogą być mylące i mało informatywne. Z drugiej strony mapy te mogą być wykorzystane do wykrycia (na podstawie wizualizacji) niektórych typów ataków. Z powodu wzajemnej relacji wyjaśnialności i bezpieczeństwa, konieczne jest skupienie się na obu tych aspektach i zaproponowanie rozwiązań mających na celu ich poprawę i poszerzenie stanu wiedzy w ich zakresie. Również istotne jest aby nowe metody z zakresu bezpieczeństwa cechowały się prostotą interpretacji i działania, co zwiększy ich transparentność oraz wyjaśnialność, a zatem szansę na ich efektywne zastosowanie w praktyce.

# Rozdział 3

# Bezpieczeństwo

W niniejszym rozdziale opisano prace oraz autorskie rozwiązania mające na celu poprawę bezpieczeństwa systemów inteligentnych. Są to rozwiązania z zakresu tradycyjnych oraz dedykowanych zagrożeń sztucznej inteligencji. Chociaż głównym tematem rozdziału jest bezpieczeństwo, to niektóre z proponowanych rozwiązań mają również na celu poprawę wyjaśnialności systemów inteligentnych.

## 3.1 Analiza krajobrazu luk bezpieczeństwa bibliotek uczenia głębokiego i numerycznych na przykładzie TensorFlow

Krajobraz zagrożeń AI jest bardzo trudny do zmapowania, ponieważ obejmuje wiele zaawansowanych metod i złożonych procesów. Co więcej, produkty tego typu często korzystają z platform zewnętrznych dostawców. TensorFlow jest jedną z najpopularniejszych platform uczenia maszynowego [158], często postrzeganą jako najpopularniejsza [159]. Korzystają z niej tacy giganci technologiczni jak PayPal czy eBay [160]. Podobnie jak w przypadku każdego innego oprogramowania, jest ono narażone na awarie bezpieczeństwa, często wynikające z luk w oprogramowaniu. W pewnych warunkach błędne wykonanie kodu może mieć wpływ na poufność, dostępność lub integralność (triada CIA) systemu [161].

Pomimo faktu, że wiele pracy włożono w poprawę bezpieczeństwa, luki w zabezpieczeniach są nadal niezwykle powszechne. Najczęściej występującymi i najgroźniejszymi lukami są te związane z językami C/C++ (według raportu Veracode [12] i rankingu CWE TOP25 [13]). Oferują one wysoką kontrolę zasobów i wydajność, co wiąże się z kosztem - zagrożeniami o wysokim stopniu

szkodliwości dla systemu. Ze względu na swoją wydajność, to właśnie języki C i C++ są najczęściej wykorzystywane do implementacji wysokopoziomowych bibliotek numerycznych (np. TensorFlow czy Caffe). Chociaż istnieje wiele innych (dedykowanych) zagrożeń w aplikacjach AI, ważne jest, aby skupić się na tych znanych podatnościach, ponieważ często zapomina się o nich w walce z zagrożeniami bezpieczeństwa związanymi ze sztuczną inteligencją. Z tego powodu w niniejszej pracy postanowiono przeprowadzić badania analityczne dotyczące luk w zabezpieczeniach TensorFlow - przykładowej, wiodącej biblioteki uczenia głębokiego. Wynikiem jest zestaw obserwacji dotyczących najczęściej występujących w oprogramowaniu luk, ich przyczynach oraz potencjalnych skutkach.

Na potrzeby pracy opisano luki w zabezpieczeniach na poziomie typów CWE wraz z przykładami z implementacji TensorFlow (instancje podatności według standardu CVE). Opracowano kompleksową bazę wiedzy na temat głównych przyczyn różnych typów CWE, ich skutków i sposobów wykorzystania. Zwrócono również uwagę na sposoby uniknięcia/zminimalizowania szansy ich wykorzystania. Przeprowadzona analiza obejmuje większą liczbę typów CWE niż inne prace w literaturze, które zazwyczaj analizują jeden lub dwa typy CWE jednocześnie. Co więcej, według najlepszej wiedzy autorki rozprawy wcześniejsze prace opisane w literaturze nie analizowały bibliotek uczenia głębokiego samych w sobie, a jedynie ich zależności. W pracy przetestowano również zdolność przykładowych statycznych analizatorów kodu do wykrywania analizowanych luk w TensorFlow, co także wcześniej nie zostało zbadane.

### 3.1.1 Proponowane rozwiązanie i metodologia badań

Praca miała na celu scharakteryzowanie sześciu różnych typów CWE podatności oprogramowania w implementacji TensorFlow. Przeanalizowano 104 instancje CVE. Obejmowały one podatności związane z zarządzaniem pamięcią, przekroczeniem zakresu liczb całkowitych, nieprawidłową walidacją danych wejściowych i dereferencją wskaźnika NULL (ang. NULL pointer dereference). Aby to osiągnąć, **podjęto cztery uzupełniające się kroki**:

1. Analiza metadanych dostępnych na CVEDetails.com: statystyki dotyczące uzyskanych przez luki wartości Common Vulnerability Scoring System (CVSS), wpływ na triadę CIA i możliwe potencjalne rezultaty ich wykorzystania;
2. Analiza podatności i ich poprawek w TensorFlow na podstawie commitów dotyczących tych poprawek za pośrednictwem Orthogonal Defect Classification (ODC);

3. Analiza alertów SAT uzyskanych za pomocą CppCheck, FlawFinder i Visual Code Grepper dla podatnych i neutralnych plików (przed i po naprawie);
4. Analiza opisów podatności i ich poprawek dostępnych na stronie `CVED etails.com` i „TensorFlow github Advisory" oraz zmian w kodzie.

**W celu realizacji zadania z punktu 1.**, korzystając z danych pozyskanych z bazy CVE Details, zbadano metadane dotyczące badanych instancji CVE: wartości Common Vulnerability Scoring System (CVSS) oraz wpływ na poufność, integralność i dostępność (triada CIA) systemu, w przypadku potencjalnego wykorzystania podatności. Zgromadzono wszystkie dostępne dane. Następnie obliczone zostały podstawowe statystyki wartości wyników CVSS dla różnych typów badanych podatności: średnia, odchylenie standardowe, minimum i maksimum. Ponadto zliczono instancje CVE z każdej grupy CWE należące do różnych kategorii triady CIA.

**W celu realizacji zadania z punktu 2.**, korzystając z atrybutów typ defektu i kwalifikator metody ODC, skojarzono każdą podatność z określonymi działaniami na różnych etapach tworzenia oprogramowania i błędami w tym procesie [162]. Atrybuty te dostarczyły informacji przyczynowych dotyczących każdej analizowanej podatności. Wynikowe statystyki wartości tych atrybutów w systematyczny i łatwy do interpretacji sposób opisują naturę różnych typów podatności. W analizie wybrano odpowiednie wartości atrybutów na podstawie zmian zachodzących podczas usuwania danej podatności. ODC jest klasyfikacją manualną, więc podobnie jak w przypadku pracy [112], dwóch badaczy przeprowadziło analizę dla większej niezależności i niezawodności. Najpierw przeprowadzono dyskusję i ustalono kryteria klasyfikacji. Pierwszych kilka przykładów zostało przeanalizowanych wspólnie. Następnie wszystkie pozostałe instancje CVE zostały przeanalizowane osobno, a na koniec omówiono niepokrywające się rezultaty i znaleziono wspólny konsensus. W ten sposób uzyskano podstawę dla ostatecznych wyników.

**W celu realizacji zadania z punktu 3.**, kody źródłowe związane z badanymi podatnościami przebadano za pośrednictwem statycznych analizatorów kodu. Uzyskane za ich pośrednictwem alerty przeanalizowano pod kątem występowania ich prawdziwego identyfikatora CWE oraz określono precyzję znajdowania linii dotyczących badanych instancji CVE. W tym celu wykorzystano informacje dotyczące linii, dla których dany SAT znalazł błędy, znaleziono je w plikach dotyczących konkretnych podatności i porównano z prawdziwymi liniami, które były odpowiedzialne za daną podatność. Przeanalizowano również, czy dane alerty powinny być brane pod uwagę (wybrano przykładowe alerty, które można wykorzystać do poprawy jakości kodu TensorFlow i opisano potencjalne problemy i ulepszenia). Porównano

również wyniki uzyskane dla odpowiednich podatnych i neutralnych wersji plików.

**W celu realizacji zadania z punktu 4.**, skoncentrowano się na opisach tekstowych podatności dotyczących wszystkich badanych instancji CVE dostępnych na stronie CVE Details i odpowiadających im adnotacjach w ramach repozytorium GitHub TensorFlow, a także implementacjach kodu w celu znalezienia wspólnych przyczyn i rezultatów występowania i wykorzystywania podatności. Zdefiniowano również możliwe strategie wykorzystania podatności przez użytkowników aplikacji wykorzystujących TensorFlow.

Podstawowym źródłem podatności i informacji o nich wykorzystanym w pracy była strona **CVE Details** i znajdująca się na niej wyszukiwarka podatności. W wyszukiwarce można znaleźć podatności należące do konkretnego typu CWE. Aby wybrać określone typy CWE, wzięto pod uwagę liczbę dostępnych instancji CVE w TensorFlow (CVE Details) dla każdego typu CWE i jego pozycję w rankingu CWE TOP25 2021 [13]. W ten sposób uwzględniono trzy różne aspekty poszczególnych typów podatności: ich rozpowszechnienie i dotkliwość w różnych typach oprogramowania, a także ich częstość występowania w Tensorflow. Pozwala to traktować TensorFlow indywidualnie, a także jako przykład nowoczesnej aplikacji C/C++.

Przy opisie zagrożenia związanego z podatnościami badanymi w pracy oraz ich potencjalnych skutków wykorzystano **Common Vulnerability Scoring System (CVSS)** oraz potencjalny wpływ tych podatności na **triadę CIA** opisany na stronie **CVE Details**. CVSS ocenia wpływ luki na poufność, integralność i dostępność systemu, gdy zostanie ona wykorzystana. Wynik przyjmuje wartości z zakresu od 0 do 10. CVSS jest uważany za standardowy system pomiarowy dla organizacji, instytucji rządowych i przemysłu, ponieważ zapewnia spójne i precyzyjne wyniki oceny podatności (zgodnie z National Institute of Standards and Technology - NIST [91]). Z drugiej strony **triada CIA** odnosi się do podstawowych elementów systemów informatycznych: Poufności, Integralności i Dostępności (ang. Confidentiality, Integrity and Availability) [163]. Trzy typy są używane do opisania wpływu na każdą część triady CIA: brak wpływu, wpływ częściowy i całkowity [164].

Do systematycznego opisu luk w oprogramowaniu TensorFlow wykorzystano Ortogonalną Klasyfikację Defektów (ang. **Orthogonal Defect Classification, ODC**) [162]. ODC to opracowane przez IBM systematyczne podejście do klasyfikacji i analizy defektów oprogramowania poprzez przekształcanie informacji semantycznych w miary [112]. Podejście to jest często stosowane w analizie przyczyn źródłowych błędów oprogramowania (ang. root-cause analysis) [112]. Pełna analiza polega na opisaniu defektu przed i po naprawie za pomocą serii 8 atrybutów (z predefiniowanymi wartościami). Można użyć następujących atrybutów: aktywność (ang. activity), wyzwalacz (ang.

trigger), wpływ (ang. impact), cel (ang. target), typ defektu (ang. defect type), kwalifikator (ang. qualifier), źródło (ang. source) i wiek (ang. age). Aby zbadać podatności na potrzeby pracy, użyto dwóch atrybutów: **typ defektu** i **kwalifikator** (pozostałe nie zawierają żadnych dodatkowych istotnych informacji dotyczących charakterystyki danej podatności [112]).

Atrybut **defect type** jest używany do reprezentowania semantyki poprawki (oddaje znaczenie wprowadzonej poprawki, a więc naturę konkretnego problemu) [162]. Atrybut ten może przyjąć jedną z ośmiu wartości. Z analizy wykluczono dwie z nich (build/package/merge i documentation), ponieważ w analizie uwzględniono wyłącznie pliki z kodem źródłowym. Opierając się na definicjach z [162] i [165], można zdefiniować typy defektów w następujący sposób:

1. **Assignment/Initialization** - problemy związane z brakiem/poprawnym przypisaniem wartości zmiennej
2. **Checking** - problemy związane z logiką warunkową (nieprawidłowe lub brakujące warunki). W przypadku konieczności dodania/poprawienia wielu instrukcji sprawdzających może należeć do typu Algorithm/Method.
3. **Timing** - problemy związane z serializacją współdzielonych zasobów.
4. **Algorithm/Method** - problemy związane z implementacją (zasadniczo poprawki konstrukcyjne nie powinny być konieczne)
5. **Function** - problemy związane z brakiem lub nieprawidłową implementacją niektórych istotnych funkcji, interfejsów użytkownika końcowego lub globalnych struktur danych (zazwyczaj wymaga to zmiany konstrukcyjnej)
6. **Interface** - problemy związane z komunikacją między podsystemami.

Powyższe typy zostały opisane bardziej szczegółowo w pracy [165].

Atrybut **qualifier** służy do uchwycenia tego, co konieczne było do zrobienia w celu naprawy określonej luki w zakresie zmian w kodzie i przyjmuje następujące wartości:

1. **Missing** - aby naprawić błąd, należało dodać nowy kod,
2. **Incorrect** - aby naprawić błąd, konieczne było dostosowanie kodu,
3. **Extraneous** - aby naprawić błąd, część kodu musiała zostać usunięta.

Uwzględnienie atrybutu **qualifier** rozszerza informacje semantyczne dostarczane przez atrybut **defect type**. Użycie tych dwóch atrybutów pozwala

powiązać podatność z określonymi działaniami na różnych etapach tworzenia oprogramowania i potencjalnymi błędami w tym procesie [162].

W pracy wykorzystano również trzy darmowe **statyczne analizatory kodu** do sprawdzenia ich umiejętności wykrywania znanych podatności w TensorFlow:

- **CppCheck** [166] (wersja 2.6),
- **FlawFinder** [167] (wersja 2.0.19),
- **Visual Code Grepper** [168] (wersja 2.2.0.0).

Zdecydowano się wykorzystać te konkretne analizatory kodu, ponieważ wykazały one lepszą skuteczność niż inne analizatory testowane w pracach [100, 101, 102, 103, 134]. W pracy [102] przetestowano 9 systemów SATa i stwierdzono, że Cppcheck generuje bardzo niewiele fałszywie pozytywnych ostrzeżeń. Również w pracy [101] wykazano, że CppCheck był w stanie znaleźć maksymalną liczbę podatności spośród testowanych narzędzi. W pracy [134] uzyskano wyniki pokazujące, że chociaż Visual Code Grepper SAT daje stosunkowo dużą liczbę fałszywie pozytywnych ostrzeżeń, to był w stanie wykryć wszystkie badane podatności. W pracy [103], Flawfinder znacząco przewyższył inny testowany tradycyjny SAT pod względem znajdowania błędów CWE-119. Również w pracy [100], Flawfinder był w stanie wygenerować najwyższe wartości czułości spośród wszystkich testowanych SAT.

W analizie uwzględniono 6 typów CWE oraz 104 instancje CVE. W Tabeli 3.1, przedstawiono wszystkie zbadane typy CWE wraz z liczbą ich instancji w TensorFlow i ich pozycją w rankingu CWE TOP25. Zbadane podatności wydają się być adekwatne z punktu widzenia bezpieczeństwa biblioteki numerycznej. Dominujące typy to te związane z problemami z pamięcią. Biblioteki numeryczne często opierają się na skomplikowanych operacjach matematycznych wykorzystujących tablice. Chociaż języki takie jak C lub C++ oferują szybkość obliczeń, nie zapewniają wbudowanej ochrony przed dostępem do pamięci / nadpisywaniem lub sprawdzaniem granic bufora, które stosunkowo łatwo mogą być przekroczone w przypadku niedotrzymania pewnych warunków operacji. Analiza objęła bardziej ogólny typ CWE związany z pamięcią - CWE-119, ale także jego bardziej szczegółowe typy potomne, typy CWE-125 oraz CWE-787. Ponadto przeanalizowano podatności związane z operacjami numerycznymi i dynamicznymi strukturami danych: NULL Pointer Dereference (CWE-476) oraz Integer Overflow or Wraparound (CWE-190). Przeanalizowane zostały również problemy związane z nieprawidłową walidacją danych wejściowych (CWE-20). Ten typ podatności jest przykładem niezależnej od języka programowania grupy CWE, która ma wpływ na praktycznie wszystkie rodzaje aplikacji.

**Tabela 3.1.** Wykorzystane w pracy typy CWE posortowane względem malejącej liczby instancji. Rank oznacza pozycję w rankingu CWE TOP25.

| Instancje | Rank | CWE | Nazwa | Opis |
|---|---|---|---|---|
| 34 | #3 | CWE-125 | Out-of-bounds Read | Dane mogą zostać odczytane zarówno za końcem, jak i przed początkiem przydzielonego bufora. |
| 25 | #15 | CWE-476 | NULL Pointer Dereference | Występuje, gdy dany program dereferencjonuje wskaźnik, który powinien być prawidłowy, ale w rzeczywistości jest typu NULL. |
| 19 | #1 | CWE-787 | Out-of-bounds Write | W rezultacie dane mogą zostać zapisane za końcem lub przed początkiem przydzielonego bufora. |
| 11 | #4 | CWE-20 | Improper Input Validation | Jest to spowodowane brakiem lub nieprawidłową walidacją danych wejściowych. |
| 8 | #17 | CWE-119 | Improper Restriction of Operations within the Bounds of a Memory Buffer | Operacje wykonywane na buforze pamięci powodują odczyt lub zapis do lokalizacji pamięci, która znajduje się poza zamierzonymi granicami bufora (rodzic CWE-787 i CWE-125). |
| 7 | #12 | CWE-190 | Integer Overflow or Wraparound | Obliczenie wartości może spowodować przepełnienie liczby całkowitej, podczas gdy wynik powinien być zawsze większy niż oryginalna wartość. |
| W sumie: 104 | | | | |

W celu zebrania danych dotyczących wszystkich badanych instancji CVE wykorzystano stronę CVE Details. Dane te obejmowały metadane każdej instancji CVE, linki do commitów GitHub z kodem źródłowym oraz zgłoszeniami dotyczącymi danych podatności. Na stronie każdej instancji CVE na CVE Details można znaleźć informacje o wartości CVSS, wpływie na triadę CIA oraz potencjalnych wynikach wykorzystania podatności. Do opisu wpływu na triadę CIA stosowane są trzy poziomy: Brak (ang. None), Częściowy (ang. Partial) i Całkowity (ang. Complete). Na potrzeby niniejszej pracy zebrano niezbędne dane i określono statystyki tych wartości opisujące różne typy CWE. W celu manualnej analizy kodów, wykorzystano commity GitHub naprawiające dane instancje CVE. Wykorzystano przeglądarkę GitHub do analizy z wykorzystaniem ODC oraz pobrano podatne i neutralne pliki do analizy za pomocą programów SAT (jako podatne pliki potraktowano wersje przed poprawką, podczas gdy neutralne pliki to te zawierające poprawkę).

### 3.1.2 Wyniki

#### Analiza metadanych dotyczących podatności

Dla wszystkich badanych typów CWE obliczono średnią, odchylenie standardowe, minimalne i maksymalne wartości wyników CVSS. Ponadto zliczono instancje CVE z każdej grupy CWE należące do różnych kategorii wpływu triady CIA. Wartości te wykorzystano do obliczenia procentowego udziału tych kategorii wpływu dla każdego typu CWE. Dane wejściowe pochodziły z bazy danych CVE Details. Wyniki przedstawiono na Rysunku 3.1 oraz w Tabelach 3.2 i 3.3. Najwyższe wartości CVSS uzyskano dla błędów związanych z pamięcią (CWE-787, CWE-119 oraz CWE-125). Typy te przeważnie wpływają na wszystkie elementy triady CIA. Podatności te są również najbardziej zróżnicowane pod względem możliwych wyników wykorzystania podatności (przepełnienia, obejścia, wykonania kodu, uszkodzenia pamięci i DoS, co można zobaczyć w Tabeli 3.2). Grupa CWE-190 (Integer Overflow or Wraparound) uzyskała średnio najniższe wartości CVSS (podatności te zazwyczaj wpływają tylko na dostępność - patrz Tabela 3.3). Luki CWE-190 często skutkują przepełnieniami i atakami DoS. Badane przykłady CWE-20 (Improper Input Validation) uzyskały wyższe średnie i maksymalne wartości niż podatności CWE-190. Uzyskały one również najwyższe odchylenie standardowe - co sugeruje najbardziej zróżnicowaną punktację CVSS. Podatności te mają wpływ głównie na dostępność systemów. Przyczyną awarii dostępności może być odmowa usługi (patrz Tabela 3.2). Wyniki uzyskane dla podatności CWE-476 są podobne do tych uzyskanych dla CWE-20 pod

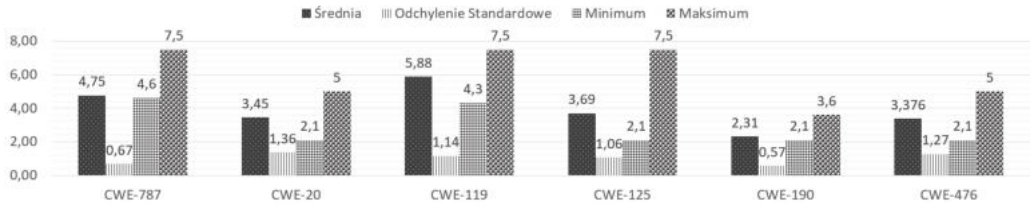
**Tabela 3.2.** Rozkład procentowy możliwych rezultatów wykorzystania różnych typów podatności określonych za pośrednictwem standardu CWE.

| | Bypass | DoS | Code Exec. | Mem. Corr. | Overflow |
|---|---|---|---|---|---|
| CWE-787 | 21.05% | 0.00% | 5.26% | 5.26% | 68.42% |
| CWE-20 | 0.00% | 72.73% | 0.00% | 0.00% | 0.00% |
| CWE-119 | 0.00% | 12.50% | 12.50% | 0.00% | 100.00% |
| CWE-125 | 2.94% | 2.94% | 0.00% | 0.00% | 8.82% |
| CWE-190 | 0.00% | 42.86% | 0.00% | 0.00% | 100.00% |
| CWE-476 | 4.00% | 12.00% | 8.00% | 0.00% | 0.00% |

**Tabela 3.3.** Rozkład procentowy wpływu na pojedyncze składniki triady CIA dla różnych typów podatności określonych za pośrednictwem standardu CWE.

| | Poufność (**C**onfidentiality) | | | Integralność (**I**ntegrity) | | | Dostępność (**A**vailability) | | |
|---|---|---|---|---|---|---|---|---|---|
| | Brak | Częściowy | Całkowity | Brak | Częściowy | Całkowity | Brak | Częściowy | Całkowity |
| CWE-787 | 0.00% | 100.00% | 0.00% | 0.00% | 100.00% | 0.00% | 0.00% | 100.00% | 0.00% |
| CWE-20 | 90.91% | 9.09% | 0.00% | 90.91% | 9.09% | 0.00% | 0.00% | 100.00% | 0.00% |
| CWE-119 | 25.00% | 75.00% | 0.00% | 25.00% | 75.00% | 0.00% | 25.00% | 75.00% | 0.00% |
| CWE-125 | 11.76% | 88.24% | 0.00% | 82.35% | 17.65% | 0.00% | 14.71% | 85.29% | 0.00% |
| CWE-190 | 100.00% | 0.00% | 0.00% | 85.71% | 14.29% | 0.00% | 0.00% | 100.00% | 0.00% |
| CWE-476 | 56.00% | 44.00% | 0.00% | 52.00% | 48.00% | 0.00% | 0.00% | 100.00% | 0.00% |

względem wyników CVSS (podobna średnia, odchylenie standardowe oraz te same wartości minimalne i maksymalne). Również te rodzaje podatności wpływają głównie na dostępność. Prawie 50 procent z nich może wpływać na poufność lub integralność (w przeciwieństwie do podatności CWE-20). Niektóre z luk mogą również skutkować obejściem zabezpieczeń, odmową usługi lub wykonaniem kodu.




**Rysunek 3.1.** Średnia, odchylenie standardowe oraz wartości minimalne i maksymalne ocen CVSS dla różnych typów podatności określonych za pośrednictwem standardu CWE.

### Orthogonal Defect Classification

Tabela 3.4 prezentuje szczegółowe wyniki ODC dla wszystkich typów CWE. W analizie ODC w żadnym przypadku atrybut „Qualifier" nie przyjął war-

tości „Extraneous", a atrybut „Defect Type" - „Timing" i „Assignment". Dlatego nie uwzględniono ich w wynikach przedstawionych w Tabeli 3.4. W przypadku podatności związanych z pamięcią (CWE-787, CWE-125 i CWE-119) najczęstszym problemem były brakujące instrukcje sprawdzające. W przypadku CWE-119 takie same wyniki uzyskano dla „Checking" i „Algorithm/Method". Wskazuje to na potrzebę wprowadzenia bardziej rozbudowanych poprawek. Ponieważ ten typ CWE jest typem nadrzędnym dla innych typów CWE związanych z pamięcią (CWE-787, CWE-125), może on zawierać bardziej zróżnicowane przypadki niż typy potomne. Częstość występowania wartości „Missing" dla atrybutu „Qualifier" oznacza, że prawdopodobnie w fazie implementacji niektórych funkcjonalności zapomniano/nie wzięto pod uwagę potrzeby prostszej lub bardziej złożonej walidacji niektórych parametrów. Ten sam problem można zaobserwować dla CWE-20. W przypadku CWE-125 zaobserwowano również wartości „Function" i „Interface" atrybutu „Defect Type". Ich występowanie sugeruje bardziej złożoną naturę defektów powodujących podatności, ponieważ wymagają one zmian projektowych i komunikacyjnych. W przypadku CWE-476, ODC wykazało, że problemem były głównie brakujące mechanizmy kontrolne (w wielu przypadkach do naprawienia podatności wymagane jest sprawdzenie, czy wskaźnik, do którego ma zostać uzyskany dostęp, nie jest wskaźnikiem NULL). Jedynym typem CWE, w którym zaobserwowano więcej defektów wymagających modyfikacji kodu niż jego dodania, był CWE-190 (Integer Overflow lub Wraparound). Większość defektów związanych z tym typem wymagało przebudowania większej części danego fragmentu kodu („Defect Type" przybierał wartość „Algorithm").

### Analiza skuteczności statycznych analizatorów kodu

Do analizy wykorzystano programy CppCheck, FlawFinder i Visual Code Grepper - trzy statyczne analizatory kodu o otwartym kodzie źródłowym - dla podatnych i neutralnych wersji plików kodu źródłowego. W Tabeli 3.5 podsumowano liczbę błędów zgłoszonych dla różnych typów CWE i analizatorów. Wyróżniono zmiany w liczbie znalezionych błędów w wersjach kodu przed i po wprowadzeniu poprawek. W przypadku większości CWE największą liczbę błędów zgłosił Visual Code Greper. Natomiast najmniejszą liczbę błędów zaraportował FlawFinder. Szczególną uwagę zwrócono na przypadki, w których liczba błędów zmieniała się pomiędzy wersjami podatnymi i neutralnymi. W tym celu manualnie sprawdzono wszystkie zgłoszone błędy. Analiza ujawniła, że CppCheck przeprowadza bardziej dogłębną analizę oprócz prostego sprawdzania „niebezpiecznych" funkcji w stosunku do pozostałych analizatorów. Visual Code Grepper oprócz kodu analizuje również

**Tabela 3.4.** Wyniki uzyskane za pośrednictwem Orthogonal Defect Classification dla różnych typów podatności określonych standardem CWE.

| | Missing | Incorrect | **Suma** | Missing | Incorrect | **Suma** |
|---|---|---|---|---|---|---|
| | **CWE-787** | | | **CWE-20** | | |
| Checking | 53% | 16% | 68% | 45% | 9% | 55% |
| Algorithm/Method | 5% | 26% | 32% | 18% | 9% | 27% |
| Function | 0% | 0% | 0% | 0% | 9% | 9% |
| Interface | 0% | 0% | 0% | 0% | 9% | 9% |
| **Suma** | 58% | 42% | | 64% | 36% | |
| | **CWE-119** | | | **CWE-125** | | |
| Checking | 38% | 13% | 50% | 68% | 9% | 76% |
| Algorithm/Method | 38% | 13% | 50% | 6% | 9% | 15% |
| Function | 0% | 0% | 0% | 0% | 6% | 6% |
| Interface | 0% | 0% | 0% | 0% | 3% | 3% |
| **Suma** | 75% | 25% | | 74% | 26% | |
| | **CWE-190** | | | **CWE-476** | | |
| Checking | 14% | 14% | 29% | 56% | 24% | 80% |
| Algorithm/Method | 14% | 57% | 71% | 4% | 8% | 12% |
| Function | 0% | 0% | 0% | 0% | 0% | 0% |
| Interface | 0% | 0% | 0% | 0% | 8% | 8% |
| **Suma** | 29% | 71% | | 60% | 40% | |

komentarze - w celu znalezienia fraz wskazujących na uszkodzony lub niedokończony kod (np. szuka fraz TODO lub dotyczących konieczności wprowadzenia poprawek). FlawFinder przeprowadza jedynie proste kontrole pod kątem „niebezpiecznych" funkcji - jego praktyczne zastosowanie w celu poprawy bezpieczeństwa jest znikome (ta sama funkcjonalność jest realizowana przez inne narzędzia SAT jako uzupełnienie innych funkcjonalności). W dalszej części sekcji przedstawiono bardziej szczegółowo spostrzeżenia oraz przykłady. Oprócz aspektu wykrywania podatności, zaprezentowano również kilka przykładowych sugestii SAT, które mogą pomóc w prewencyjnej poprawie bezpieczeństwa. Są to głównie poprawki jakościowe i stylistyczne, które wpływają na przejrzystość kodu, a ich wprowadzenie może w przyszłości zapobiec pojawieniu się nowych luk w zabezpieczeniach.

**FlawFinder** W przypadku większości grup CWE liczba błędów była dokładnie taka sama dla podatnych i neutralnych wersji plików: **CWE-787**: 7 ostrzeżeń, **CWE-20**: 9 z 1 CWE-20/CWE-807, **CWE-190**: 1, **CWE-119**: 0, **CWE-476**: 48. Jedynym wyjątkiem była grupa **CWE-125**, w której FlawFinder znalazł 23 błędy dla podatnych i 21 dla neutralnych wersji tych samych plików. Różnica wynikała z faktu, że poprawka CVE-2020-26267 usunęła niepotrzebne wywołania „strings::StrCat" [169]. Znaleziono błędy dotyczące następujących identyfikatorów CWE: **CWE-120** („Buffer Copy without

**Tabela 3.5.** Liczba ostrzeżeń znalezionych przez FlawFinder, CppCheck i Visual Code. Każdy wiersz zawiera liczbę ostrzeżeń znalezionych dla plików zawierających instancje CVE należące do danego typu CWE, a nie liczbę podatności danego typu CWE. Konkretne typy CWE zwrócone dla analizatorów (jeśli taka informacja była dostępna) podano w szczegółowym opisie spostrzeżeń dotyczących danego narzędzia.

| | **FlawFinder** | | **CppCheck** | | **VCG** | |
|---|---|---|---|---|---|---|
| | Podatny | Neutralny | Podatny | Neutralny | Podatny | Neutralny |
| **CWE-787** | 7 | 7 | 7 | 7 | 16 | 16 |
| **CWE-20** | 9 | 9 | 16 | 16 | 27 | 27 |
| **CWE-190** | 1 | 1 | 2 | 2 | **15** | **16** ↑ |
| **CWE-119** | 0 | 0 | 10 | 10 | 0 | 0 |
| **CWE-125** | **23** | **21** ↓ | 27 | 27 | **61** | **60** ↓ |
| **CWE-476** | 48 | 48 | 35 | 35 | 79 | 79 |

checking Size of Input"), **CWE-327** („Use of a Broken or Risky Cryptographic Algorithm"), **CWE-20** („Improper Input Validation") i **CWE-807** („Reliance on Untrusted Inputs in a Security Decision"). Okazały się one fałszywymi alarmami (w określonym kontekście ostrzeżenie nie jest zasadne). Na przykład ostrzeżenia CWE-327 zostały zgłoszone dla użycia funkcji *random*. Funkcja ta nie była używana w kontekście bezpieczeństwa. Problemy związane z CWE-120 dotyczyły użycia funkcji *memcpy* i *StrCat*. Zdecydowana większość instancji *StrCat* mogła zostać usunięta, jako że funkcja *„InvalidArgument"* z „tensorflow/tensorflow/core/platform/errors.h" konkatenuje ciągi podane jako argumenty - nie jest więc wymagane wykorzystanie funkcji StrCat. Co więcej, większość ostrzeżeń związanych z *memcpy* wydaje się być fałszywymi alarmami i nie znajduje się w pobliżu rzeczywistych luk. Niemniej jednak, jeden z błędów *memcpy* zgłoszony przez SAT został uwzględniony wraz z innymi poprawkami w ramach modyfikacji związanej z podatnością CVE-2021-37637 (CWE-476) w commicie [170]. Poniżej przedstawiono treść błędu związanego z wykorzystaniem funkcji *memcpy*:

> „memcpy: Brak sprawdzania przepełnienia bufora podczas kopiowania do miejsca docelowego (CWE-120). Upewnij się, że miejsce docelowe zawsze może pomieścić dane źródłowe."

Poniżej przedstawiono również wprowadzoną poprawkę (żółte linie to te powodujące podatność, pozostałe to wprowadzona poprawka):

```
if (buffer) {
    memcpy(position, buffer->data(), buffer->size());
    metadata->set_tensor_size_bytes(buffer->size());
}
```

**CppCheck** Dla wszystkich badanych grup CWE wyniki były dokładnie takie same dla podatnych i neutralnych wersji plików (patrz Tabela 3.5). **W żadnym przypadku instancja prawdziwej podatności nie została znaleziona przez CppCheck**. Znaleziono błędy o następujących identyfikatorach CWE: **CWE-398** („Code Quality"), **CWE-563** („Assignment to Variable without Use"), **CWE-571** („Expression is Always True"), **CWE-197** („Numeric Truncation Error"). Ogólnie rzecz biorąc, zdecydowana większość ostrzeżeń była typu CWE-398 (Quality) i dotyczyła np. surowych pętli, zmiennej lokalnej przesłaniającej argument zewnętrzny, zmniejszania zakresów zmiennych - niektóre z nich były uzasadnione i zmiany mogłyby zostać wprowadzone. Z tego powodu jedną z takich sugestii załączono w pracy. Poprawkę można wykorzystać do poprawy jakości kodu. Dla plików ([171]) powiązanych z CVE-2021-29566 (CWE-787), CppCheck zwrócił następujące ostrzeżenie: „Local variable 'd' shadows outer argument (CWE-398)":

```
1 // TODO(gpapan): Write multi-threaded implementation
2  ...
3 void operator()(const CPUDevice& d, typename TTypes<T, 4>::
     ConstTensor input,
4  ...
5  for (int d = 0; d < depth; ++d)
```

Nie jest dobrą praktyką redefiniowanie argumentu funkcji i używanie go jako licznika w pętli. Aby uzyskać lepszą jakość kodu, należy użyć innej zmiennej. Z komentarza wynika, że planowane jest napisanie wielowątkowej implementacji tej funkcji i potencjalnie zostanie tam użyty *const CPUDevice& d* (który jest teraz redefiniowany). Podobne problemy dotyczyły np. zmiennej „i" w pliku „tensorflow/core/kernels/save_restore_v2_ops.cc". Jednoliterowe zmienne takie jak „i", „d" są często wykorzystywane jako liczniki pętli. Zawsze należy sprawdzić jednak, czy dana zmienna nie jest już zajęta. Nawet jeśli nadpisanie danej zmiennej nie powoduje podatności w danej chwili, to potencjalnie może spowodować jej wprowadzenie w przyszłości (np. gdy kod zostanie przebudowany przez innego programistę).

**Visual Code Grepper** Visual Code Grepper nie udostępnia w swoich raportach numerów identyfikacyjnych Common Weakness Enumeration. Udostępnia natomiast ostrzeżenia z przypisanymi znacznikami (np. „potencjalnie niebezpieczny", „podejrzany komentarz", „krytyczny"). Najwyższa zaobserwowana waga ostrzeżenia to „wysokie" - nie zaobserwowano przypadków z ostrzeżeniami „krytyczne". Oprócz analizy kodu, VCG przetwarza również komentarze i analizuje ich zawartość za pomocą analizy tekstu (np. wyszukuje terminy takie jak „TODO"). Może to być przydatne np. w przypadku planowanych funkcjonalności. W przypadku wyszukiwania „TODO", gdy nawias

jest używany tuż po "TODO", "TODO" nie jest wykrywane. Takie przypadki mogą być postrzegane jako fałszywe negatywy (ang. false negatives) - są to potencjalne części kodu, które powinny zostać ukończone lub naprawione. Z drugiej strony, VCG wyszukuje terminy takie jak „to do", co często prowadzi do zgłaszania fałszywych pozytywów (kiedy „to do" jest równe angielskiego „to perform", czyli wykonać jakąś akcję, zrealizować coś, robić - używane podczas tłumaczenia operacji wykonywanych przez dany fragment kodu). Z drugiej strony fałszywe pozytywy/alarmy oznaczają komentarze, które wyjaśniają części kodu i nie sugerują o konieczności wprowadzenia poprawek/zmian. Oczywiście, nie wszystkie TODO implikują kwestie bezpieczeństwa, ale powinny zostać przejrzane przez programistów, ponieważ mogą zawierać potencjalne zagrożenia bezpieczeństwa (niedokończony/niepoprawny kod), a nawet być „wskazówką" dla atakującego. Poniżej przytoczono kilka przykładowych fragmentów kodu, które są traktowane jako fałszywie pozytywne i fałszywie negatywne przykłady z pliku „tensorflow/stream_executor/cuda/cuda_dnn.cc" przed usunięciem luki CVE-2020-26270 (w celu znalezienia fałszywych negatywów TODO wykorzystano wyszukiwanie za pośrednictwem skrótu ctrl+f):

- Przykład fałszywie pozytywny:

```
// The reason that we only support the kBatchYXDepth
   output layout
// is that we have  to do
// something in the depth for each (y,x)
// coordinate.
```

- Przykład fałszywie negatywny:

```
LOG(FATAL) << "not yet implemented";  //  TODO(leary)
```

lub

```
//  TODO(yangzihao):
//Test with negative dilation to make sure that cudnn
// doesn't crash.
```

VCG znajduje również instancje StrCat i memcpy i sugeruje zastąpienie ich innymi funkcjami, ponieważ funkcje te pojawiają się na liście zakazanych funkcji firmy Microsoft. VCG zwraca ostrzeżenia, gdy znajdzie wiele operacji delete dla określonej zmiennej lub gdy nie może znaleźć operacji delete dla nowej zmiennej. Były to fałszywe alarmy - dotyczyły sytuacji, w których wskaźnik został utworzony wewnątrz funkcji, a następnie zwrócony. VCG wykrywa również przypadki użycia funkcji „getenv", która powinna być używana z ostrożnością. VCG zwraca ostrzeżenia dotyczące „Signed/Unsigned

Comparison" jako funkcjonalność w wersji beta: „To zachowanie może zwrócić nieoczekiwane wyniki, ponieważ liczby ujemne będą na siłę rzutowane na duże liczby dodatnie". W badanych przypadkach wszystkie ostrzeżenia wydają się być fałszywymi alarmami, a nie rzeczywistymi zagrożeniami bezpieczeństwa. Zmienne, o których mowa, są tworzone wewnątrz pętli for i inkrementowane z wyraźną górną granicą (nie występują wartości ujemne). Co więcej, inną wartość ostrzeżeń można zaobserwować dla grupy CWE-125 w Tabeli 3.5. Podobnie jak w przypadku programu FlawFinder, program VCG wykazał, że jedno ostrzeżenie StrCat zostało usunięte w ramach naprawy podatności. W przypadku CWE-190 liczba błędów jest większa w przypadku neutralnym. Wynika to z faktu, że poprawka podatności CVE-2021-29605 spowodowała pojawienie się ostrzeżenia dotyczącego malloc bez free. Jest to fałszywy alarm - pamięć jest alokowana dla zmiennej zwracanej przez funkcję.

### Analiza manualna opisów podatności i związanego z nimi kodu

**CWE-119 - Improper Restriction of Operations within the Bounds of a Memory Buffer** Głównymi źródłami podatności CWE-119 są *brak/niepełna/niewłaściwa walidacja parametrów wejściowych funkcji* wykorzystywane do uzyskiwania dostępu i przetwarzania struktur danych. Przykładem może być podatność CVE-2020-15205 spowodowana brakiem walidacji jednego z argumentów, przez co możliwe było przekazanie wartości mogących spowodować błąd przepełnienia sterty/wyciek zawartości pamięci, która może zawierać adresy zwrotne i może zostać wykorzystana do obejścia mechanizmu randomizacji układu przestrzeni adresowej (ang. Address Space Layout Randomization). Innym problemem może być *brak/niepełna/nieprawidłowa walidacja obliczanych wartości* wykorzystywanych do uzyskiwania dostępu i przetwarzania danych. Problem dotyczy także sytuacji, w których walidacja jest zaimplementowana dla trybu debugowania, ale nie dla trybu release. Występują również problemy związane z *naśladowaniem indeksowania Pythona z wartościami ujemnymi*. Przykładem może być CVE-2020-15207 - aby naśladować indeksowanie Pythona z wartościami ujemnymi, TFLite konwertuje wartości ujemne na dodatnie. Jedyne sprawdzenie przekonwertowanego indeksu występuje w trybie debugowania. W trybie release wykonywanie kodu jest kontynuowane z ujemnym indeksem. Również *brak/niepełna/niewłaściwa walidacja indeksów uzyskanych przy użyciu wzorca podwójnego indeksowania* mogą być źródłem podatności (np. dostęp do wartości gradientów w odwróconej kolejności w podatności CVE-2020-15195). Kolejnym problemem jest *brak/niepełna/niewłaściwa walidacja kształtów badanych tensorów*. Problemy często pojawiają się, gdy struktury danych są przetwarzane równolegle (dzieje się tak np. w przypadku CVE-2020-15198). Innym problemem

jest *brak walidacji, czy wartości podane jako parametry nie są zbyt duże dla typów używanych w funkcji i dla alokacji pamięci*. Przykładem może być CVE-2020-15266 - gdy jeden z argumentów jest bardzo duży, implementacja kernela CPU odczytuje go jako wartość zmiennoprzecinkową C++ „nan". Ostatni błąd znaleziony w analizie dotyczy *braku/niekompletnego/niewłaściwego sprawdzania, czy zainicjalizowane wartości są tworzone poprawnie*.

**CWE-787 - Out-of-bounds Write** Podatności te mogą być spowodowane przez *brak/niepełną/nieprawidłową walidację parametrów wejściowych* i zawierać *nieprawidłowe typy*, *nieprawidłowe wartości* lub *puste zmienne* jako argumenty (np. CVE-2021-29535 - zakłada się, że argumenty wejściowe są prawidłowymi skalarami, więc ich wartości liczbowe są bezpośrednio odczytywane. W przypadku pustych tensorów dostęp do indeksu 0 powoduje przepełnienie). Innym źródłem jest *brak/niepełna/błędna walidacja wartości obliczonych wewnątrz funkcji*. Przykładem może być CVE-2021-29540 - obliczony rozmiar filtra nie jest walidowany i, jako rezultat, jest on wykorzystywany zamiast faktycznej liczby elementów filtra podczas wstecznej propagacji filtra konwolucyjnego. Może to skutkować przepełnieniem, gdy obliczony rozmiar jest zbyt duży. Innym źródłem może być brak walidacji czy *struktury danych nie są puste* lub *brak sprawdzenia, czy struktura danych ma wystarczającą liczbę elementów* dla pewnych operacji. Przykładem może być podatność CVE-2021-29578, której powodem był brak sprawdzenia, czy argumenty sekwencji pooling mają wystarczającą liczbę elementów. Gdy w operacji używany jest więcej niż jeden tensor, źródłem może być *brak walidacji zgodności kształtu tensorów* (np. CVE-2021-29609 - nie jest sprawdzane, czy drugi wymiar tensorów rzadkich używanych w operacji jest zgodny). Problem może być również związany z *brakiem/niewłaściwą walidacją modeli uczenia maszynowego*, a specjalnie spreparowane modele mogą spowodować wykorzystanie podatności.

**CWE-125 - Out-of-bounds Read** Podatności te mogą być spowodowane przez *brak/niepełną/niewłaściwą walidację parametrów podanych przez użytkownika*. Problemy te mogą dotyczyć *wartości wymiarów* (np. CVE-2020-15265 - można przekazać nieprawidłową wartość wymiaru, co skutkuje dostępem do wymiaru poza zakresem tensora), *brak sprawdzania poprawności łańcuchów* (np. reprezentujących format - CVE-2020-26267), *brak sprawdzania granic tensora dla operacji tensorowych* (liczby ujemne, liczby przekraczające rozmiar), *brak sprawdzania, czy parametry używane w specyfikacji określonej struktury danych są poprawne dla jej typu*. Inne problemy wiążą się z *brakiem/niekompletną/nieprawidłową walidacją struktur danych podanych przez*

*użytkownika - (zwykle tablic)*: *puste sprawdzenia*. Ponadto *brak/niekompletna/niewłaściwa walidacja kompatybilności struktur danych* może wprowadzać podatności, gdy wszystkie te struktury są używane razem w operacji (również równolegle). Niezgodność może często skutkować powiązaniem referencji ze wskaźnikiem null (np. CVE-2021-37655). Bardzo duże wartości parametrów i tensorów mogą powodować przepełnienia liczb całkowitych, skutkujące odczytem poza granicami i ujawnieniem zawartości pamięci procesu - np. CVE-2018-21233. Problem może być spowodowany przez *brak sprawdzenia poprawności wymiarów tablic*. Problemy mogą być powodowane przez *brak kontroli nad inkrementacją wskaźników* i próbę uzyskania dostępu do danych (np. CVE-2021-29532) lub *użycie jednej zmiennej do indeksowania dwóch oddzielnych tablic równolegle*, a także *implementację kontroli jedynie za pośrednictwem typu debug*. Również *brak/niepełna/niewłaściwa walidacja modeli* używanych w operacjach może powodować podatności. Ostatnią kwestią wyszczególnioną w analizie jest *niewłaściwa kontrola przepływu programu* (np. CVE-2021-29551 - implementacja nie kończy wykonywania jądra, jeśli jeden warunek walidacji nie powiedzie się i jedynie powraca z funkcji).

**CWE-190 - Integer Overflow or Wraparound** Jednym ze źródeł podatności CWE-190 może być *używanie niebezpiecznych funkcji*, np. „InitDims" zamiast „AddDimWithStatus" (lub „BuildTensorShapeBase‘ w konstruktorze tensora), co może skutkować błędami „CHECK" w obecności przepełnień. Funkcja ta jest nadal używana przez niektóre API, np. w pliku „mkl_util.h" („tensorflow/tensorflow/core/util/") - stanowi to potencjalną podatność. Innym źródłem może być *brak/niewłaściwa walidacja rozmiarów tensorów i modeli* - proste operacje (np. konkatenacja - CVE-2021-29601) używane z dużymi modelami w TFLite mogą powodować przepełnienia (nawet częściej niż w tradycyjnym TensorFlow, gdyż w tej implementacji często używa się typu int32 zamiast int64). Przykładem może być CVE-2021-29605 (kod TFLite używany do alokacji tablic jest podatny na przepełnienie liczb całkowitych - specjalnie przygotowany model dużych rozmiarów może spowodować, że mnożnik „size" zwróci wartość, która przepełni typ danych „int" i stanie się ujemna. Ta ujemna wartość jest przekazywana do „malloc" w wyniku czego następuje odwołanie do nieprawidłowego wskaźnika. Innym przykładem może być CVE-2021-41198 - liczba elementów w tensorze wyjściowym jest zbyt duża dla „int64_t", następuje przepełnienie, które jest wykrywane przez instrukcję „CHECK", a proces zostaje przerywany.

**CWE-476 - NULL Pointer Dereference** Podatności te są często powodowane przez *problemy z sesją* (brak walidacji dostępności stanu sesji, stare

wersje TensorFlow nie ustawiają tego stanu - np. CVE-2020-15204), *brak/nieprawidłową walidację typu tensora* (konwersja z tablicy Pythona na tablicę C++ jest podatna na pomylenie typów - CVE-2021-29513) i *brak/nieprawidłową walidację kształtu tensora* (np. CVE-2021-41215 - zostały założone dodatni rząd tensora oraz poprawność jego ostatniego wymiaru). Problemy mogą zostać wprowadzone przez *brak/nieprawidłową/niekompletną walidacja, czy tensory wejściowe nie są puste lub istnieją* (np. CVE-2021-41217 - budowanie grafu przepływu sterowania może powodować wyjątek wskaźnika null, gdy węzły nie mogą być sparowane). Problemy mogą wystąpić w przypadku *braku/niewłaściwej walidacji argumentów wejściowych pod względem ich dostępności i poprawności*. Kolejną przyczyną może być *brak/niewłaściwa walidacja przypadków szczególnych i wszystkich ograniczeń operatora* (np. CVE-2021-29616 - niezdefiniowane zachowanie może zostać wywołane z powodu dereferencji w przypadkach szczególnych, które skutkują optymalizacją węzła bez wejścia). Inną przyczyną może być *niewłaściwa kontrola przepływu programu*. Również korzystanie z nieobsługiwanych interfejsów API może wprowadzać luki w zabezpieczeniach (np. CVE-2021-41208 - nieobsługiwane drzewa wzmacniane). Problemem może być zaplanowana i niezaimplementowana walidacja (np. CVE-2021-29565 - brak walidacji pod „TODO").

**CWE-20 - Improper Input Validation** Głównymi źródłami podatności CWE-20 są *brak/niewłaściwe sprawdzenia, czy badane struktury danych nie są puste* lub *mają odpowiedni wymiar* oraz brak obsługi tych przypadków. Podatności mogą wystąpić, gdy implementacja zależy od domyślnych wartości osi, niektóre wartości opisujące badane struktury danych są używane do tworzenia innych struktur. Przykładem może być podatność CVE-2020-15199 spowodowana brakiem walidacji, czy badany tensor ma minimalną liczbę elementów wymaganą do wykonania operacji. Ilość ta jest wykorzystywana do inicjalizacji innej struktury danych, która wymaga, aby przynajmniej jeden element nie był typu nullptr. Niewłaściwy kształt jednej struktury może wiązać się z ryzykiem, że inna struktura stanie się wskaźnikiem null. Innym problemem może być niespełnienie wszystkich założeń metody (np. CVE-2021-37674 - tensory wejściowe w tej konkretnej metodzie muszą być 4-wymiarowe, ale nie jest to sprawdzane). Ponadto niektóre metody zewnętrzne mogą nie być przygotowane do działania z pustymi strukturami danych. Przykładem może być CVE-2020-26270 - uruchomienie modelu LSTM/GRU, w którym warstwa LSTM/GRU otrzymuje dane wejściowe z wynikami zawierającymi 0 elementów. Rezultatem jest błąd CHECK, w przypadku wykorzystania mechanizmu CUDA.

### 3.1.3 Wnioski

Chociaż dedykowane zagrożenia związane ze sztuczną inteligencją są niewątpliwie niezwykle ważne i należy się nimi zajmować, zagrożenia spowodowane standardowymi lukami w C/C++ są równie ważne. Dlatego w pracy zdecydowano się zbadać standardowe podatności na zagrożenia w implementacji TensorFlow opisane za pomocą standardów CWE i CVE. Przeanalizowano 6 popularnych typów podatności, biorąc pod uwagę metadane, opisy i kody ich instancji. Ortogonalna klasyfikacja defektów, ręczna analiza kodu i opisy konkretnych CVE pozwoliły scharakteryzować ich przyczyny i skutki. Wyniki pokazały, że większość luk w zabezpieczeniach TensorFlow nie różni się całkowicie od luk w zabezpieczeniach produktów ogólnego przeznaczenia napisanych w języku C/C++. Mogą być jednak trudniejsze do znalezienia, ponieważ często wymagają głębszej interpretacji semantycznej i dobrego zrozumienia stosowanych metod matematycznych. Wydaje się to być najbardziej prawdopodobną przyczyną praktycznie zerowej skuteczności badanych programów SAT w wykrywaniu znanych podatności. Tylko jedno zgłoszone ostrzeżenie zostało uwzględnione w jednej z badanych poprawek podatności. Niektóre z ostrzeżeń można wykorzystać do poprawy jakości kodu, a co za tym idzie zmniejszenia szansy na wprowadzenie nowych błędów. Niektóre komercyjne narzędzia SAT mogą być bardziej skuteczne, jednak zdecydowano się użyć tylko tych narzędzi SAT, które są dostępne dla wszystkich badaczy oraz programistów.

Ponieważ automatyczna analiza zawodzi, społeczność badawcza powinna skupić się bardziej na przeciwdziałaniu wprowadzania i naprawianiu istniejących luk. Istnieje konieczność proponowania nowych rozwiązań z zakresu przeciwdziałania podatnościom oprogramowania napisanego w językach C/C++. **Wyniki opisane w niniejszej sekcji zostały opublikowane w pracy [172].** Na podstawie wyników analizy w załączniku do wspomnianego artykułu zamieszczono listę kontrolną dla programistów/użytkowników TensorFlow (i innych bibliotek numerycznych i DL). Można ją wykorzystać do sprawdzenia kodu źródłowego i potencjalnego wyeliminowania niektórych typowych błędów programistycznych. Lista ta może być również wykorzystywana przez programistów platform innych niż te wykorzystujące sztuczną inteligencję wraz z innymi wytycznymi dotyczącymi bezpiecznego programowania, ponieważ wiele z opisanych problemów dotyczy również aplikacji ogólnego przeznaczenia napisanych w języku C/C++. Ponadto, lista kontrolna może być przydatna dla twórców nowych testów SAT - jako źródło inspiracji dla nowych reguł i do budowania przypadków testowych. Z tego powodu artykuł [172] jest wartościowym źródłem informacji dla osób rozpoczynających pracę z bezpieczeństwem oprogramowania ze względu na liczne

przykłady, komentarze i praktyczne podejście do opisywanych problemów. Analiza pokazuje również, że dla efektywnej ewaluacji narzędzia do analizy statycznej powinny być testowane na nowoczesnych i wymagających produktach oprogramowania, takich jak TensorFlow.

## 3.2 Wykorzystanie metryk statycznych analizatorów kodu do wykrywania luk bezpieczeństwa w kodzie napisanym w językach C/C++

Analiza krajobrazu zagrożeń związanych z biblioteką numeryczną TensorFlow przeprowadzona w Sekcji 3.1, dostarczyła wielu cennych wniosków. Najczęstsze podatności występujące w tej, jednej z najbardziej popularnych, bibliotece numerycznej w dużym stopniu pokrywają się z najbardziej powszechnymi podatnościami oprogramowania C/C++ niezależnie od zastosowania. Dlatego też zdecydowano się na dalszą eksplorację tego tematu, przenosząc badania na inne aplikacje napisane w językach programowania C/C++. Chociaż analizowane w tej części pracy oprogramowanie nie jest bezpośrednio związane z uczeniem maszynowym, to ze względu na podobieństwo najczęstszych typów podatności oprogramowania w różnych produktach C/C++ oraz powszechne wykorzystanie właśnie tych języków do tworzenia bibliotek numerycznych - podstawy systemów inteligentnych - kierunek badań jest zasadny i może przynieść wartościowe wnioski. Podobieństwa w strukturach językowych w różnorodnym oprogramowaniu napisanym w C/C++, od bibliotek do uczenia maszynowego po systemy operacyjne i przeglądarki internetowe, sugerują, że można oczekiwać podobnych rodzajów zagrożeń w szerokim spektrum zastosowań. Największą zaletą rozszerzenia analizy o produkty tradycyjne jest fakt, że pozwala to na wykorzystanie większej próbki danych w analizie, niż jedynie pojedynczych bibliotek, a także potencjalnie większą moc generalizacji uzyskanych wyników, co z kolei może prowadzić do bardziej uniwersalnych wniosków dotyczących bezpieczeństwa systemów inteligentnych, które oprócz samych algorytmów sztucznej inteligencji wykorzystują także tradycyjne elementy oprogramowania.

Podatności bezpieczeństwa są często wprowadzane na etapie tworzenia kodu cyklu życia oprogramowania. Luki w zabezpieczeniach są trudne do wykrycia, dopóki nie spowodują awarii bezpieczeństwa na etapie operacyjnym SDLC. Problemy związane z bezpieczeństwem nie zawsze są rozwiązywane lub znane wcześniej. W związku z tym niezwykle ważne byłoby określenie ze-

stawu metryk/cech, które mogą pomóc wskazać możliwe występowanie luk w zabezpieczeniach oprogramowania, tak aby testowanie na wczesnych etapach SDLC było skuteczne i pozwoliło usunąć lub naprawić lukę przed wprowadzeniem produktu na rynek. W rezultacie ułatwiłoby to programistom rozważenie kwestii bezpieczeństwa od najwcześniejszych etapów procesu rozwoju oprogramowania nawet bez specjalistycznej wiedzy w tym zakresie [173].

Analiza statyczna to proces badania systemu lub jego komponentów, który bierze pod uwagę jego formę, strukturę, zawartość lub dokumentację bez konieczności kompilacji i uruchamiania kodu [174]. Narzędzia do analizy statycznej służą do szukania problemów w implementacji, zwykle w oparciu o predefiniowany zestaw reguł, które reprezentują potencjalne anomalie. Takie nieprawidłowości często występują w kodzie źródłowym. Automatyczne narzędzia do analizy statycznej nazywane są Automatyczną Analizą Statyczną (ang. Automatic Static Analysis, ASA). Ostrzeżenie ASA to pojedynczy raport z narzędzia ASA, który wskazuje obszar w kodzie, który łamie predefiniowaną regułę analizy statycznej. Typ ostrzeżenia określa regułę, która została złamana. Tego typu informacje były wykorzystane (z niską skutecznością) w celu wykrywania znanych luk w oprogramowaniu C/C++ w Sekcji 3.1. Ze względu na uogólnienia dokonane podczas dopasowywania reguł, narzędzia ASA często generują wysoki wskaźnik wyników fałszywie pozytywnych, a ich alerty muszą być sprawdzane przez ekspertów zajmujących się bezpieczeństwem [175], co potwierdzono w Sekcji 3.1. Analiza kodu, oprócz pojedynczych alertów, może dostarczyć różnorodne metryki oprogramowania, które mierzą pewne właściwości kodu źródłowego. Przykładami metryk tradycyjnych są: metryka rozmiaru (ang. size), metryka złożoności (ang. complexity) [115], złożoność, sprzężenie i spójność (ang. complexity, coupling and cohesion - CCC) [126], a także metryki dotyczące aktywności programistów - Code Churn, Developer Activity [83]. Nowoczesne statyczne analizatory kodu oferują wiele różnorodnych metryk kodu: tradycyjne metryki (rozmiar, złożoność itp.) oraz wiele innych metryk dotyczących liczby błędów znalezionych w analizie, łatwości utrzymania kodu i jego niezawodności itp.

Firmy wytwarzające oprogramowanie często korzystają z testów analizy statycznej, ponieważ pozwalają one poprawić jakość oprogramowania jeszcze na etapie tworzenia kodu w SDLC [95]. Analiza statyczna w ograniczonej formie może być wykonywana przy użyciu zintegrowanych środowisk programistycznych, np. Visual Studio [176] dla C/C++ lub IntelliJ IDEA [177] dla Javy. Na rynku dostępne są również dedykowane narzędzia: np. Veracode [178] i SonarQube [179]. Ze względu na popularność tych narzędzi w praktyce, cenne byłoby wykorzystanie ich w procesie tworzenia nowych rozwiązań poprawiających bezpieczeństwo oprogramowania. Wprowadzenie przewidywania podatności (zwykle rozumianej jako binarnej klasyfikacji podatnych i

neutralnych części kodu źródłowego) pozwala zmniejszyć liczbę fałszywych ostrzeżeń narzędzi ASA i skoncentrować ograniczone wysiłki testowe na potencjalnie podatnych plikach [126]. Niestety, na chwilę obecną analiza statyczna nie jest zbyt skuteczna, dlatego też należy włożyć więcej wysiłku w tworzenie dokładnych i wydajnych rozwiązań. Ze względu na popularność statycznych analizatorów kodu w firmach programistycznych, uzasadnione jest wykorzystanie nie tylko oferowanych przez nie alertów, ale także metryk wynikowych do budowy modeli przewidywania podatności oprogramowania. Uzasadnione wydaje się również wykorzystanie modeli uczenia maszynowego do tworzenia systemów przewidywania podatności na ataki w oparciu o dane z narzędzi do analizy statycznej. Ponieważ modele uczenia maszynowego często działają lepiej, gdy do uczenia wykorzystywane są tylko najlepsze cechy, także aspekt selekcji cech powinien być brany pod uwagę w tym rozwiązaniu. Dodatkową zaletą stosowania selekcji cech jest to, że powstałe modele są zazwyczaj mniejsze i bardziej wydajne w działaniu ze względu na wykorzystanie mniejszej liczby cech.

W niniejszej pracy, metryki ze statycznych analizatorów kodu zostały wykorzystane jako cechy służące do przewidywania podatności komponentów oprogramowania napisanego w językach z rodziny C/C++. Celem było sprawdzenie, czy wygenerowane przez analizatory metryki (a nie bezpośrednio kod źródłowy, czy też alerty SAT) mogą zostać wykorzystane jako wskaźniki występowania podatności w kodzie. Do testowania wykorzystano dwa typy podatności sklasyfikowane przy użyciu standardu Common Weakness Enumeration (CWE). Jeden z typów - CWE-119 - jest jednym z najczęstszych typów znanych podatności w oprogramowaniu TensorFlow, co zostało opisane w Sekcji 3.1. Dokładność wykrywania badanych typów podatności testowano razem i osobno w celu zbadania umiejętności wyłapywania przez modele wzorców charakterystycznych dla konkretnych typów podatności oraz tych ogólnych. Wykorzystano zbiór danych wprowadzony w publikacji [16]. Przeprowadzono kompleksową analizę i selekcję cech, a także ocenę przy użyciu 13 modeli uczenia maszynowego. Uwzględniono trzy rodzaje analizy korelacji i cztery techniki selekcji cech. Do wygenerowania cech wykorzystanych w eksperymentach zastosowano komercyjny program SonarQube z wtyczką stworzoną przez społeczność [179] oraz projekt badawczy - CCCC [180, 181].

### 3.2.1 Proponowane rozwiązanie i metodologia badań

W pracy wykorzystano cechy opisujące kod C/C++ wygenerowane za pośrednictwem dwóch badanych statycznych analizatorów kodu: SonarQube [179] oraz CCCC [180, 181]. Wartości wynikowe metryk z dwóch analizatorów, uzyskane dla wykorzystanych w pracy elementów kodu, zapisano w

postaci plików csv. Każdy wiersz pliku csv zawierał pole identyfikujące dany element kodu, uzyskane metryki oraz binarną etykietę. Pliki csv stworzono osobno dla badanych typów podatności, a także złączono je do wspólnego pliku w celu analizy łączonej wartości metryk dla tych typów podatności kodu. Zbiór danych przetworzony do takiej formy wykorzystano w dalszych częściach analizy - związanych z selekcją cech oraz algorytmami uczenia maszynowego wykorzystanymi do wykrywania podatności.

### Selekcja cech

Zbiory danych z dużą liczbą atrybutów często zawierają cechy niepotrzebne dla danego zadania. Skutkiem jest występowanie „szumu" utrudniającego efektywne trenowanie algorytmów uczenia maszynowego. Jest to spowodowane w dużej mierze koniecznością znajdowania przydatnych wzorców w wielowymiarowej przestrzeni cech. Redukcja wymiarowości tej przestrzeni często skutkuje bardziej efektywnym trenowaniem modeli, a co za tym idzie - ich wyższą dokładnością oraz krótszym czasem trenowania. Zastosowanie technik selekcji cech może także przeciwdziałać zbytniemu dopasowaniu modeli do zbioru treningowego, czyli mieć pozytywny wpływ na generalizację.

Aby ocenić jakość cech, wykorzystano trzy rodzaje powszechnie stosowanych technik analizy korelacji: Korelację Pearsona, Korelację Spearmana oraz Korelację Kendalla. Wykorzystano także trzy rodzaje metod rankingowych opartych na entropii (przyrost informacji, współczynnik przyrostu informacji i wskaźnik spadku Giniego) oraz technikę rankingu $\chi^2$. Celem analizy korelacji w tej pracy jest wykrycie, czy istnieją cechy, które wykazują statystycznie istotną korelację z atrybutem klasy. W tym celu, po obliczeniu wartości współczynników korelacji, przeprowadzono analizę istotności. Podobne podejście zastosowano w pracy [182]. Wszystkie z wymienionych metod mogą być wykorzystywane do oceny jakości cech i ich sortowania w celu selekcji cech, jednak ich charakter znacząco się różni. Z tego powodu dobrą praktyką jest porównanie rezultatów uzyskanych dla różnych metod.

**Analiza korelacji.** Techniki korelacji są powszechnie stosowane do badania związków między zmiennymi, zwłaszcza ciągłymi. Współczynniki korelacji mierzą różne typy związków. Współczynniki korelacji Spearmana lub Kendalla wskazują na występowanie zależności monotonicznych (nie muszą być liniowe) między dwiema zmiennymi, w przeciwieństwie do współczynników korelacji Pearsona, które określają jedynie zależność liniową. Współczynnik korelacji Pearsona jest miarą parametryczną, a współczynniki korelacji Spearmana i Kendalla są nieparametryczne. Możliwe jest traktowanie zmiennych porządkowych jako zmiennych ciągłych, ale może to wprowadzić potencjalnie

niepoprawne oszacowanie miar korelacyjnych, zwłaszcza gdy istnieje niewiele kategorii porządkowych. Problem ten dotyczy głównie korelacji Pearsona, która nie powinna być stosowana do zmiennych porządkowych [183]. Niemniej jednak, podejścia oparte na korelacji Pearsona są odporne i często mogą z powodzeniem znaleźć związek liniowy nawet wtedy, gdy tradycyjne założenie jest naruszone [184]. Z tego powodu na potrzeby pracy zdecydowano się pokazać wyniki wszystkich współczynników korelacji. Należy jednak podkreślić, że w przypadku zmiennych porządkowych (zwłaszcza tych o małej liczbie wartości) bezpieczniej jest użyć jednego ze współczynników korelacji rangowej (Spearmana lub Kendalla).

W pracy zastosowano wyżej wspomniane powszechnie stosowane metody korelacji, aby uzyskać wartości współczynników korelacji i odpowiadające im p-wartości. Następnie, aby przetestować istotność współczynnika korelacji, przeprowadzono test hipotezy przy użyciu wartości p. Poniżej zdefiniowano hipotezy testowe:

- $H_0$: Korelacja między daną cechą a etykietą jest znacząca;
- $H_1$: Korelacja między daną cechą a etykietą nie jest znacząca;

Przyjęto standardowy poziom ufności $\alpha = 0,05$. Hipoteza zerowa jest przyjmowana, gdy p-wartość jest mniejsza niż $\alpha$, a odrzucana w przeciwnym wypadku. Porównując p-wartość z $\alpha$ stwierdzono, czy hipoteza zerowa H0 powinna zostać odrzucona.

**Pozostałe techniki selekcji cech.** W pracy wykorzystano cztery standardowe techniki selekcji cech: trzy miary oparte na entropii oraz metrykę $\chi^2$. Wykorzystane techniki selekcji cech opisano bardziej szczegółowo w dalszej części sekcji. W przeciwieństwie do analizy korelacji, techniki selekcji cech oparte na entropii i technika rankingu $\chi^2$ mogą być stosowane w przypadku wszystkich typów cech i opierają się na statystycznych właściwościach zmiennych. Metody te są powszechnie stosowane w domenie selekcji cech [182]. Niemniej jednak metody te również charakteryzują się pewnymi wadami. Technika rankingu $\chi^2$ nie sprawdza się dobrze w przypadku występowania rzadkich wartości w danych [185], a Information Gain faworyzuje cechy o wielu równomiernie rozłożonych wartościach [186]. Z tego powodu dobrze jest przetestować różne rodzaje selekcji cech i przeprowadzić dodatkową ocenę, np. ocenę opartą na uczeniu maszynowym, która jest również przeprowadzana w [182] oraz w niniejszej pracy.

**Information Gain, Zysk informacyjny** to technika statystyczna pierwotnie używana do wyboru atrybutów używanych w kolejnych węzłach drzewa decyzyjnego w algorytmie ID3 [186]. Zysk informacyjny określa skutecz-

ność wykorzystania określonej cechy do oddzielenia próbek zgodnie z ich przynależnością do klasy. Miara ta opiera się na entropii, jednym z podstawowych pojęć w teorii informacji. Entropię dla klasyfikacji binarnej można zdefiniować w następujący sposób:

$$Entropia(S) = -p_1 log_2 p_1 - p_0 log_2 p_0, \tag{3.1}$$

gdzie $p_0$ i $p_1$ to prawdopodobieństwa, że próbka należy odpowiednio do klasy negatywnej i pozytywnej. Dla klasyfikacji wieloklasowej, wzór przyjmuje formę:

$$Entropia(S) = -\sum_{i=1}^{N} p_i log_2 p_i, \tag{3.2}$$

gdzie $p_i$ jest prawdopodobieństwem przynależności próbki do $i$-tej klasy. Information Gain można zdefiniować w następujący sposób:

$$InfoGain(S, A) = Entropia(S) - \sum_{v \in V_A} \frac{|S_v|}{|S|} Entropia(S_v), \tag{3.3}$$

gdzie $V_A$ jest zbiorem możliwych wartości cechy A, $S_v$ jest podzbiorem $V_A$, dla którego cecha A przyjmuje wartość v. Przy tak zdefiniowanej mierze opisuje ona zmniejszenie entropii. Jest to oczekiwane, gdy badany atrybut jest używany do podziału zbioru danych według klas [186].

**Gain Ratio, Współczynnik przyrostu informacji** to kolejna technika wyboru atrybutów decyzyjnych, która wykorzystuje wcześniej zdefiniowaną miarę Information Gain, a także miarę zdefiniowaną w następujący sposób [186]:

$$SplitInfo(S, A) = -\sum_{i=1}^{N} \frac{|S_i|}{|S|} log_2 \frac{|S_i|}{|S|}, \tag{3.4}$$

gdzie $S_i$ jest podzbiorem instancji, wynikiem podziału zbioru $S$ na $N$ klas. Metryka mierzy entropię S, ale bierze pod uwagę liczbę charakterystycznych wartości obserwowanej cechy. Współczynnik przyrostu informacji można zdefiniować następująco [186]:

$$GainRatio(S, A) = \frac{InfoGain(S, A)}{SplitInfo(S, A)} \tag{3.5}$$

**Gini Decrease, Spadek Giniego** lub spadek nieczystości Giniego można interpretować jako spadek nieczystości między kolejnymi węzłami w lesie losowym [187]. Nieczystość ta określa prawdopodobieństwo uzyskania dwóch instancji oddzielnych klas w dwóch losowaniach, przy założeniu, że rozkład instancji jest wielomianowy. Przykłady ze zbioru danych (można go postrzegać jako węzeł, używając terminologii drzewa losowego, $w$) są podzielone na

dwie części (na dwa węzły potomne, $w_1$ i $w_2$). Spadek nieczystości Giniego można ocenić w następujący sposób [187]:

$$\delta i(w; v; \eta_v) = i(W) - \frac{n_{w_1}}{n_w} i(w_1) - \frac{n_{w_2}}{n_w} i(w_2), \tag{3.6}$$

gdzie $v$ to rozważana cecha, a $\eta$ to próg dla tej wartości. Cechy są wybierane tak, aby zmaksymalizować redukcję zanieczyszczenia.

**Statystyka $\chi^2$** mierzy różnicę między zaobserwowaną liczbą wystąpień cechy $f$ dla danej klasy a wartością oczekiwaną. Zakłada się, że żadna cecha nie jest zależna od etykiety $c$. Miarę można zdefiniować w następujący sposób [188]:

$$\chi^2(f) = \sum_{c=1}^{k} \sum_{i=1}^{m} \frac{(O(f = v_i, c) - \mathbb{E}(f = v_i, c))^2}{\mathbb{E}(f = v_i, c)}, \tag{3.7}$$

gdzie $v_i$ jest kategorią, $k$ - liczbą klas, a $m$ - liczbą wystąpień cechy $f$. $O(f = v_i, c)$ jest liczbą wystąpień cechy $v_i$ o wartości $c$. Może to być wykorzystane w teście niezależności zmiennej losowej. Zdefiniujmy hipotezę zerową jako $H_0$ (zmienne losowe są niezależne) i alternatywną - $H_1$ (badane zmienne nie są niezależne). Im większa wartość statystyki $\chi^2(f)$, tym większa szansa, że badana zmienna losowa jest skorelowana z klasą decyzyjną i hipotezę zerową należy odrzucić.

### Ewaluacja jakości cech za pośrednictwem algorytmów uczenia maszynowego

Wykorzystanie wielu heterogenicznych modeli uczenia maszynowego w nadzorowanym zadaniu i obserwacja ich wydajności może być traktowana jako wskaźnik „wartościowości" zbioru danych treningowych i jego cech. Akceptowalne wyniki większości algorytmów mogą być wskaźnikiem przydatności użytych cech do wykonywania danego zadania [189]. Metody selekcji cech stosowane w pracy mają zróżnicowany charakter i generują różne podzbiory cech jako „najlepsze cechy". Dlatego też zastosowano modele uczenia maszynowego także w celu przetestowania skuteczności metod selekcji cech. Aby ocenić skuteczność różnych modeli uczenia maszynowego, zastosowano 5-krotną walidację krzyżową. Zastosowano trzynaście różnorodnych algorytmów uczenia maszynowego, w tym osiem prostych: drzewa decyzyjne, algorytm K-najbliższych sąsiadów (ang. k-Nearest Neighbours, KNN), regresję logistyczną, Kernel Naive Bayes, maszyny wektorów nośnych (ang. Support Vector MAchines, SVM). Przetestowano kernele liniowe, kwadratowe, sześcienne i gaussowskie. W pravy wykorzystano również pięć modeli zespołowych: Boosted Trees, Bagged Trees, Subspace Discriminant, Subspace KNN and RUS-Boosted Trees. Jako wskaźniki wydajności modelu zostały wykorzystane trzy

standardowe miary uczenia maszynowego: *Dokładność*, *Czułość* (ang. Recall, True Positive Rate, TPR, Sensitivity) i *Specyficzność* (ang. True Negative Rate, TNR). Dokładność pozwala ocenić ogólną skuteczność modelu, a dwie dodatkowe metryki koncentrują się na poszczególnych częściach predykcji - poprawności przewidywania elementów podatnych (Recall) i neutralnych (Specificity) [190].

Niektóre modele uczenia maszynowego, tzw. modele białoskrzynkowe, pozwalają na uzyskanie interpretacji procesu predykcji, a w rezultacie na jeszcze bardziej szczegółową ocenę wiarygodności cech. Jest to związane z aspektem wyjaśnialności sztucznej inteligencji. W pracy zdecydowano się zaprezentować graficzne reprezentacje drzew decyzyjnych wytrenowanych na dwóch podzbiorach danych, z których pierwszy uwzględniał jedynie luki w zabezpieczeniach CWE-119, a drugi - luki w zabezpieczeniach CWE-399. Wykorzystano wszystkie 33 cechy, aby pozwolić modelowi zdecydować, które cechy wykorzysta do podjęcia decyzji. Wiedza ta może zostać wykorzystana przez ekspertów do oceny bezpieczeństwa elementów kodu (lub przynajmniej dać im pewien wgląd w kwestie, które są wybierane przez modele jako najsilniejsze wskaźniki podatności).

### Wykorzystany zbiór danych

Na potrzeby analizy wykorzystano publiczny zbiór danych dotyczący podatności w oprogramowaniu napisanym w językach C i C++ - VulDeePecker (Code Gadget Database) [16]. Zbiór danych dostępny jest za pośrednictwem repozytorium GitHub pod adresem `https://github.com/CGCL-codes/VulDeePecker`. Zbiór danych zawiera dwa typy podatności opisane za pośrednictwem standardu Common Weakness Enumeration (CWE). Są to typy Improper Restriction of Operations within the Bounds of a Memory Buffer (CWE-119) oraz Resource Management Errors (CWE-399). Każdy element zbioru danych składa się z szeregu instrukcji programu, które są ze sobą powiązane zgodnie z przepływem danych. Kody zawarte w zbiorze danych zostały wprowadzone do badanych analizatorów kodu i za ich pośrednictwem wygenerowano zbiór danych wykorzystany w analizie. W zbiorze wynikowym zawarto 7534 rekordy danych. W Tabeli 3.6 przedstawiono szczegółowe informacje dotyczące liczności poszczególnych kategorii.

### Analizatory statyczne kodu

W celu ekstrakcji cech opisujących kod wykorzystano dwa narzędzia do statycznej analizy kodu: SonarQube [179] i CCCC [180, 181]. Narzędzia te są szeroko stosowane w literaturze do celów analizy kodu [191, 192, 193]. **So-**

**Tabela 3.6.** Liczność poszczególnych zbiorów elementów kodu wraz z informacją o podziale na klasy wykorzystanym w eksperymentach.

| Typ podatności | Rodzaj elementu | Liczba elementów | Suma |
|---|---|---|---|
| CWE-399 | Podatny | 684 | 1498 |
| | Neutralny | 814 | |
| CWE-119 | Podatny | 2702 | 6036 |
| | Neutralny | 3334 | |
| Razem | Podatny | 3386 | 7534 |
| | Neutralny | 4148 | |

**narQube** to narzędzie do automatycznego przeglądu kodu. Jest ono dostarczane przez szwajcarską firmę SonarSource. SonarQube umożliwia korzystanie z wielu języków. Analizatory statyczne dla niektórych języków (w tym C/C++) są poza zakresem wersji społecznościowej. Dlatego w analizie wykorzystano wtyczkę o otwartym źródle stworzoną przez społeczność [194] umożliwiającą statyczną analizę plików napisanych w językach C i C++. **CCCC** to darmowe narzędzie opracowane przez Tima Littlefaira. Jest to projekt badawczy, który koncentruje się na gromadzeniu metryk oprogramowania programu. Zapewnia proste metryki kodu dla wybranego pliku/projektu.

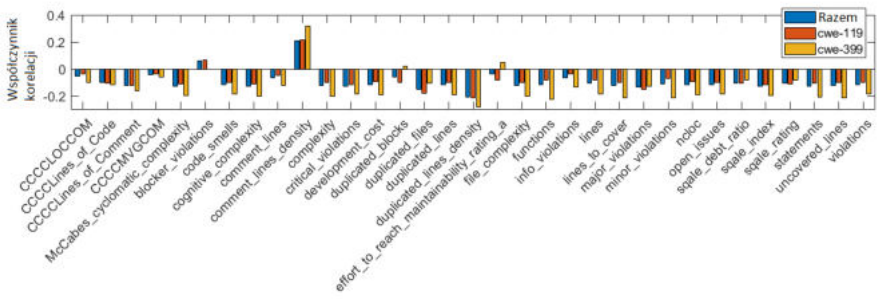
### Przeprowadzone eksperymenty

Na potrzeby pracy przeprowadzono różnorodne eksperymenty z trzema stworzonymi zbiorami danych (zbiory uwzględniające podatności typu CWE-399 i CWE-119 razem i osobno). Pierwszym krokiem po wygenerowaniu zbiorów danych z cechami uzyskanymi za pośrednictwem analizatorów kodu było przeprowadzanie na wygenerowanych plikach analizy korelacji. W tym celu wyznaczono wartości korelacji Pearsona, Spearmana oraz Kendalla między wszystkimi cechami a wartością etykiety (pliki neutralne - 0 oraz podatne - 1). Wyznaczono także p-wartości dla poszczególnych korelacji i przetestowano istotność współczynników korelacji. Wyznaczono również wartości dla różnych technik selekcji cech. Wartości przedstawiono w postaci wykresów słupkowych. Wykorzystano wymienione metody selekcji cech i korelację Spearmana do wygenerowania zestawów dziesięciu najlepszych cech dla danej metody. Cechy te wykorzystano jako wejście algorytmów uczenia maszynowego. Wyniki dla 13 modeli uczenia maszynowego uzyskane dla tych zestawów cech porównano z wynikami uzyskanymi dla bazowego zbioru (wszystkie cechy, brak selekcji cech). Wyniki dla 13 modeli przedstawiono w pracy zbiorczo poprzez wyznaczenie wartości średnich, minimalnych i maksymalnych, a

także odchylenia standardowego dla różnych podzbiorów cech. Dodatkowo wygenerowano drzewa decyzyjne w celu prezentacji ich struktury. Celem było interpretowalne podejście do uczenia maszynowego, a także dodatkowa – wizualna – metoda oceny przydatności cech.

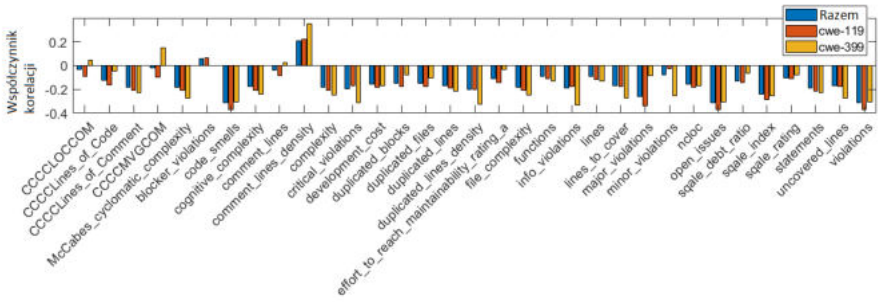
### 3.2.2 Wyniki

Na Rysunkach 3.2, 3.3 i 3.4 przedstawiono współczynniki korelacji dla różnych współczynników i podzbiorów zbioru danych, uwzględniając oba typy podatności razem, a także osobno. Wyraźnie widać, że wśród współczynników najwyższe wartości osiągane są dla podzbioru CWE-399. W większości przypadków korelacje są ujemne, a ich siła jest słaba lub średnia. Można również zauważyć, że wartości współczynników korelacji są zbliżone dla dwóch współczynników korelacji rangowej (Kendalla i Spearmana), a różnią się znacząco dla współczynnika Pearsona – niektóre z cech użytych w badaniu są wartościami porządkowymi z niewielką liczbą wartości.

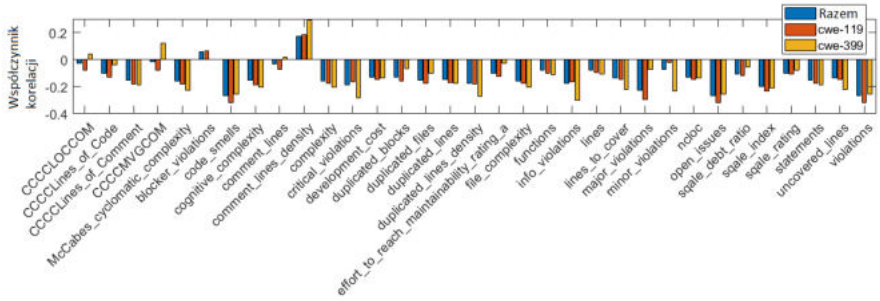



**Rysunek 3.2.** Wartości korelacji Pearsona przy użyciu różnych podzbiorów zbioru danych (CWE-119, CWE-399, razem) dla badanych cech. Na osi x umieszczono oryginalne nazwy cech.

Aby zbadać istotność korelacji, wykorzystano poziom ufności $\alpha = 0,05$ i określono, czy hipoteza zerowa $H_0$ powinna zostać odrzucona. W Tabeli 3.7 znajdują się wyniki p-wartości dla kolumn, w których była ona większa niż $\alpha$. W pozostałych przypadkach wartości p wynosiły $\ll 0.001$. Podkreślone, pogrubione wartości sugerują, że hipoteza zerowa $H_0$ powinna zostać odrzucona. Dla tych przypadków wnioskuje się, że nie ma wystarczających dowodów na to, że korelacja między konkretną cechą dla tego typu korelacji jest znacząca. W grupie cech, dla których hipoteza zerowa została odrzucona przynajmniej w jednym przypadku, znajdują się trzy cechy uzyskane z analizatora CCCC.




**Rysunek 3.3.** Wartości korelacji Spearmana przy użyciu różnych podzbiorów zbioru danych (CWE-119, CWE-399, razem) dla badanych cech. Na osi x umieszczono oryginalne nazwy cech.




**Rysunek 3.4.** Wartości korelacji Kendalla przy użyciu różnych podzbiorów zbioru danych (CWE-119, CWE-399, razem) dla badanych cech. Na osi x umieszczono oryginalne nazwy cech.

W celu zbudowania zbioru danych 10 najlepszych cech przy użyciu wyników korelacji, wykorzystano cechy o najwyższej wartości współczynnika korelacji Spearmana w zbiorze danych z uwzględnieniem obu typów podatności. W tym przypadku p-wartość jest większa niż poziom istotności tylko dla cechy *CCCCMVGCOM*. Ta cecha nie jest uwzględniona w zredukowanym zbiorze danych.

Cechy zostały uszeregowane przy użyciu zysku informacyjnego, współczynnika wzrostu informacji, spadku Giniego i techniki $\chi^2$, aby określić dziesięć najlepszych cech dla każdej z metod, które następnie zostały wykorzystane do wytrenowania trzynastu modeli uczenia maszynowego. Wybrano dziesięć najlepszych cech z całego zbioru danych, aby ujednolicić proces oceny. Wyniki tej analizy można zobaczyć na Rysunku 3.5. Cechy użyte w zredukowanych zestawach zostały wymienione w Tabeli 3.8. Można zauważyć, że siedem z dziesięciu cech jest takich samych dla wszystkich podzbiorów. Cechy te to: code_smells, open_issues, violations, major_violations, sqale_index,

**Tabela 3.7.** P-wartości uzyskane w analizie korelacji dla kolumn, w których co najmniej jedna p-wartość sugeruje, że hipoteza zerowa powinna zostać odrzucona.

| | minor_violations | CCCCLines_of_Code | comment_lines | duplicated_blocks | CCCCLOCCOM | CCCCMVGCOM | effort_to_reach _maintainability_rating_a |
|---|---|---|---|---|---|---|---|
| CWE-399 | | | | | | | |
| Pearson | ≪0.001 | ≪0.001 | ≪0.001 | **0.4415** | 0.0002 | 0.0231 | **0.0513** |
| Spearman | ≪0.001 | **0.0726** | **0.3540** | 0.0033 | **0.0723** | ≪0.001 | **0.2308** |
| Kendall | ≪0.001 | **0.0726** | **0.3539** | 0.00336 | **0.0723** | ≪0.001 | **0.2307** |
| CWE-119 | | | | | | | |
| Pearson | ≪0.001 | ≪0.001 | ≪0.001 | ≪0.001 | 0.00439 | 0.0088 | ≪0.001 |
| Spearman | **0.0694** | ≪0.001 | ≪0.001 | ≪0.001 | ≪0.001 | ≪0.001 | ≪0.001 |
| Kendall | **0.0694** | ≪0.001 | ≪0.001 | ≪0.001 | ≪0.001 | ≪0.001 | ≪0.001 |
| All | | | | | | | |
| Pearson | ≪0.001 | ≪0.001 | ≪0.001 | ≪0.001 | ≪0.001 | 0.0009 | 0.0024 |
| Spearman | ≪0.001 | ≪0.001 | 0.0011 | ≪0.001 | 0.0027 | **0.0581** | ≪0.001 |
| Kendall | ≪0.001 | ≪0.001 | 0.0011 | ≪0.001 | 0.0027 | **0.0581** | ≪0.001 |

comment_lines_density i critical_violations. Większość z nich jest powiązana z liczbą błędów znalezionych w kodzie przez SonarQube, co wydaje się być rozsądnym wyborem do zbudowania modelu predykcji podatności. Co istotne, żadna z par podzbiorów nie zawiera tych samych informacji.

Ponadto przeprowadzono analogiczną analizę na podzbiorach zbioru danych skoncentrowanych na podatnościach CWE-399 (Rysunek 3.6) i CWE-119 (Rysunek 3.7). Ze względu na większą liczbę próbek z podzbioru CWE-119, na ogólne 10 najlepszych cech według Information Gain dla zbioru danych podsumowujących składa się dziewięć z dziesięciu najlepszych cech z podzbioru CWE-119. Jedna cecha - „minor violations" pochodzi z innego podzbioru i uzyskała ona najwyższą wartość Information Gain dla podzbioru CWE-399. Siedem z tych 10 najlepszych wartości można znaleźć w grupie stosunkowo wysokich wartości Information Gain dla podzbioru CWE-399. W przypadku wskaźnika Gain Ratio, 8 z 10 najlepszych cech ogółem zostało znalezionych w 10 najlepszych cechach CWE-119, a 7 z 10 najlepszych cech ogółem zostało znalezionych w 10 najlepszych cechach CWE-399 i stosunkowo wysokich kolejnych wartościach. Podobnie w przypadku spadku Giniego, było to 9/10 cech dla podzbioru CWE-119 i 7/10 cech dla podzbioru CWE-399. Dla metryk $\chi^2$ współczynniki wynosiły 8/10 dla podzbioru CWE-119 i 9/10 cech dla podzbioru CWE-399. W przypadku CWE-399 wartości wszystkich wskaźników spadają wolniej niż w innych przypadkach, dlatego rozsądne jest rozważenie niektórych cech nieuwzględnionych w 10 najlepszych jako potencjalnych źródeł informacji dla modelu.

Zagregowano wyniki dokładności przy użyciu różnych podzbiorów cech wejściowych jako atrybutu grupującego i następujących metod uzyskania agregacji: wartość średnia, odchylenie standardowe, wartość maksymalna i wartość minimalna. Wyniki można zaobserwować na Rysunkach 3.8 („Accuracy"), 3.9 („Recall") i 3.10 („Specificity"). Rysunek 3.8 pokazuje, że średnie wyniki dokładności są porównywalne dla wszystkich podzbiorów cech wejściowych. Odchylenie standardowe jest również porównywalne dla wszyst-


**Rysunek 3.5.** Rangi cech uzyskane przy użyciu różnych typów metryk dla całego zbioru danych

kich badanych przypadków. Można zaobserwować znaczącą różnicę między maksymalnymi wynikami dla podzbioru uwzględniającego tylko podatności CWE-399, co sugeruje, że ten typ podatności jest łatwiejszy do wykrycia przy użyciu cech statycznych analizatorów kodu. W przypadku wartości minimalnych różnica ta jest mniej widoczna. Najwyższą wartość minimalną uzyskano dla podzbioru opartego na CWE-399 z uwzględnieniem wszystkich cech. Wyniki są znacznie bardziej zróżnicowane dla Recall i Specificity, przy czym bardziej dla Recall niż dla Specificity. Na Rysunkach 3.9 i 3.10 widać, że minimalne wyniki Specificity są w większości przypadków wyższe niż Recall. W prawie wszystkich przypadkach najwyższe średnie wyniki uzyskano dla pełnego zestawu cech. Taki zbiór danych jest największy i zawiera najbardziej zróżnicowane rodzaje informacji oraz dostarcza najbardziej ogólnych informacji, które mogą być wykorzystywane przez różne klasyfikatory.

Ponadto dokonano agregacji wyników, grupując je według typu modelu uczenia maszynowego, aby ocenić wydajność modeli wytrenowanych na różnych podzbiorach zbioru danych. Ponownie wykorzystano dokładność, czułość i specyficzność i ich cechy statystycznych (średnia, odchylenie standardowe, maksimum i minimum). Wyniki zaprezentowano na Rysunkach 3.11,

**Tabela 3.8.** 10 najlepszych cech uzyskanych dla różnych technik rankingowych selekcji cech.

| Ranga | Korelacja Spearmana | Information Gain | Gain Ratio | Gini Decrease | $\chi^2$ |
|---|---|---|---|---|---|
| 1 | code_smells | violations | blocker_violations | violations | critical_violations |
| 2 | open_issues | open_issues | minor_violations | open_issues | violations |
| 3 | violations | code_smells | violations | code_smells | open_issues |
| 4 | major_violations | major_violations | open_issues | major_violations | code_smells |
| 5 | sqale_index | minor_violations | code_smells | minor_violations | major_violations |
| 6 | comment_lines_density | sqale_index | major_violations | sqale_index | duplicated_lines_density |
| 7 | duplicated_lines_density | comment_lines_density | critical_violations | comment_lines_density | sqale_index |
| 8 | critical_violations | sqale_debt_ratio | sqale_index | sqale_debt_ratio | comment_lines_density |
| 9 | info_violations | duplicated_lines_density | comment_lines_density | duplicated_lines_density | duplicated_lines |
| 10 | statements | critical_violations | info_violations | critical_violations | uncovered_lines |

3.12 i 3.13.

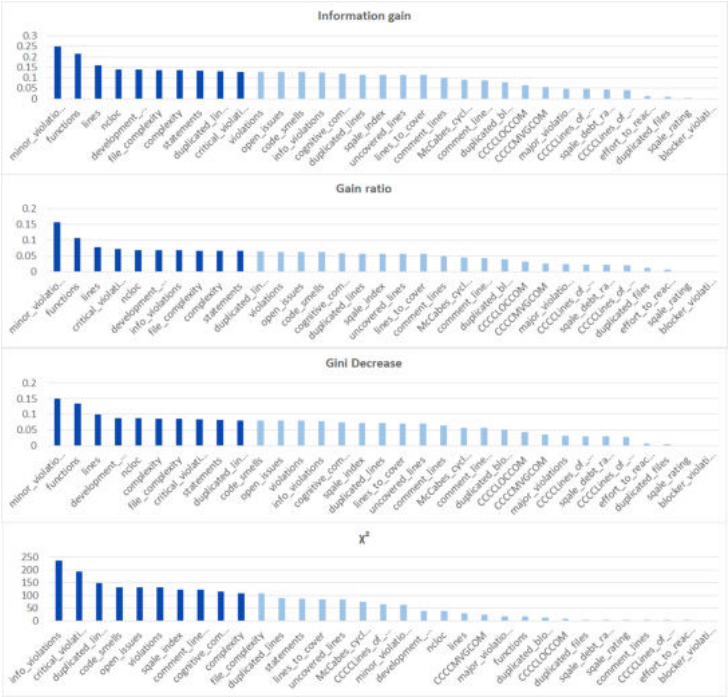
Na Rysunku 3.11 widać, że pod względem dokładności najlepszymi klasyfikatorami są algorytm KNN i Bagged Trees. Ich dokładność jest również dobra pod względem wartości maksymalnej i minimalnej, a także odchylenia standardowego, które wynosi 6% dla obu modeli. Modele, które uzyskały najgorsze wyniki to Regresja Logistyczna, Naiwny klasyfikator Bayesa, SVM (wyjątkiem był SVM z gaussowskim jądrem) i Subspace Discriminant.

Na Rysunku 3.12 wyraźnie widać, że wyniki Recall są znacznie bardziej zróżnicowane niż wyniki Accuracy. Chociaż maksymalne wartości klasyfikatorów przyjmują podobne wartości, wartości minimalne i średnie pokazują, że niektóre klasyfikatory są bardzo niestabilne pod względem Recall, a mianowicie naiwny klasyfikator Bayesa i maszyny wektorów nośnych z jądrami sześciennym i kwadratowym. W przypadku tych klasyfikatorów minimalna wartość Recall jest bliska 0%, co oznacza praktycznie brak przewidywań luk w zabezpieczeniach.

Pod względem specyficzności (Rysunek 3.13) wyniki są bardziej niezawodne niż czułości, jednak w przypadku SVM z jądrem sześciennym wartość odchylenia standardowego osiąga 31,18%, a wartość minimalna 0%. Również model maszyny wektorów nośnych z jądrem kwadratowym jest wysoce zawodny.

Modele zespołowe osiągają wiarygodne wyniki pod względem wszystkich miar i kategorii grupowania (statystyki). Wyjątkiem jest model Subspace Discriminant.

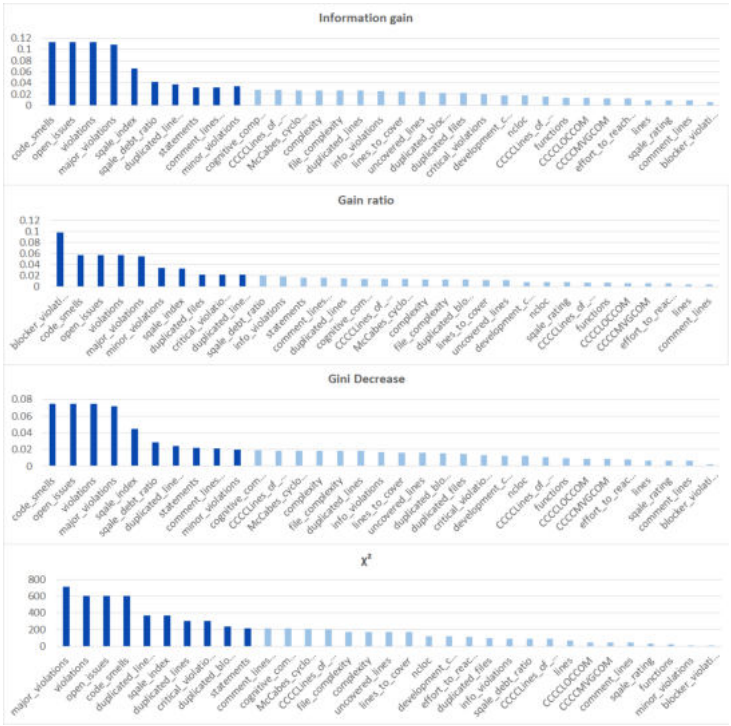
Białoskrzynkowe modele uczenia maszynowego kładą nacisk na interpretowalność. Celem jest stworzenie przejrzystego procesu predykcji. Pozwalają one wizualizować, jakie cechy zostały wykorzystane w procesie predykcji. Ta forma prezentacji wywiera pozytywny aspekt na wyjaśnialność algorytmów sztucznej inteligencji. Aby dostarczyć więcej informacji na temat cech odpowiednich do zadania przewidywania podatności, zdecydowano się przedstawić strukturę drzew decyzyjnych, które przewidują występowanie podatności na




**Rysunek 3.6.** Rangi cech uzyskane przy użyciu różnych typów metryk dla podzbioru danych CWE-399

podstawie wszystkich uzyskanych cech (33 cechy) i pozwalają im zdecydować, które cechy powinny być użyte w modelu średniej wielkości. W analizie dokładności wykorzystano większe modele drzew decyzyjnych, jednak ich rozmiar utrudnia interpretację, dlatego na potrzeby interpretacji wykorzystanych cech zdecydowano się na budowę mniejszych modeli, które w tym przypadku są bardziej praktyczne. Dokładność uzyskanych modeli wyniosła 75,9 % (86,7 % specyficzności, 62,6 % czułości) dla podzbioru CWE-119 i 85,3 % (81,6 % specyficzności, 89,8 % czułości) dla podzbioru CWE-399.

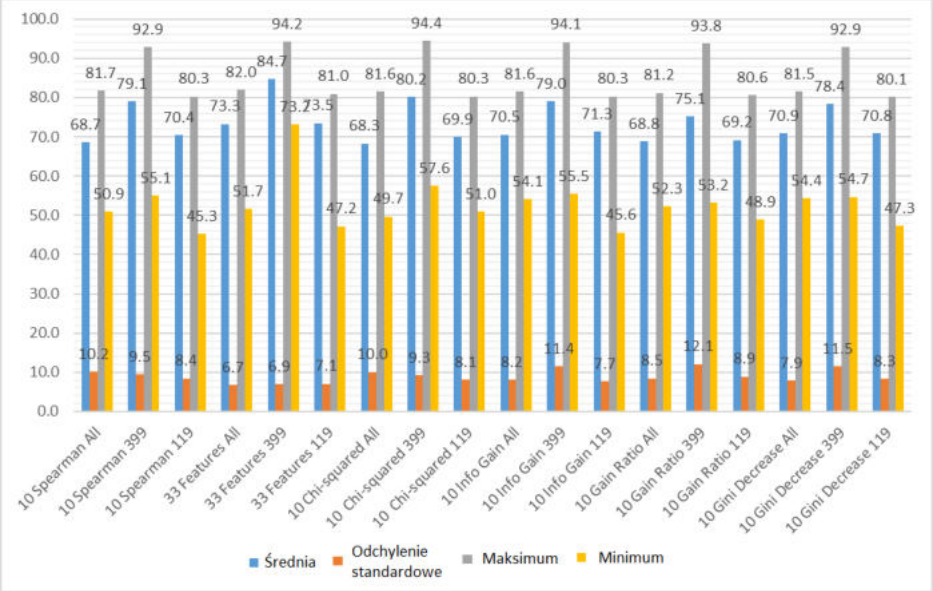
Na Rysunku 3.14 przedstawiono strukturę drzewa wytrenowanego na podzbiorze z uwzględnionymi podatnościami CWE-399. Cecha comment_lines_density została wykorzystana w korzeniu drzewa. Również parametr comment_lines znajduje się wysoko w hierarchii. Sugeruje to na przykład, że kod jest skomplikowany i wymaga wielu komentarzy, aby programiści mogli go zrozumieć. Sugeruje także o potencjalnych niedokończonych fragmentach kodu lub o fragmentach alarmujących o konieczności wprowadzenia poprawek. Również cechy z kategorii problemów (minor_issues i major_issues) są ważne dla predykcji. Określają one szereg potencjalnych problemów w kodzie - tutaj wykorzystywana jest moc reguł statycznego analizatora kodu. Wykorzystywane są również cechy związane z łatwością utrzymania: dług techniczny (ang.




**Rysunek 3.7.** Rangi cech uzyskane przy użyciu różnych typów metryk dla podzbioru danych CWE-119

Technical Debt) i zapachy kodu (ang. Code Smells). Niska łatwość utrzymania i duża złożoność kodu skutkuje podatnością na błędy (udowodnione w badaniu empirycznym - [195]).

Na Rysunku 3.15 przedstawiono strukturę drzewa, które zostało wytrenowane na podzbiorze z uwzględnionymi podatnościami CWE-119. W korzeniu drzewa można znaleźć jedną z cech utrzymywalności kodu (ang. Maintainability) - zapachy kodu. Tutaj można zauważyć, że problemy z utrzymywalnością i złożonością kodu mogą być sygnałem do rozważenia podatności kodu na błędy [195]. Inne cechy wykorzystane w tym modelu: functions, LOCCOM, complexity, cognitive_complexity. Sugerują one również, że rozmiar i złożoność kodu mogą być wskaźnikami podatności oprogramowania na błędy. Liczba funkcji (przy mniejszej liczbie funkcji prawdopodobieństwo podatności na błędy jest większe) może być oznaką, że duże funkcje są używane zamiast małych, a zasada pojedynczej odpowiedzialności może zostać złamana (ale jest to również silnie zależne od rozmiaru elementu kodu). Z drugiej strony, liczba zduplikowanych linii może wskazywać na złe praktyki związane z programowaniem.

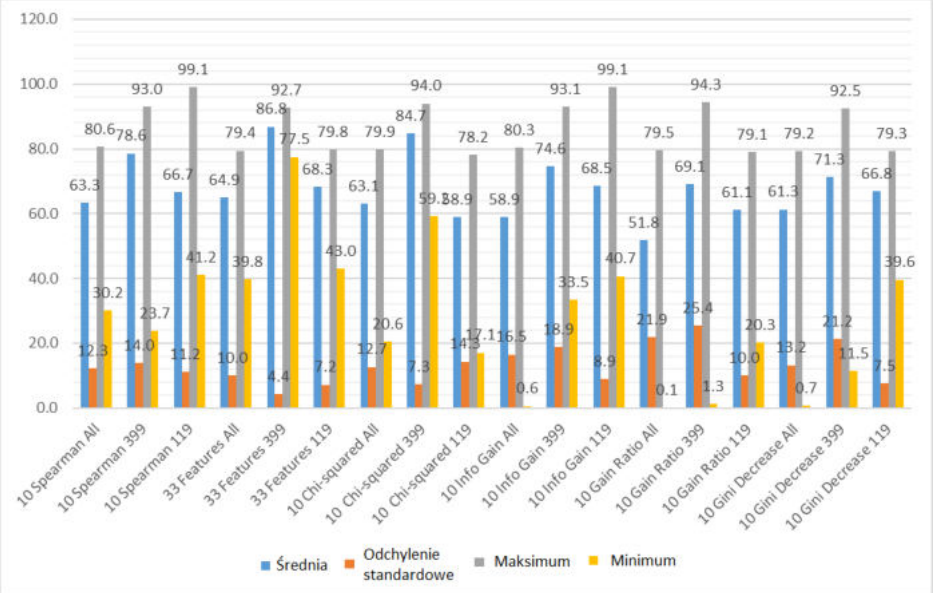



**Rysunek 3.8.** Zagregowane wyniki dokładności pogrupowane według podzbioru cech

### 3.2.3 Wnioski

Analizatory statyczne kodu są często wykorzystywane w firmach tworzących oprogramowanie, stąd uzasadnione jest wykorzystanie do oceny bezpieczeństwa oferowanych przez nie metryk. W pracy zbadano potencjalną przydatność tych metryk do znajdowania podatności typu CWE-399 oraz CWE-119 – podatności związanych z niepoprawnym zarządzaniem zasobami oraz nieodpowiednim ograniczeniem operacji w granicach bufora pamięci. Są to typowe problemy związane z oprogramowaniem napisanym w językach C i C++ ze względu na ich wysoką kontrolę nad zasobami. Języki te są wykorzystywane do tworzenia bibliotek numerycznych szeroko stosowanych w erze sztucznej inteligencji. Z tego powodu zdecydowano się przeanalizować możliwość wykrywania przykładowych podatności właśnie dla tych języków. Celem było również zainspirowanie przyszłych prac społeczności akademickiej i przemysłowej poprzez dostarczenie obszernej bazy wiedzy i dyskusji na temat wielu metod selekcji cech i ich przydatności do przewidywania podatności na zagrożenia przy wykorzystaniu różnych standardowych i złożonych modeli uczenia maszynowego.

Wyniki analizy korelacji wykazały, że korelacje między cechami zebranymi z analizatorów statycznych kodu są w większości przypadków istotne statystycznie. Fakt, że dla 3 z 5 cech z narzędzia CCCC hipoteza o istotności korelacji została odrzucona, sugeruje, że SonarQube oferuje lepsze cechy

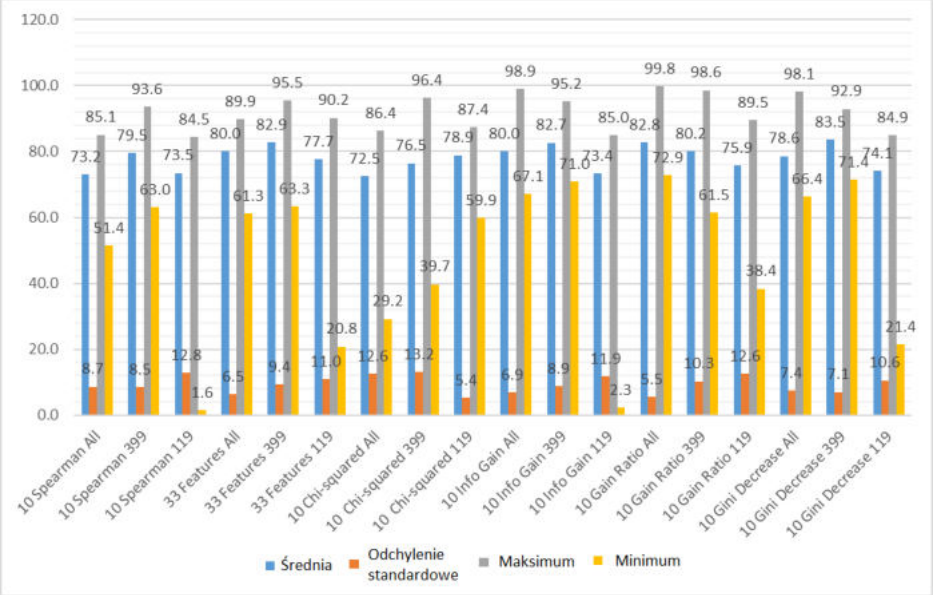



**Rysunek 3.9.** Zagregowane wyniki czułości pogrupowane według podzbioru cech

pod względem istotności statystycznej. Można to uzasadnić faktem, że SonarQube, w przeciwieństwie do CCCC, jest narzędziem szeroko stosowanym w praktyce do analizy statycznej, a także tym, że generuje on metryki związane z liczbą błędów znalezionych w kodzie i nie ogranicza się do tradycyjnych metryk oprogramowania.

Wszystkie podzbiory cech wyznaczone na potrzeby pracy umożliwiły poprawne wytrenowanie przynajmniej części z klasyfikatorów uczenia maszynowego, przy czym najlepsze modele osiągnęły dokładność nawet powyżej 90%. Zastosowanie selekcji cech w wielu przypadkach umożliwiło stworzenie modeli o lepszej dokładności (szczególnie w przypadku metod opartych na entropii) przy znacząco mniejszej liczbie cech wykorzystanych do zwrócenia wyniku. Zagregowane wyniki uzyskane dla różnych typów podzbiorów cech pokazują, że wyniki dokładności są porównywalne, a główna różnica jest widoczna dzięki miarom Recall i Specificity. Wczesne wskazanie podatności powinno charakteryzować się wyższą specyficznością i mniejszą liczbą fałszywych alarmów. Alternatywnie można stworzyć bardziej czuły model, który może skutkować większą liczbą alertów bezpieczeństwa, ale dla którego istnieje większa szansa na wykrycie prawdziwej podatności.

Korzystając z różnych metod selekcji cech (analiza korelacji, oparta na entropii, chi-kwadrat) i interpretacji modeli białoskrzynkowych uczenia maszynowego można określić cechy, które są najsilniejszymi wskaźnikami podatności na zagrożenia w przypadku podatności CWE-119 i CWE-399 dla



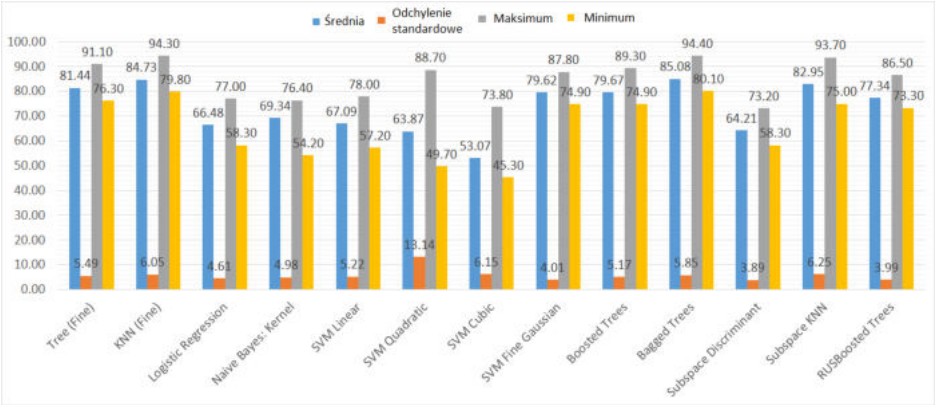

**Rysunek 3.10.** Zagregowane wyniki specyficzności pogrupowane według podzbioru cech

elementów kodu C/C++. Wyniki przeprowadzonej analizy pokazują, że cechy reprezentujące rozmiar, postać (gęstość komentarzy, liczba zduplikowanych linii) i złożoność kodu, a także metryki Maintainability (Code Smells i Technical Debt) mogą być wskaźnikami występowania podatności. Cechy te mogą wskazywać na złe praktyki programistyczne oraz fakt, że kod może być nadmiernie skomplikowany itp. Ponadto niska łatwość utrzymania kodu może być sygnałem do przeprowadzenia testów bezpieczeństwa.

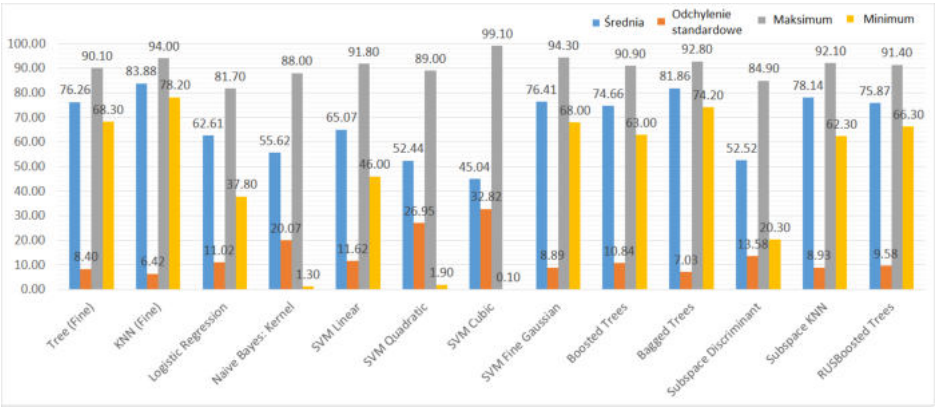
**Wyniki opisane w niniejszej sekcji przedstawiono w publikacji [196] opublikowanej w ramach projektu Horyzont2020 „SDK4ED: Software Development ToolKit for Energy Optimization and Technical Debt Elimination" [197].**

## 3.3 Wykorzystanie autorskiej modyfikacji Losowych Sieci Neuronowych do wykrywania zagrożeń bezpieczeństwa

W Sekcji 3.2 zaprezentowano, że konwencjonalne algorytmy uczenia maszynowego i cechy statycznych analizatorów kodu wykazują potencjał dla poprawy bezpieczeństwa oprogramowania. Niemniej jednak, mimo udokumen-

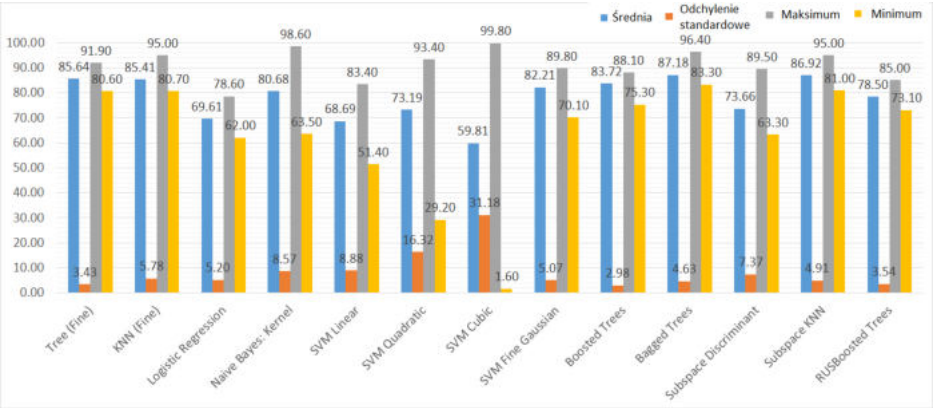



**Rysunek 3.11.** Statystyki dokładności (minimum, maksimum, średnia i odchylenie standardowe) uzyskane dla różnych klasyfikatorów.




**Rysunek 3.12.** Statystyki czułości (minimum, maksimum, średnia i odchylenie standardowe) uzyskane dla różnych klasyfikatorów.

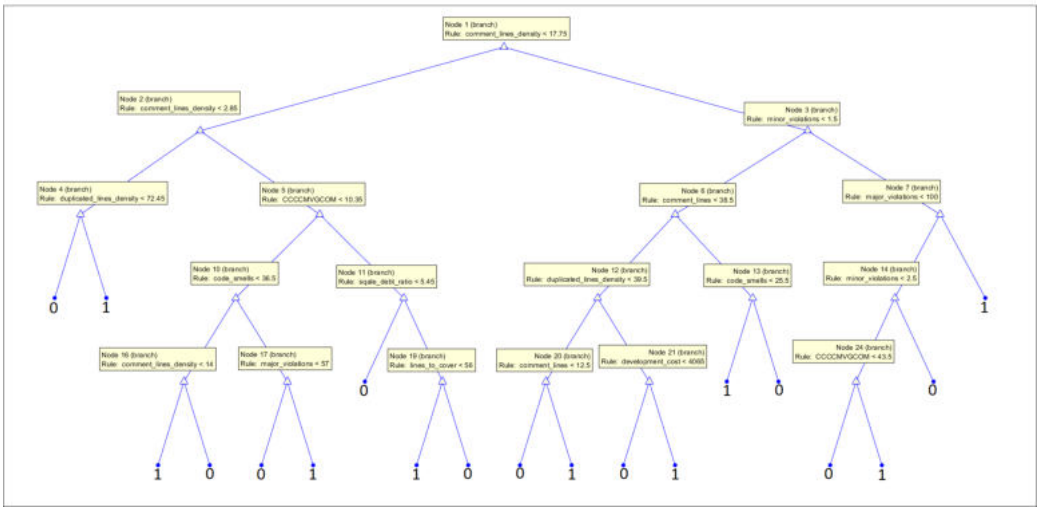
towanej skuteczności zbadanych metod, istotne jest kontynuowanie badań w kierunku poszukiwania innowacyjnych rozwiązań, które mogą oferować wysoką skuteczność w wykrywaniu zagrożeń poprzez otwarcie nowych perspektyw badawczych. Dlatego też w sekcji zaproponowano modyfikację Losowych Sieci Neuronowych (ang. Random Neural Networks, RNN) i zastosowano ją w dwóch zadaniach z zakresu bezpieczeństwa. Na przestrzeni lat ten rodzaj sieci był stosowany w różnorodnych zadaniach: między innymi w takich obszarach jak klasyfikacja pojazdów [42], rzeczywistość rozszerzona [198], a także w kontekście bezpieczeństwa sieciowego [45, 46, 199]. Z powodu tej uniwersalności, sieć wykazuje potencjał do wniesienia istotnej wartości także w kontekście bezpieczeństwa systemów informatycznych. Unikalną cechą tego typu sieci neuronowej jest fakt, że w przeciwieństwie do większości konwencjonal-




**Rysunek 3.13.** Statystyki specyficzności (minimum, maksimum, średnia i odchylenie standardowe) uzyskane dla różnych klasyfikatorów.s

nych sieci neuronowych, dla każdego połączenia między neuronami posiada ona dwa rodzaje wag - dodatnie, wzbudzające aktywację neuronu docelowego, oraz ujemne - pełniące rolę inhibitorów. Proponując własne rozwiązania, mamy możliwość tworzenia ich od podstaw z uwzględnieniem aspektu wyjaśnialności, który jest kluczowy w dziedzinie sztucznej inteligencji. Z tego powodu w pracy zaproponowano dwie modyfikacje Losowych Sieci Neuronowych. Pierwsza modyfikacja obejmuje sposób inicjalizacji wag, tak aby przed rozpoczęciem trenowania sieć była neutralna. Takie rozwiązanie ma pozytywny wpływ na wyjaśnialność działania sieci neuronowej oraz interpretację sposobu jej inicjalizacji. Drugim proponowanym rozwiązaniem jest specjalna restrykcja wag nakładana podczas trenowania. Restrykcja ta wiąże ze sobą wartości par wag (dodatniej i ujemnej) w celu ograniczenia liczby obciążających operacji matematycznych wykonywanych podczas trenowania sieci z wykorzystaniem metody najszybszego spadku. Na potrzeby pracy wyznaczono formuły aktualizacji wag i inicjalizacji dla różnych architektur RNN, co stanowi istotny wkład teoretyczny dla tego typu sieci. Proponowane rozwiązanie wykorzystano do stworzenia systemu wykrywania ataków sieciowych wykorzystujących infrastrukturę botnet oraz systemu wykrywania podatności oprogramowania.

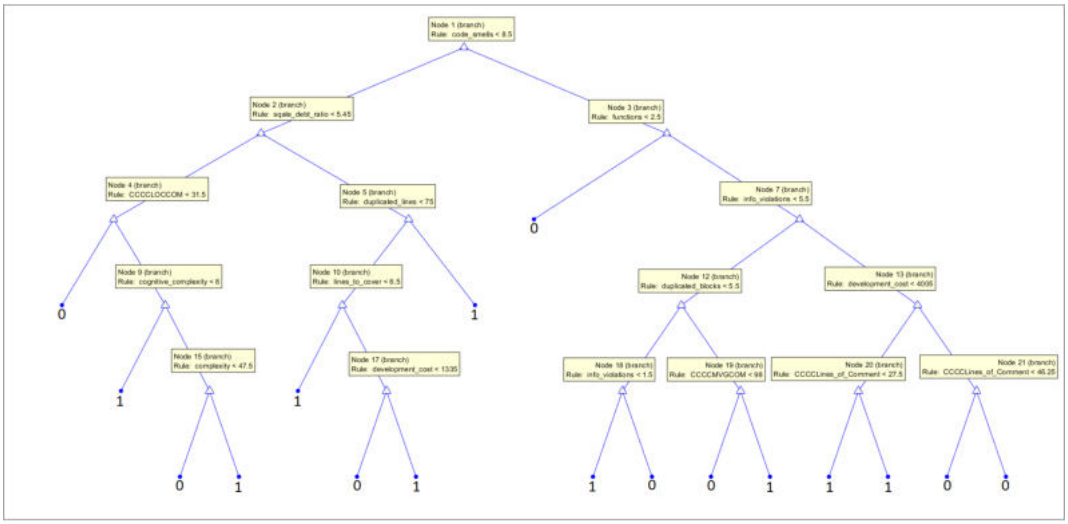
Pierwszym zastosowaniem proponowanej sieci jest wykrywanie ataków sieciowych. Jest to niezwykle istotny problem z perspektywy bezpieczeństwa systemów inteligentnych, ponieważ ataki sieciowe stają się coraz bardziej powszechne, zaawansowane i złośliwe. Jednocześnie rośnie zależność ludzi od urządzeń podłączonych do internetu niemal we wszystkich dziedzinach życia. Szczególnie jest to istotne w domenie rozwiązań inteligentnych, które często opierają swoje działanie na obliczeniach wykonywanych w chmurze



**Rysunek 3.14.** Średnie drzewo decyzyjne z uwzględnieniem wszystkich 33 cech jako danych wejściowych dla podzbioru z uwzględnionymi podatnościami CWE-399.

obliczeniowej, co wymaga częstego przesyłania informacji między różnymi komponentami systemu. Co więcej, gwałtowny wzrost liczby urządzeń IoT wykorzystujących algorytmy sztucznej inteligencji wprowadza dodatkowe zagrożenia dla bezpieczeństwa [200, 201]. Jest tak dlatego, że tego typu proste urządzenia są bardziej podatne na ataki niż te pełnowymiarowe. Może to prowadzić do kradzieży danych, wyczerpania zasobów energetycznych [202] oraz do przejęć prowadzących do ich wykorzystania w sieciach typu botnet do przeprowadzana ataków typu Distributed Denial of Service (DDoS) [75]. Ponieważ urządzenia te są często dostępne w Internecie i połączone do komunikacji machine-to-machine (M2M), urządzenia Internetu Rzeczy (IoT) są naturalnym „wejściem" dla atakujących. Wykorzystanie komunikacji bezprzewodowej dla urządzeń IoT zwiększa również ich podatność [8].

Urządzenia IoT często są wyposażone w mikrokontrolery zasilane bateryjnie stąd oszczędzanie energii jest ważne, a złożone techniki wykrywania ataków wymagające dużej ilości obliczeń nie mogą być na nich wykorzystywane [45]. Dlatego kluczowe jest zaproponowanie nowych efektywnych i precyzyjnych algorytmów do wykrywania ataków, które mogłyby działać na ograniczonych urządzeniach. Ważne jest również zbadanie metod spoza typowego zestawu algorytmów uczenia płytkiego i głębokiego. Stąd w pracy zaproponowano modyfikację Losowej Sieci Neuronowej (ang. Random Neural Network, RNN) do wykrywania ataków w sieciach IoT. Sieć może być trenowana offline, tworząc mały, ale skuteczny model, który daje zadowalające



**Rysunek 3.15.** Średnie drzewo decyzyjne z uwzględnieniem wszystkich 33 cech jako danych wejściowych dla podzbioru z uwzględnionymi podatnościami CWE-119.

rezultaty przy znacznie mniejszej liczbie operacji obliczeniowych niż wymaga jego standardowa wersja. Jest to możliwe dzięki wprowadzonym ograniczeniu wag do standardowego algorytmu uczenia RNN. Przedstawiono wyniki numeryczne pokazujące wyższość prezentowanych metod nad standardowymi procedurami wykorzystywanymi przez Losowe Sieci Neuronowe.

Drugim rozpatrywanym problemem w tej części pracy było wykorzystanie proponowanej modyfikacji Losowej Sieci Neuronowej do wykrywania podatności w kodzie. Przewidywanie podatności oprogramowania jest ważnym i aktywnym obszarem badań, w którym potrzebne są nowe metody tworzenia dokładnych i wydajnych narzędzi, które mogą identyfikować problemy związane z bezpieczeństwem. Bezpieczeństwo oprogramowania opiera się głównie na starannie napisanym kodzie źródłowym, a luki w oprogramowaniu są konsekwencją defektów, które są trudne do znalezienia [106]. Większość luk w zabezpieczeniach jest spowodowana typowymi błędami programistycznymi wprowadzanymi przez programistów na wczesnych etapach cyklu życia oprogramowania [113]. Programistom często brakuje wiedzy specjalistycznej w zakresie bezpieczeństwa, a ich zasoby testowe są ograniczone. Dlatego też statyczne analizatory kodu oparte na regułach i standardach bezpieczeństwa oprogramowania, np. SonarQube [179] i Veracode [178] są często używane w cyklu produkcyjnym oprogramowania. Popularnym podejściem w literaturze stało się również wykorzystywanie klasyfikatorów binarnych do priorytetyzacji testowania [130]. Tego typu podejście, oparte na bardziej konwencjo-

nalnych metodach uczenia maszynowego, zostało przedstawione w Sekcji 3.2. Jako że analiza kodu pod kątem bezpieczeństwa jest czasochłonna i kosztowna [128], konieczne jest tworzenie dokładnych i wydajnych systemów predykcji podatności opartych na nowych podejściach, między innymi na różnych typach cech wykorzystywanych w jednym systemie.

W ostatnich latach wyniki oparte na różnych typach cech sugerują, że jest to dobry kierunek rozwoju [124, 125], podczas gdy zastosowanie metod uczenia głębokiego do przewidywania podatności także rośnie [129, 130, 203]. Dlatego też, w niniejszej pracy stworzono hybrydowy model przewidywania podatności oprogramowania oparty na Losowych Sieciach Neuronowych i konwolucyjnych sieciach neuronowych, który wykorzystuje zarówno cechy tekstowe, jak i metryki generowane przez statyczny analizator kodu SonarQube (cechy te wykazały potencjał do wykrywania podatności oprogramowania w Sekcji 3.2). Celem pracy było zbadanie, czy takie połączenie cech i technik może być wykorzystywane do przewidywania podatności. Wykorzystanie cech hybrydowych może dostarczyć różnorodnych informacji opisujących analizowane oprogramowanie. Wykorzystanie modeli, które nie były wcześniej używane w tym zadaniu, może skutkować dobrą dokładnością. Przykładem jest Losowa Sieć Neuronowa i jej autorska modyfikacja. Szerokie wykorzystanie zarówno sieci RNN, jak i sieci konwolucyjnych i ich wysoka skuteczność rozwiązywania szerokiej gamy problemów były motywacją do wykorzystania ich w celu stworzenia systemu wykrywającego podatności w oprogramowaniu napisanym w językach C/C++. Sieci CNN są wykorzystywane do przetwarzania i redukcji wymiarowości danych tekstowych (ekstrakcji cech), a sieć RNN wykorzystano do złączenia wyekstrahowanych cech tekstowych z metrykami statycznego analizatora kodu.

### 3.3.1 Proponowane metody dla Losowych Sieci Neuronowych

Proponowane w niniejszej pracy metody obejmują nowy sposób inicjalizacji wag sieci oraz wprowadzenie specjalnej restrykcji wag mającej na celu uproszczenie procesu trenowania Losowych Sieci Neuronowych. Model Losowych Sieci Neuronowych został wprowadzony w pracy [40]. Charakterystyczną cechą tej sieci jest fakt, że interakcje między jej neuronami odbywają się poprzez wymianę pozytywnych i negatywnych sygnałów, które są definiowane przez model probabilistyczny. Właśnie ta cecha odróżnia je od większości modeli sztucznych sieci neuronowych [51]. Sygnały z zewnątrz przychodzą zgodnie z rozkładem Poissona, neurony emitują sygnały zgodnie z rozkładem wykładniczym, a ruchy sygnałów w sieci są markowowskie. Neuron $i$

przekazuje pozytywny sygnał do neuronu $j$ z prawdopodobieństwem $p^+_{i,j}$, negatywny sygnał z prawdopodobieństwem $p^-_{i,j}$ i emituje sygnał opuszczający sieć z prawdopodobieństwem $d_i$. Ponieważ wartości te reprezentują prawdopodobieństwa, zachodzi zależność $\sum_j [p^+_{i,j} + p^-_{i,j}] + d_i = 1$. Pozytywne i negatywne sygnały wejściowe z zewnątrz przychodzą z szybkościami $\Lambda_i$ i $\lambda_i$ odpowiednio. $r_i$ oznacza szybkość wysłania sygnału przez $i$-ty neuron.

Szybkość sygnału docierającego od neuronu $i$ do neuronu $j$ można wyznaczyć za pośrednictwem wzoru:

$$w^*_{i,j} = r_i p^*_{i,j} \tag{3.8}$$

Gwiazdkę w powyższym równaniu należy zastąpić minusem/plusem dla sygnałów hamujących/pobudzających. Wykorzystując terminologię sztucznych sieci neuronowych są to wagi sieci opisujące siłę połączeń między neuronami. Wagi w modelu RNN różnią się jednak od wag występujących w innych standardowych modelach (połączeniowych) - opisują one szybkości, z jakimi sygnały są wymieniane między neuronami. Z tego powodu przyjmują one tylko wartości nieujemne. Wykorzystując tak zdefiniowane elementy, wyrażenie $r_i$ przyjmuje formę $r_i = \sum_j (w^+_{i,j} + w^-_{i,j})$.

Zdefiniujmy wielkość $q_i$:

$$q_i = \lambda^+_i / [r_i + \lambda^-_i], \tag{3.9}$$

gdzie $\lambda^+_i, \lambda^-_i$ spełniają następujący system równań nieliniowych:

$$\begin{aligned} \lambda^+_i &= \Lambda_i + \sum_j q_j w^+_{j,i} \\ \lambda^-_i &= \lambda_i + \sum_j q_j w^-_{j,i} \end{aligned} \tag{3.10}$$

Jeśli $\lambda^+_i \geqslant r_i + \lambda^-_i$, zakłada się, że $q_i = 1$ i $i$-ty neuron jest nazywany nasyconym. Wielkość $q_i$ można interpretować jako prawdopodobieństwo nasycenia $i$-tego neuronu [51], lub stosując standardową terminologię sztucznych sieci neuronowych - aktywację neuronu. Zarówno modele „do przodu" (ang. feed-forward), jak i rekurencyjne, gwarantują nieujemne i unikalne rozwiązanie dla $\lambda^+_i, \lambda^-_i$. Jeśli dla wszystkich neuronów $q_i < 1$, sieć jest stabilna (momenty marginalne i łączne można obliczyć za pomocą rozwiązania w postaci produktu) [50].

Najczęściej używanym algorytmem do trenowania Losowych Sieci Neuronowych jest algorytm oparty na metodzie najszybszego spadku, który został wprowadzony dla tej sieci neuronowej w pracy [50].

Pary wejścia-wyjścia sieci neuronowej oznaczane są jako $(\iota, Y)$. Zbiór $K$ wejść i wyjść oznaczane są przez $\iota = \iota^{(1)}, ..., \iota^{(K)}$ i $Y = y^{(1)}, ..., y^{(K)}$ odpowiednio.

Pozytywne i negatywne $k$-te sygnały wejściowe są oznaczone odpowiednio przez $\Lambda^{(k)}$ i $\lambda^{(k)}$. $\Lambda^{(k)}$ i $\lambda^{(k)}$ są wektorami $n$ prędkości przepływu sygnałów wpływających do każdego neuronu:

$$\Lambda^{(k)} = [\Lambda_1^{(k)}, ..., \Lambda_n^{(k)}] \quad \lambda^{(k)} = [\lambda_1^{(k)}, ..., \lambda_n^{(k)}] \tag{3.11}$$

Pożądane wektory wartości wyjściowych są oznaczane jako $y^{(k)} = [y_1^{(k)}, ..., y_n^{(k)}]$. Głównym celem modelu jest przybliżenie tych wartości w sposób minimalizujący funkcję straty zdefiniowaną następująco:

$$E_k = \frac{1}{2} \sum_{i=1}^{n} a_i (q_i^{(k)} - y_i^{(k)})^2, \tag{3.12}$$

gdzie $a_i$ jest stałą oraz $a_i \in \langle 0, 1 \rangle$.

Podczas procesu trenowania iteruje się wartości wejściowe i aktualizuje wagi na podstawie odpowiedzi sieci neuronowej oraz wyznaczonych wartości funkcji straty. W celu aktualizacji wag wykorzystuje się metodę najszybszego spadku. W tym procesie szukane są tylko pozytywne wartości wag. Niech $n$ oznacza liczbę neuronów w sieci. Reguła aktualizacji może być napisana w następujący sposób:

$$w_{u,v}^{*(k)} = w_{u,v}^{*(k-1)} - \eta \sum_{i=1}^{n} a_i (q_i^{(k)} - y_i^{(k)}) [\frac{\delta q_i}{\delta w_{u,v}^*}]^{(k)}, \tag{3.13}$$

gdzie $\eta$ nazywany współczynnikiem uczenia określa w jakim stopniu zmieniane są wagi w danym kroku uczenia. Wyrażenie dla $\frac{\delta q_i}{\delta w_{u,v}^*}$ oraz metoda obliczania tej wartości zostały przedstawione w [50].

**Interpretowalna metoda inicjalizacji wag** Standardowym podejściem jest inicjalizacja wag sieci neuronowej za pomocą losowo generowanych wag, ale można je wybierać również za pomocą innych metod, które optymalizują jakieś kryterium [51]. W pracy zaprezentowano nową metodę inicjalizacji wag. Wagi sieci dobiera się w taki sposób, aby przed rozpoczęciem uczenia prawdopodobieństwo pobudzenia każdego neuronu było dane przez wartość „neutralną" $q_i = 0.5$ dla $i = 1, ..., N$, kiedy wejście do każdego neuronu jest również ustawione na wartość neutralną. W szczególności wartość neutralna wejścia jest wybrana jako $\lambda_i = 0$, $\Lambda_i = \Lambda^o > 0$ i $w_{i,j}^+ = w_{i,j}^- = w$, dla

$i, j = 1, ..., N$, tak, że:

$$q_i = \frac{\Lambda^o + wQ}{2Nw + wQ},$$
$$gdzie\ Q = \sum_{i=1}^{N} q_i = Nq,$$
$$gdy\ q_i = q = 0.5,\ otrzymujemy\ w = \frac{4\Lambda^o}{3N}.$$

Zaletą metody jest jej intuicyjność. Stworzenie tej strategii było możliwe dzięki prostej w interpretacji aktywacji neuronów RNN, które są wyrażone jako prawdopodobieństwo pobudzenia neuronu. W tym przypadku neutralną wartością rozpatrywanego prawdopodobieństwa jest $\frac{1}{2}$. Inicjalizacja ustawiająca wartości aktywacji wszystkich neuronów w ten sposób sprawia, że sieć nie jest stronnicza w kierunku żadnej z opcji na początku treningu. Skutkuje to również stałymi warunkami początkowymi treningu w przeciwieństwie do inicjalizacji losowej, która zazwyczaj wymaga wielokrotnych uruchomień, aby wybrać najlepszy model (uzyskany przy korzystnym zestawie losowo wybranych początkowych parametrów). Poniżej wyprowadzono faktyczne wzory dla różnych architektur Losowej Sieci Neuronowej.

.

Ogólny model sieci RNN zakłada architekturę **rekurencyjną**. Dlatego, biorąc pod uwagę warunek $w^+_{i,j} = w^-_{i,j} = w$ oraz $q_i = 0.5$, można zainicjować wagi dla takiej sieci w następujący sposób:

$$0.5 = \frac{\lambda + 0.5Nw}{\lambda + 2.5Nw}, \quad lub \quad \lambda = 1.5Nw. \tag{3.14}$$

Aby więc uzyskać prawdopodobieństwo pobudzenia $q_i = 0.5$, można ustawić $w$ na dowolną wartość, pod warunkiem, że $\lambda = 1.5Nw$.

W przypadku **sieci „do przodu" (ang. feed-forward**), bez utraty ogólności można numerować węzły tak, że $w^+_{i,j} = w^-_{i,j} = 0$ jeśli $j \leqslant i$. W tym przypadku, ustawiając wszystkie niezerowe wagi na identyczną wartość $w$ zachodzi $\sum_{j=1}^{N} w^+_{j,i} = \sum_{j=1}^{N} w^-_{j,i} = (i-1)w$, gdy $r_i = 2w(N-i)$. Jeśli prawdopodobieństwo pobudzenia wszystkich neuronów ma być ustawione na $q_i = 0.5$, można zapisać:

$$0.5 = \frac{\lambda_i + 0.5(i-1)w}{\lambda_i + 2w(N-i) + 0.5w(i-1)}, \tag{3.15}$$

lub $\lambda_i = w(2N - 2.5i + 0.5)$.

W przypadku **wielowarstwowej sieci RNN**, kiedy ma ona $L$ warstw, oznaczmy warstwę 1 jako warstwę wejściową, a warstwę $L$ jako warstwę wyjściową. Oznaczymy przez $K_l$ liczbę neuronów w warstwie $l$, a przez $l, i$ indeks

$i-tego$ neuronu w warstwie $l$ przy czym $1 \leqslant i \leqslant K_l$. Zakłada się, że każdy neuron w warstwie $l$ wysyła sygnały tylko do neuronów w warstwie $l+1$, dla $1 \leqslant l \leqslant L$. Neurony w warstwie 1 są jedynymi, które odbierają zewnętrzne sygnały wejściowe, podczas gdy neurony w warstwie $L$ nie mają połączeń z innymi neuronami, ale odbierają połączenia od neuronów w warstwie $L-1$.

Ponownie rozważmy inicjalizację sieci przed zastosowaniem algorytmu uczenia, tak aby wszystkie neurony zaczynały z początkową wartością aktywacji $q_{l,i} = 0.5$ , gdy wejścia pierwszej warstwy są identyczne $\Lambda_{1,i} = \lambda_{1,i} = \lambda$.

Ponieważ sieć ma wielowarstwową strukturę typu feed-forward, wszystkie wagi można zainicjalizować w następujący sposób:

$$\begin{aligned} w^{+}_{l,i;k,j} = w^{-}_{l,i;k,j} = 0,\ & gdy\ k \neq\ l+1, \\ & dla\ 1 \leqslant l \leqslant L-1, \\ w^{+}_{l,i;l+1,j} = w^{-}_{l,i;l+1,j} = w_{l,i} > 0, & \\ d_{l,i} = 0,\ 1 \leqslant l \leqslant L-1,\ & r_{l,i} = 2w_l K_{l+1}, \end{aligned} \tag{3.16}$$

Tak więc dla $l = 1$ parametry można zainicjować w następujący sposób:

$$\begin{aligned} 0.5 &= \frac{\Lambda_{1,i}}{\lambda_{1,i} + 2w_{1,i}K_2},\ lub \\ w_{1,i} = w_1 &= \frac{\lambda - 0.5\lambda}{K_2} = \frac{0.5\lambda}{K_2} \end{aligned} \tag{3.17}$$

Ponieważ dla warstw $l = 2, ..., L-1$ wejścia zewnętrzne są równe zero, tj. $\Lambda_{l,i} = \lambda_{l,i} = 0$, początkowe parametry dla neuronu warstwy $l$ można zainicjalizować w następujący sposób:

$$\begin{aligned} 0.5 &= \frac{0.5w_{l-1}K_{l-1}}{2w_l K_{l+1} + 0.5w_{l-1}K_{l-1}},\ lub \\ w_l &= 0.25w_{l-1}\frac{K_{l-1}}{K_{l+1}}. \end{aligned} \tag{3.18}$$

W końcu dla neuronów warstwy wyjściowej można zapisać:

$$\begin{aligned} 0.5 &= \frac{0.5w_{L-1}K_{L-1}}{r_L + 0.5w_{L-1}K_{L-1}},\ lub \\ r_L &= 0.5w_{L-1}K_{L-1}, \end{aligned} \tag{3.19}$$

**Uproszczona metoda trenowania sieci** Obliczanie wartości pochodnych jest jedną z najbardziej czasochłonnych operacji w treningu Losowej Sieci Neuronowej z wykorzystaniem metody najszybszego spadku. W klasycznym podejściu oblicza się te wartości osobno dla dodatnich i ujemnych

wag. W pracy zaprezentowano nowe podejście wykorzystujące specjalną restrykcję wag, które redukuje liczbę tych kosztownych operacji.

Pierwszym krokiem jest dodanie warunku:

$$w^+_{i,j} + w^-_{i,j} = W_{i,j}, \tag{3.20}$$

gdzie $W_{i,j}$ jest stałą dla wszystkich $(i, j)$. W rezultacie należy wprowadzić zmiany do wyrażeń dla pochodnych cząstkowych.

Biorąc pod uwagę taką zmianę, składnik $r_i$ jest teraz stałą, a wyrażenie dla $q_i$ przyjmuje następującą formę:

$$q_i = \frac{\Lambda_i + \sum_j q_j w^+_{j,i}}{\lambda_i + \sum_j q_j (W_{j,i} - w^+_{j,i}) + r_i} \tag{3.21}$$

Oznaczmy przez $D_i$ mianownik $q_i$. Z wyrażenia 3.21 wyprowadzono następujące wyrażenie dla pochodnej cząstkowej $\frac{\delta q_i}{\delta w^+_{u,v}}$:

$$\begin{aligned} \frac{\delta q_i}{\delta w^+_{u,v}} = &\sum_j \frac{\frac{\delta q_j}{\delta w^+_{u,v}} [w^+_{j,i} - (W_{j,i} - w^+_{j,i}) q_i]}{D_i} \\ &+ \mathbf{1}[v = i](\frac{q_u}{D_i} + \frac{q_u q_i}{D_i}) \end{aligned} \tag{3.22}$$

Analogiczne równanie można otrzymać dla $\frac{\delta q_i}{\delta w^-_{u,v}}$. Aby ograniczyć liczbę operacji wymagających dużego nakładu obliczeniowego, wystarczy obliczyć tylko wartości jednego typu pochodnych i wykorzystać zależność:

$$\frac{\partial q_i}{\partial w^-_{u,v}} = -\frac{\partial q_i}{\partial w^+_{u,v}} \tag{3.23}$$

Główny wkład pracy to opracowane wzory umożliwiające inicjalizację wag sieci o różnorodnej architekturze oraz uproszczenie jej procesu trenowania za pośrednictwem metody najszybszego spadku poprzez wprowadzenie specjalnego ograniczenia na wagi sieci. Niemniej jednak na potrzeby pracy przeprowadzono również eksperymenty mające na celu sprawdzenie wpływu proponowanych metod na działanie Losowej Sieci Neuronowej w zadaniu wykrywania ataków sieciowych wykorzystujących infrastrukturę botnet oraz sprawdzenie możliwości wykorzystania tego typu sieci do wykrywania podatności oprogramowania.

### 3.3.2 Wykrywanie ataków sieciowych opartych na infrastrukturze botnet z wykorzystaniem Losowych Sieci Neuronowych

Ponieważ domena IoT została zdominowana przez rozwiązania inteligentne, takie jak inteligentne domy, w pracy skupiono się na takich właśnie środowiskach i ich bezpieczeństwie. Chociaż w ostatnich latach stworzono różne zbiory danych z ruchem IoT, między innymi zbiory wprowadzone w pracach [75, 204, 205, 206], zazwyczaj do ich stworzenia wykorzystano jedynie bardzo ograniczony zakres urządzeń IoT - głównie urządzenia multimedialne. Co więcej, część prac w domenie IoT nadal wykorzystuje do ewaluacji zbiory danych z rodziny DARPA [207, 208], KDDCup'99 [209]. Te zbiory danych zostały stworzone ponad 20 lat temu, kiedy to de facto nie istniał Internet Rzeczy, a rozwiązania inteligentne nie były stosowane na szeroką skalę. Taksonomia ataków, a także sam ruch sieciowy zmieniły się znacząco od tego czasu, a dane zebrane w tych zbiorach nie stanowią adekwatnej reprezentacji ataków. Modele wytrenowane i zwalidowane na takich zbiorach danych nie mają zatem zastosowania we współczesnym szybko zmieniającym się świecie [210]. W pracy zdecydowano się wykorzystać zbiór danych BoT-IoT, ponieważ zawiera on wiele heterogenicznych urządzeń typowych dla inteligentnych domów, takich jak inteligentne oświetlenie, termostat, stacja monitorowania pogody, inteligentne drzwi garażowe i lodówkę. Składa się on z rzeczywistego i symulowanego ruchu IoT. Pomimo faktu, że zbiór danych jest jednym z najnowszych tego typu zbiorów, to był już wykorzystywany w wielu różnych pracach, np. [15, 79, 211, 212, 213].

W pracy wykorzystano "wersję z 10 najlepszymi cechami" zestawu danych Bot-IoT [189] do przeprowadzenia testów z proponowanymi metodami dla Losowych Sieci Neuronowych. Aby wygenerować te cechy, autorzy [189] wykorzystali dane zebrane w plikach pcap i zastosowali techniki współczynnika korelacji oraz entropii, aby wybrać najlepsze cechy uczące. Wykorzystany w pracy zbiór danych składa się z 1177 próbek. Zbiór danych testowych obejmuje 589 próbek, w tym 350 próbek reprezentujących ataki i 239 reprezentujących ruch neutralny.

Losowa Sieć Neuronowa zastosowana w eksperymentach składa się z $N = 12$ neuronów, 10 z nich odbiera sygnały zewnętrzne, a jeden neuron wyjściowy zwraca predykcję binarną. Przeprowadzono wstępne eksperymenty również z minimalnym rozmiarem sieci z jedenastoma neuronami, a także większych, ale najlepsze wyniki uzyskano przy użyciu dwunastu neuronów. Gdy liczba neuronów była mniejsza niż 12, wyniki były gorsze, a dla większych topologii wyniki się nie poprawiły. Aktualizacje wag były przeprowadzane po ustale-

niu wyjścia dla pojedynczej próbki danych wejściowych, w przeciwieństwie do uczenia wsadowego (ang. batch learning). Porównano wyniki uzyskane dla modyfikacji RNN z podstawowym modelem RNN. W porównaniu uwzględniono następujące warianty:

- z losową inicjalizacją i standardowym procesem trenowania
- z losową inicjalizacją i uproszczonym procesem trenowania (proponowana restrykcja wag)
- z neutralną inicjalizacją i standardowym procesem trenowania
- z neutralną inicjalizacją i uproszczonym procesem trenowania (proponowana restrykcja wag)

Wyniki zaprezentowano za pomocą ogólnej dokładności, czułości, precyzji i macierzy pomyłek.

### 3.3.3 Wykrywanie podatności w kodzie z wykorzystaniem Losowych Sieci Neuronowych

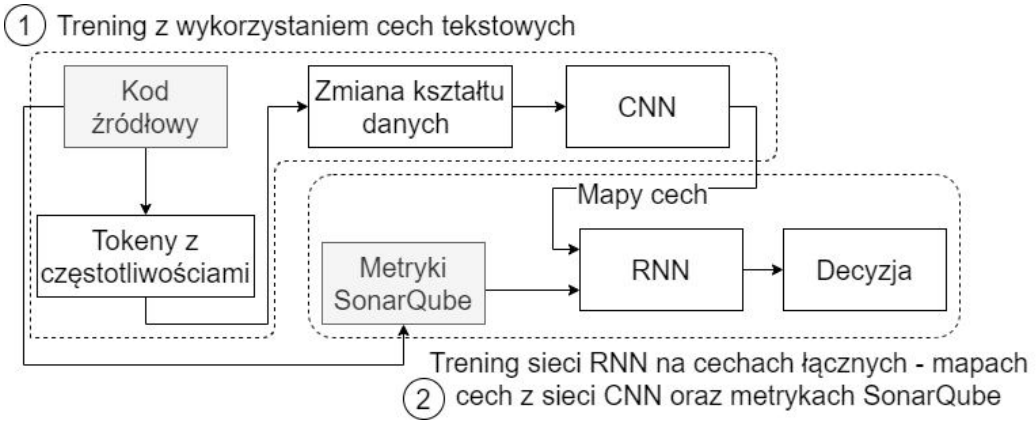
W prezentowanym rozwiązaniu zaproponowano system, który łączy cechy tekstowe z metrykami uzyskanymi ze statycznego analizatora kodu SonarQube. Do przewidywania podatności badanych komponentów oprogramowania (w postaci binarnych etykiet) wykorzystano dwa rodzaje sztucznych sieci neuronowych, a mianowicie losowe sieci neuronowe i konwolucyjne sieci neuronowe.

Część systemu odpowiedzialna za predykcję składa się z dwóch części: pierwsza z nich służy do redukcji wymiarowości i ekstrakcji informacji z cech tekstowych, a druga jest modelem wiążącym dla ekstrahowanych map cech i metryk wygenerowanych z narzędzia do analizy statycznej. Pierwsza część została zbudowana w oparciu o małą sieć neuronową CNN. Druga część to wydajna obliczeniowo modyfikacja sieci neuronowej RNN. Przepływ pracy systemu przedstawiono na Rysunku 3.16.

Mapy cech, które są pobierane z części konwolucyjnej, są jednowymiarowe i zawierają 10 cech (cechy te można postrzegać jako skompresowaną reprezentację danych wejściowych). Aby potencjalnie poprawić wyniki, wykorzystano dodatkowe cechy wygenerowane za pomocą narzędzia SonarQube [179]. Spośród wszystkich metryk wygenerowanych przez wybrany SAT wybrano pięć najlepszych, stosując technikę rankingu $\chi^2$.

Dodatkowo w pracy przeprowadzono analizę cech za pomocą metody wizualizacji $t-$*distributed* Stochastic Neighbour Embedding (t-SNE), która

jest nieliniowym algorytmem używanym do zmniejszania wymiarowości danych [214]. W metodzie tej wielowymiarowe dane są najczęściej osadzane na płaszczyźnie euklidesowej. W przypadku pracy, metoda ta służy do badania, czy istnieje lokalna struktura w wielowymiarowych danych [214].




**Rysunek 3.16.** Zarys systemu przewidywania luk w oprogramowaniu stworzony w oparciu zarówno o dane tekstowe, jak i metryki generowane z narzędzia do analizy statycznej kodu. Sieci neuronowe wykorzystane do stworzenia systemu to sieci konwolucyjne i losowe sieci neuronowe.

W eksperymentach wykorzystano zbiór danych wprowadzony w pracy [16]. W tym zbiorze danych uwzględniono dwa rodzaje luk w zabezpieczeniach: CWE-119 - podatność na błędy bufora oraz CWE-399 - podatność na błędy zarządzania zasobami, przy czym w zestawieniu uwzględniono wyłącznie podatność CWE-399. Aby uzyskać cechy eksploracji tekstu, zastosowano metodę opisaną w pracy [121]. Podejście to opiera się na założeniu, że język programowania można traktować jako język naturalny, dzięki czemu do reprezentowania zawartych w nim informacji można zastosować popularne techniki eksploracji tekstu. Korzystając z techniki „Bag-Of-Words", elementy mogą być reprezentowane jako tokeny z odpowiadającymi im licznościami występowania w tekście. Aby uzyskać pożądane metryki, wykorzystano natomiast statyczny analizator kodu SonarQube [179]. Narzędzie to pozwala na wyodrębnienie szerokiej gamy metryk. Liczba instancji w klasach i całym zbiorze danych została przedstawiona w Tabeli 3.9.

W eksperymentach wykorzystano standardowy podział danych: 70 % danych włączono do zbioru treningowego, a 30 % do zbioru testowego. Aby wyeliminować niezrównoważenie danych, dokonano redukcji klasy większościowej (tj. klasy niepodatnej) dla zbioru szkoleniowego i pozostawiono oryginalną liczbę próbek w zbiorze testowym. Jako cechy nietekstowe zastosowano pięć najlepszych cech spośród wszystkich wskaźników wygenerowanych

**Tabela 3.9.** Liczba elementów w zbiorze testowym wraz z podziałem na elementy podatne i neutralne.

| **Kategoria** | **Liczba próbek** | |
|---|---|---|
| Neutralne | 815 | 1499 |
| Podatne | 684 | |

przez narzędzie do statycznej analizy kodu przy użyciu techniki rankingu $\chi^2$. Zdecydowano się również zaprezentować macierze pomyłek (dla zbioru treningowego i testowego), ponieważ pozwalają one na bardziej szczegółowe przedstawienie wyników i wyrażają czułość i specyficzność modeli (zdolność do przewidywania, że klasa jest odpowiednio podatnym i neutralnym elementem). Jest to szczególnie istotne w przypadku zbioru testowego, który nie jest zrównoważony. Wykonano również analizę cech za pomocą algorytmu t-SNE i zaprezentowano wyniki wizualne. Zaprezentowano wyniki algorytmu t-SNE dla różnych podzbiorów cech - cech tekstowych, metryk i łącznie.

### 3.3.4 Wyniki

W sekcji opisano wyniki uzyskane za pośrednictwem proponowanych modyfikacji Losowej Sieci Neuronowej dla dwóch zadań związanych z bezpieczeństwem: wykrywaniem ataków sieciowych ataków sieciowych wykorzystujących infrastrukturę botnet oraz wykrywaniem podatności oprogramowania.

#### Wykrywanie ataków sieciowych opartych na infrastrukturze botnet z wykorzystaniem Losowych Sieci Neuronowych

W Tabeli 3.10 zaprezentowano wyniki dokładności uzyskane dla czterech testowanych wersji Losowej Sieci Neuronowej: losowej inicjalizacji, losowej inicjalizacji + ograniczenia wag, neutralnej inicjalizacji oraz neutralnej inicjalizacji + ograniczenia wag. Wyniki pokazują, że wprowadzenie neutralnej inicjalizacji pozwala uzyskać znacznie wyższą dokładność (96% w porównaniu do 83% dla losowej inicjalizacji na danych testowych). Dla zbioru testowego uzyskano takie same wyniki dla złączenia dwóch proponowanych metod oraz wariantu z neutralną inicjalizacją. Pokazuje to, że neutralna inicjalizacja wag pozwala na uzyskanie lepszej dokładności, podczas gdy ograniczenie wag nie pogarsza tej dokładności, mimo ograniczenia liczby kosztownych operacji. Dodanie proponowanego ograniczenia wag do losowej inicjalizacji zwiększa średnią dokładność, ale powoduje także większe odchylenie standardowe (prawie 3 razy większy) - pokazuje to, że dla niekorzystnych inicjalizacji wag zostały uzyskane gorsze wyniki niż w przypadku standardowym. Dlatego

najlepszym podejściem jest wykorzystanie proponowanego ograniczenia wag wraz z neutralną inicjalizacją.

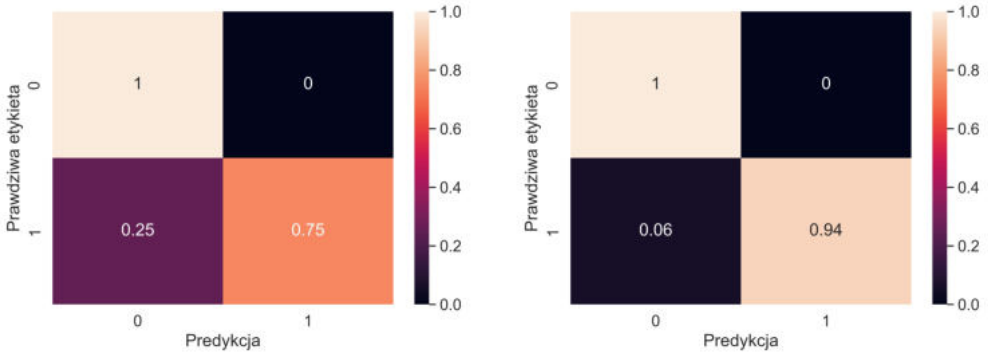
**Tabela 3.10.** Porównanie dokładności, precyzji i czułości dla wszystkich testowanych wersji Losowej Sieci Neuronowej. Dla losowej inicjalizacji przeprowadzono 20 eksperymentów i podano wyniki średnie oraz odchylenie standardowe. Wyrażenia Losowa oraz Neutralna w nagłówku tabeli odnoszą się do sposobu inicjalizacji. RW oznacza proponowaną restrykcję wag.

| **Losowa** | | **Losowa + RW** | | **Neutralna** | | **Neutralna + RW** | |
|---|---|---|---|---|---|---|---|
| **Trening** | **Test** | **Trening** | **Test** | **Trening** | **Test** | **Trening** | **Test** |
| **Dokładność (ang. Accuracy)** | | | | | | | |
| 86.41 ± 5 | 86.75 ± 4.4 | 88.46 ± 12 | 88.37 ± 12.1 | 96.90 | 96.09 | 96.80 | 96.09 |
| **Precyzja (ang. Precision)** | | | | | | | |
| 99.64 ± 0.5751 | 99.75 ± 0.4426 | 96.19 ± 3.472 | 96.4 ± 3.208 | 99.79 | 100.0 | 99.48 | 100.0 |
| **Czułość (ang. Recall)** | | | | | | | |
| 77.47 ± 8.574 | 77.91 ± 7.608 | 83.91 ± 19.82 | 83.47 ± 19.91 | 94.80 | 94.00 | 95.33 | 94.00 |

Dodatkowo zaprezentowano macierze pomyłek dla najgorszego i najlepszego przypadku (dla modelu z losową inicjalizacją bez ograniczenia wag, który uzyskał 83,71% dokładności na zestawie testowym, oraz neutralnej inicjalizacji z ograniczeniem wag) na Rysunku 3.17. Widoczne jest, że specyficzność obu modeli jest podobna i zysk w dokładności wynika z poprawy czułości (opisującej umiejętność wykrywania próbek ataku - anomalii), co jest pożądanym wynikiem.

### Wykrywanie podatności w kodzie z wykorzystaniem Losowych Sieci Neuronowych

Pierwszym krokiem analizy było zbadanie wygenerowanego zbioru danych za pomocą algorytmu t-SNE. Wyniki przedstawiono na Rysunku 3.18. Wyraźnie widać, że algorytm jest w stanie znaleźć lokalną strukturę w wysokowymiarowych danych zarówno w mapach cech wygenerowanych przy użyciu CNN, jak i metryk oprogramowania (w tym przypadku wszystkie metryki zostały wzięte pod uwagę). Chociaż trudniej jest podzielić cechy wygenerowane w narzędziu do analizy statycznej, istnieje tam lokalna struktura, która może sugerować, że można wydobyć dodatkowe informacje. Potencjalnie może to również mieć pozytywny wpływ na generalizację systemu, co może być problemem w przypadku danych tekstowych, które często mogą wprowadzać




**(a)** Losowa inicjalizacja, brak restrykcji wag

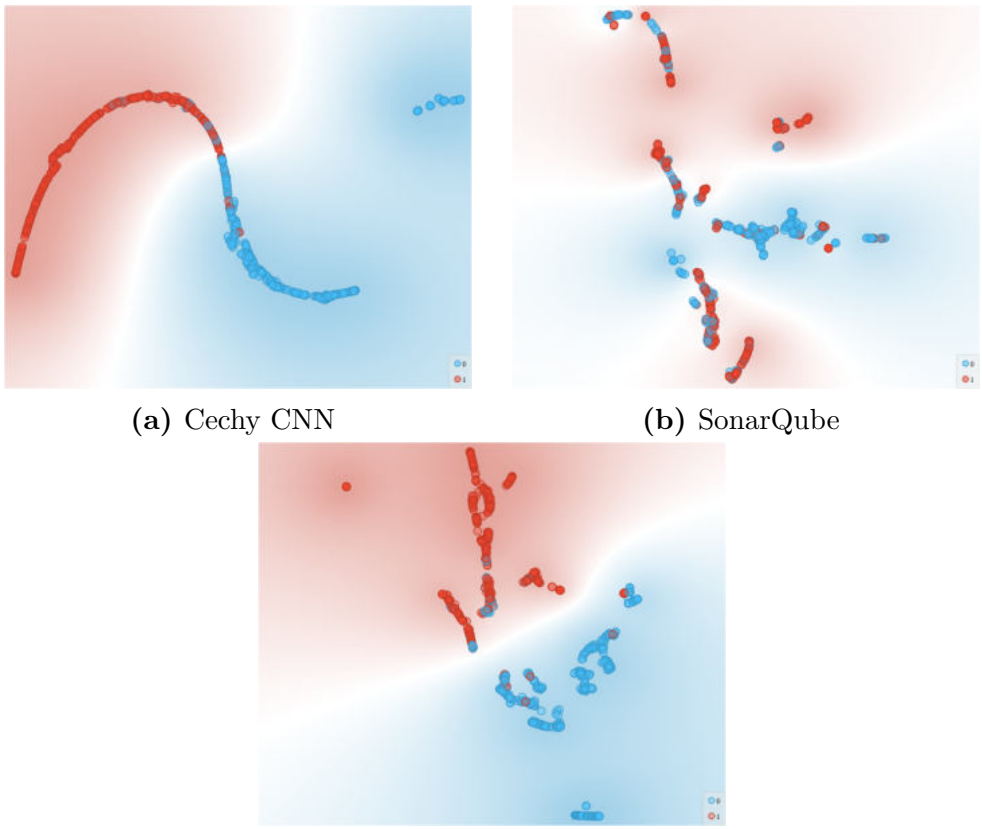
**(b)** Neutralna inicjalizacja + restrykcja wag

**Rysunek 3.17.** Macierze pomyłek uzyskane dla podstawowego wariantu Losowych Sieci Neuronowych oraz dla proponowanego wariantu z neutralną inicjalizacją wag oraz restrykcją wag nałożoną w procesie trenowania.

pewną stronniczość ze względu na charakterystyczne dla danego zbioru danych słownictwo. Z tego powodu zdecydowano się dodać te cechy do ogólnej analizy. Korzystając z obu zestawów cech, można uzyskać niemal liniową separację podatnych i niepodatnych instancji.

Najlepszy model stworzony podczas eksperymentów z eksploracją tekstu (w systemie uwzględniono jedynie cechy tekstowe i model CNN z klasycznym klasyfikatorem opartym na warstwie ściśle połączonej) i model hybrydowy osiągnęły 97 % dokładności treningowej, podczas gdy osiągnęły odpowiednio 93 % i 94 % dokładności testowej, dla modelu CNN opartego na eksploracji tekstu i modelu hybrydowego wykorzystującego RNN. Wyniki dla modelu czysto tekstowego i modelu hybrydowego przedstawiono w postaci znormalizowanych macierzy pomyłek na Rysunku 3.19.

Chociaż model hybrydowy osiąga o 1 % niższą czułość dla zestawu treningowego, wartość ta jest taka sama dla zestawu testowego zarówno dla modelu tekstowego, jak i modelu hybrydowego. Korzystając z modelu, który wykorzystuje zarówno mapy cech ekstrahowane za pośrednictwem sieci CNN, jak i metryki uzyskane ze statycznego analizatora kodu SonarQube, udało się obniżyć współczynnik wyników fałszywie pozytywnych zarówno dla danych treningowych, jak i testowych o 1 %. Można zauważyć, że model dobrze generalizuje i pomimo faktu, że dane w zbiorze testowym są rozłożone inaczej niż w zbiorze treningowym (ponieważ do zbalansowania instancji podatnych i niepodatnych na ataki w zbiorze treningowym wykorzystano próbkowanie w dół), osiąga on zadowalające wyniki pod względem czułości (ang. recall) i



**(a)** Cechy CNN **(b)** SonarQube

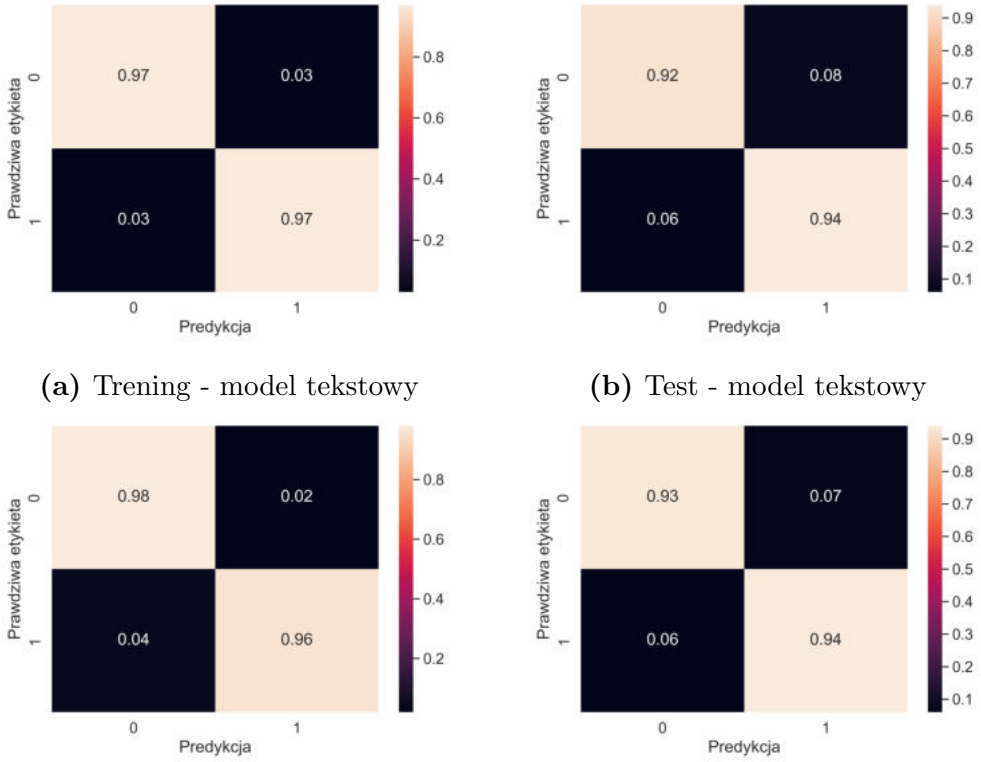
**(c)** Cechy CNN + SonarQube

**Rysunek 3.18.** Wizualizacja danych za pośrednictwem algorytmu t-SNE dla elementów podatnych i neutralnych na podstawie różnych typów cech.

specyficzności (ang. specificity).

### 3.3.5 Wnioski

W tej części pracy zaprezentowano dwie metody dedykowane Losowym Sieciom Neuronowym: nową metodę inicjalizacji oraz modyfikację procesu trenowania związaną z restrykcją wag. Na podstawie problemu detekcji ataków sieciowych wykorzystujących infrastrukturę botnet, zbadano wpływ zastosowania interpretowalnej metody inicjalizacji wag, która może być wykorzystywana do stworzenia niestronniczej (przed procesem trenowania) sieci. Zaprezentowano i przetestowano również specjalne ograniczenie wag, które pozwala znacznie zredukować liczbę operacji obliczeniowych (sprawia, że konieczne jest obliczanie jedynie połowy pochodnych w procesie optymalizacji metodą najszybszego spadku). Eksperymentalne wyniki pokazują, że wpro-




**(a)** Trening - model tekstowy **(b)** Test - model tekstowy

**(c)** Trening - model hybrydowy **(d)** Test - model hybrydowy

**Rysunek 3.19.** Znormalizowane macierze pomyłek prezentujące najlepsze wyniki osiągnięte przez model czysto tekstowy i model hybrydowy.

wadzenie proponowanej strategii inicjalizacji pozwala na uzyskanie lepszych wyników niż przy standardowej inicjalizacji. Wprowadzenie proponowanego ograniczenia wag nie ma negatywnego wpływu na dokładność tak zainicjalizowanej sieci, mimo znacznego zmniejszenia zużycia zasobów. Co więcej, ograniczenie wag stosowane do losowo zainicjalizowanej sieci może również nieznacznie zwiększyć średnie wyniki dokładności. Wstępne wyniki sugerują, że Losowe Sieci Neuronowe z neutralną inicjalizacją i uproszczoną procedurą trenowania wykazują potencjał do detekcji ataków botnetowych i poprawy bezpieczeństwa IoT i dają lepsze rezultaty w tym obszarze niże metoda bazowa. Inicjalizacja sieci w proponowany sposób ma również pozytywny wpływ na wyjaśnialność sztucznej inteligencji, ponieważ jest to naturalny i intuicyjny sposób inicjalizacji tego rodzaju sieci, tak aby była neutralna przed procesem trenowania.

Dodatkowo przeprowadzono także badania dotyczące innego aspektu poprawy bezpieczeństwa, a mianowicie wykrywania podatności w kodzie C/C++.

W pracy przedstawiono koncept systemu wykorzystującego dwa typy sieci neuronowych: losowe sieci neuronowe i sieci konwolucyjne oraz łączone cechy (tekstowe i ze statycznego analizatora kodu). Celem było stworzenie systemu przewidywania podatności oprogramowania na zagrożenia typu CWE-399. Była to pierwsza praca stosująca losowe sieci neuronowe do tego celu. Zaprezentowano także wstępne wyniki analizy dotyczącej struktury lokalnej danych dwóch typów cech kodu źródłowego: cech tekstowych oraz cech wygenerowanych za pośrednictwem analizatora kodu źródłowego. Analiza przeprowadzona przy użyciu algorytmu t-SNE wykazała, że wysokowymiarowe cechy używane do trenowania sieci neuronowych zawierają lokalną strukturę, którą można wykorzystać do przewidywania podatności oprogramowania. Chociaż ta lokalna struktura uzyskana przy użyciu cech z wybranego statycznego analizatora kodu nie jest tak wyraźna, jak cechy uzyskane z eksploracji tekstu, to rozsądne jest założenie, że informacje zawarte w tych cechach mogą pomóc w ulepszeniu podejść opartych na eksploracji tekstu. Wydaje się również, że największym wyzwaniem w wielu technikach przewidywania podatności oprogramowania i narzędziach analizy statycznej jest znaczna liczba fałszywych alarmów. Przedstawiony eksperyment pokazuje, że wykorzystując cechy tekstowe, a także metryki generowane przez statyczny analizator kodu, można zmniejszyć liczbę fałszywych alarmów, a co za tym idzie uzyskać poprawę w stosunku do modelu konwencjonalnego (sieć konwolucyjna i cechy wyłącznie tekstowe). Chociaż poprawa uzyskana za pośrednictwem łączonego modelu i cech nie jest wysoka to należy wziąć pod uwagę, że już sam bazowy model osiąga bardzo wysoką dokładność. Poprawa ta sugeruje natomiast, że dalsze prace łączące różne rodzaje cech opisujących kod źródłowy są jak najbardziej uzasadnione i jest to dobry kierunek rozwoju dziedziny przewidywania podatności oprogramowania. W przyszłych pracach z tego zakresu planowane jest wykorzystanie sieci typu Transformer [30] do analizy danych tekstowych w celu wykrywania podatności oprogramowania. Sieci te przyczyniły się do przełomu w dziedzinie przetwarzania języka naturalnego i zaczynają być stosowane także do przetwarzania kodu źródłowego [215, 216].

**Ogólna idea opisanych w niniejszej sekcji modyfikacji RNN została przedstawiona w pracy [217]. W pracy [217] zaprezentowano wyniki dotyczące wykrywania ataków za pośrednictwem tej modyfikacji, a w pracy [218] - wyniki związane z wykrywaniem podatności oprogramowania. Prace te były związane z trzema projektami w ramach europejskiego programu Horizon2020: „SDK4ED: Software Development ToolKit for Energy Optimization and Technical Debt Elimination", „SerIoT – Secure and Safe Internet of Things" oraz „IoTAC – Security By Design IoT Development and Certificate Framework with Front-end Access Control".**

## 3.4 Wykorzystanie fizycznych markerów do automatycznego tworzenia zbiorów danych oraz testowania sieci głębokich

Algorytmy głębokiego uczenia są kluczowym elementem obecnej rewolucji cyfrowej [22]. Nic więc dziwnego, że znalazły one również zastosowanie w obszarze robotyki i pojazdów sterowanych automatycznie (ang. Automated Guided Vehicles, AGV). Pojazdy AGV i innego typu roboty są coraz częściej wykorzystywane w zakładach produkcyjnych, gdzie dużą rolę przywiązuje się do automatyzacji procesów. Sieci neuronowe są tam stosowane np. do budowania systemów percepcji, w zadaniach takich jak rozpoznawanie obiektów, ich detekcja oraz segmentacja semantyczna [22]. Zakłady przemysłowe to środowiska, w których krytycznym aspektem jest bezpieczeństwo, ponieważ ludzie i roboty często tam współpracują. Z tego powodu niezawodność systemów automatycznych i autonomicznych jest wymogiem w takich zastosowaniach.

Chociaż modele głębokiego uczenia oferują duże możliwości, ich efektywne trenowanie i testowanie jest problematyczne. Algorytmy te wymagają dużych ilości danych, aby najpierw zostały odpowiednio wytrenowane, a następnie przetestowane (zarówno pod kątem dokładności, jak i bezpieczeństwa). Pozyskiwanie danych jest zazwyczaj bardzo kosztowne i wymaga dużego nakładu pracy manualnej. Dlatego też często stosuje się symulatory do generacji dużych ilości danych [22]. Niestety, pomimo postępów w wysoce realistycznych symulacjach, symulowany świat jest zawsze uproszczoną wersją rzeczywistych środowisk. Gromadzenie danych bezpośrednio na rzeczywistych platformach, takich jak AGV lub urządzenia produkcyjne, może dostarczyć danych kontekstowych. Takie dane odzwierciedlają rzeczywiste warunki pracy tych systemów i scenariusze, które mogą wystąpić. Ponadto gromadzenie danych bezpośrednio na platformach docelowych danej klasy, zapewnia, że rozdzielczość, oświetlenie i zniekształcenia danych obrazu są zgodne z tym, co system percepcji może napotkać podczas rzeczywistej pracy. Jest to ważne, ponieważ zmiana rozkładu w danych wizyjnych może spowodować znaczny spadek dokładności algorytmów percepcji [150]. Dlatego też spójność między danymi szkoleniowymi/testowymi/operacyjnymi ma kluczowe znaczenie dla niezawodnej percepcji w rzeczywistych warunkach.

Wykorzystanie danych zebranych w świecie rzeczywistym do treningu i testowania głębokich sieci neuronowych może zapewnić dokładniejsze i bardziej niezawodne algorytmy percepcji, które są w stanie efektywnie generalizować w rzeczywistych sytuacjach. Testowanie z wykorzystaniem takich danych może również pomóc zidentyfikować potencjalne słabe punkty systemów. Aby

ograniczyć czas i wysiłek niezbędny do etykietowania danych wykorzystywanych przez pojazdy AGV, korzystne byłoby stworzenie automatycznych systemów etykietowania dla różnych algorytmów percepcji. Ponieważ takie dane mają kluczowe znaczenie dla testowania, w niniejszej pracy zaproponowano strategię automatycznego gromadzenia danych i etykietowania ich w rzeczywistych środowiskach bezpośrednio na platformach docelowych. Cały system może być wykorzystywany zarówno do testowania online, jak i offline sieci do rozpoznawania obiektów.

Dodatkowym aspektem do rozważenia w przypadku rzeczywistych środowisk przemysłowych jest jak najmniejsza ingerencja w środowisko oraz efektywność budżetowa. Trudnością środowisk przemysłowych jest fakt, że często występują w nich obiekty dużej i średniej skali. W przeciwieństwie do obecnych w literaturze rozwiązań [156, 157], które do automatycznego oznaczania obiektów w świecie rzeczywistym stosują dedykowane platformy i stojaki, w przypadku dużych obiektów nie jest to możliwe. Z tego powodu w pracy zaproponowano wykorzystać pojedyncze znaczniki (ang. fiducial markers) do oznaczania obiektów w świecie rzeczywistym. Wybrano jeden z najpopularniejszych systemów znaczników - ArUco [153]. Znaczniki te oferują dokładne i szybkie wykrywanie oraz mogą być wydrukowane za pośrednictwem domowych drukarek. Dzięki wykorzystaniu znaczników do oznaczania obiektów w rzeczywistym środowisku, przedstawiona procedura testowa może być wykorzystana do testowania pojazdów AGV podczas rzeczywistej jazdy z perspektywy rzeczywistego robota. Identyfikacja słabych punktów systemów inteligentnych jest aspektem kluczowym z perspektywy bezpieczeństwa tych systemów. Jako że testowanie jest jednym z głównych sposobów identyfikacji tych punktów, w pracy zdecydowano się skupić także na tym aspekcie związanym z bezpieczeństwem systemów opartych na sztucznej inteligencji.

### 3.4.1 Proponowany system do testowania

Proponowany system pozwala na testowanie modeli rozpoznawania obiektów opartych na głębokim uczeniu w czasie rzeczywistym bezpośrednio na pojazdach AGV. W celu automatycznego odczytywania etykiet obiektów podczas robienia zdjęcia zastosowano znaczniki ArUco. Znaczniki te umieszcza się na fizycznych obiektach, które mają być rozpoznawane przez dany system wizyjny. Znaczniki dostarczają informacji na temat lokalizacji obiektu na danym zdjęciu oraz o jego przynależności do danej klasy obiektów. Aby umożliwić komunikację między różnymi komponentami systemu (czujnikami wizyjnymi, narzędziami ArUco, przetwarzaniem obrazu i modelami uczenia maszynowego), wykorzystano Robotic Operating System (ROS) działający przy użyciu języka Python. ROS to zestaw bibliotek programistycznych umożliwia-

jących tworzenie efektywnych rozwiązań dla zastosowań w robotyce. Oferuje on między innymi sterowniki, rozwiązania komunikacyjne i wiele zaimplementowanych metod z dziedziny robotyki. Aby uzyskać działanie w czasie rzeczywistym, system zaimplementowano jako aplikację wielowątkową. Poniżej opisano działanie kluczowych wątków:

- Pozyskiwanie danych z kamery RGB. Wstępnie przetworzone obrazy są później wykorzystywane jako wejście dla algorytmów rozpoznawania obiektów w czasie rzeczywistym, a obrazy bez przekształceń ukierowanych na wykorzystywany model są również wykorzystywane do tworzenia zbioru danych do testów offline (i potencjalnie trenowania nowych modeli).

- Wykrywanie i identyfikacja znaczników ArUco - są one używane do lokalizowania obiektów w klatce obrazu i oznaczania tych obiektów.

- Wstępne przetwarzanie obrazu - usuwanie znaczników, kadrowanie, skalowanie wartości pikseli do odpowiedniego zakresu dla modelu. Predykcja z wykorzystaniem modelu.

Na Listingu 1 znajduje się pseudokod, który pokazuje działanie systemu. System umożliwia testowanie w czasie rzeczywistym (przedstawia użytkownikowi predykcje modelu w formie tekstu zrozumiałego dla człowieka), ale także zapisuje obrazy wykorzystane do testowania modelu w celu dalszych eksperymentów z różnymi modelami w trybie offline. Obrazy używane do predykcji są zapisywane. Identyfikatory ArUco i przewidywania modelu są zapisywane w pliku csv wraz z nazwą konkretnego obrazu. Aby ocenić dokładność modelu, etykiety ArUco można łatwo zmapować na etykiety używane przez sieć neuronową przy użyciu stworzonego słownika.

Obrazy wynikowe (które są wykorzystywane do predykcji w czasie rzeczywistym i mogą być następnie wykorzystane do testów offline) są przygotowywane w taki sposób, aby były odpowiednie dla sieci do rozpoznawania obrazów. Po wykryciu markera AruCo system odczytuje jego identyfikator (do celów etykietowania) i kontynuuje wstępne przetwarzanie obrazu. Najpierw marker AruCo jest ukrywany na obrazie - w tym celu zastosowana została prosta strategia. Odczytywane są wartości kolorów RGB w pobliżu narożników markera. Na tym etapie wartość narożnika zostaje przesunięta na zewnątrz markera - aby uzyskać kolor, który z założenia ma być neutralny w danej scenie. W zależności od umiejscowienia markera, będzie on uwzględniał kolor obiektu i/lub kolor tła. Dzięki tej procedurze w obrazach nie pojawia się wzorzec (marker ArUco), który jednoznacznie identyfikuje różne obiekty. Dzięki temu wynikowe obrazy mogą być również wykorzystywane do uczenia głębokich sieci neuronowych. Następnie do zlokalizowania

**Algorithm 1:** Testowanie sieci neuronowych w czasie rzeczywistym i etykietowanie obrazów za pomocą znaczników AruCo w celu stworzenia realistycznego zbioru danych.

**Input:** Model; Strumień Wideo
**Output:** Obrazy zapisane do folderu; plik csv z anotacjami AruCo i z sieci neuronowej

```
while Procedura testowania do
    Czytaj klatkę;
    Wykryj znacznik AruCo;
    if Znacznik wykryty then
        if Brak przetwarzanego wykrycia then
            Identyfikuj znacznik AruCo;
            Usuń znacznik z obrazu;
            Przytnij obraz;
            Przetwórz wstępnie obraz;
            Dokonaj predykcji za pośrednictwem modelu;
            Zbierz anotowane dane i wyniki testów;
            Zaprezentuj predykcję użytkownikowi;
            Zapisz anotowane dane i wyniki testów;
        end
    end
end
```

obiektu stosowana jest względna pozycja markera względem kamery. Do tego celu wykorzystywany jest środek markera i przeskalowana maksymalna odległość między krawędziami znacznika w osi pionowej i poziomej (przeskalowana o współczynnik liczbowy - 8 - w celu przycięcia obrazu). Taki obraz, po zastosowaniu odpowiedniej funkcji do wstępnego przetworzenia danych, może być wykorzystany jako dane wejściowe modeli głębokiego uczenia wykorzystywanych w eksperymentach (zarówno online jak i offline). W przypadku testowania online skalowanie odbywa się w wątku odpowiedzialnym za przetwarzanie obrazu i predykcję modelu, a w przypadku szkolenia offline należy to zrobić przed podaniem danych do modelu. System zapisuje obrazy, które są przycinane, a znacznik ArUco jest zakrywany z wykorzystaniem opisanej wcześniej metody. Do stworzenia samego systemu zbierania danych i testowania wykorzystano elementy szeroko stosowane w obszarze robotyki: Robotic Operating System (wersja Noetic) oraz OpenCV dla języka Python (wraz z modułem oferującym obsługę systemu znaczników ArUco).

### 3.4.2 Metodologia badań

W obszarze uczenia głębokiego, konwolucyjne sieci neuronowe (CNN) stały się standardowym szkieletem wizji komputerowej dla zadań, takich jak wykrywanie obiektów. Wynika to z zastosowania mechanizmu „przesuwnego okna", wbudowanej tendencyjności indukcyjnej (ang. inductive bias) i ekwiwariancji translacyjnej (ang. translational equivariance) [26, 39]. Z tego powodu w pracy wykorzystano właśnie sieci konwolucyjne jako przykładowe algorytmy uczenia głębokiego do rozpoznawania obiektów. Wybrane sieci konwolucyjne są zróżnicowane pod względem rozmiaru oraz zastosowanych technik. Poniżej wymieniono i krótko opisano zastosowane sieci:

1. **MobileNet** [219] - przykład wydajnego modelu dedykowanego dla mobilnych i wbudowanych aplikacji wizyjnych. Wykorzystuje on konwolucję separowalną głębokościowo (ang, depthwise separable convolution) w celu zwiększenia wydajności.

2. **MobileNetV2** [220] - zmodyfikowany model MobileNet, który osiąga lepsze wyniki dokładności niż jego poprzednik. Model ten wykorzystuje odwróconą strukturę rezydualną (ang. inverted residual structure). W tym modelu wejście i wyjście bloków rezydualnych są warstwami wąskiego gardła. Opiera się również na lekkich konwolucjach głębokościowych.

3. **DenseNet121** [221] - gęsta sieć konwolucyjna, w której każda warstwa jest połączona z każdą inną warstwą na zasadzie „feed-forward". Mapy

cech wszystkich poprzednich warstw są używane jako dane wejściowe danej warstwy, a jej dane wyjściowe są używane jako dane wejściowe przez wszystkie kolejne warstwy.

4. **ResNet50** [28] - wykorzystuje połączenia szczątkowe, aby zminimalizować wpływ problemu zanikającego gradientu na trening sieci, a tym samym umożliwić tworzenie głębszych modeli. Mniejszy z dwóch zastosowanych modeli ResNet. Modele z rodziny ResNet to jedne z najpopularniejszych modeli wykorzystywanych jako szkielety innych sieci (np. do detekcji obiektów [222]).

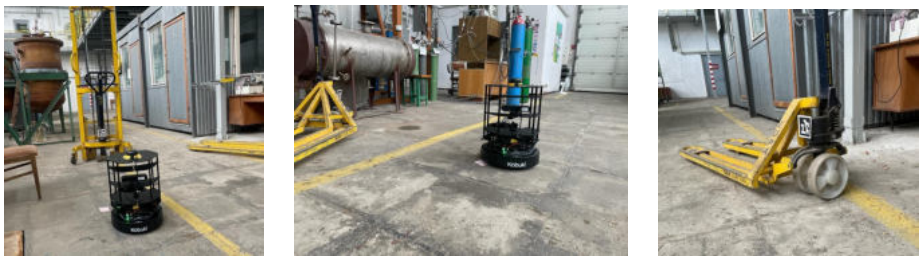
5. **ResNet101** [28] - większy z modeli ResNet.

6. **InceptionV3** [223] - sieć inspirowana architekturą typu „sieć w sieci". W celu uzyskania większego rozmiaru modelu, wykorzystuje on odpowiednio faktoryzowane konwolucje i agresywną regularyzację, aby zmaksymalizować wydajność wykonywanych operacji.

7. **Xception** [224] - model częściowo zainspirowany InceptionV3. Zawiera warstwy konwolucji separowalnej głębokościowo.

Powyższe modele wytrenowane na zbiorze danych dużej skali ImageNet są dostępne pod adresem `https://keras.io/api/applications/`. Wszystkie te modele są szeroko stosowane w literaturze, a artykuły je wprowadzające mają ponad dziesięć tysięcy cytowań. Niektóre z modeli (MobileNet [219], MobileNetV2 [220], DenseNet121 [221]) ze względu na swój rozmiar i szybkość operacji są odpowiednie dla rozwiązań mobilnych. Większe rozmiary, natomiast, zastosowano w celu porównawczym jako bardziej dokładna referencja, a także aby zbadać kompromis między rozmiarem modelu a jego dokładnością dla danych zebranych w rzeczywistym środowisku przemysłowym.

Eksperymenty przeprowadzono w rzeczywistej hali przemysłowej. Wybrano 5 obiektów, które mogą pojawić się w takim miejscu: wózek widłowy, wentylator elektryczny, butlę gazową, karton i kosz na śmieci. Do rozpoznawania obrazów na poziomie obiektów wykorzystano modele Keras dostępne na stronie `https://keras.io/api/applications/`, dlatego też wzięto pod uwagę występowanie tych obiektów w etykietach ImageNet [225] (wielkoskalowy zbiór danych wykorzystywany do trenowania tych modeli). Do oznaczania obiektów wykorzystano znaczniki ArUco, przyklejając je bezpośrednio do danego obiektu i wiążąc określone identyfikatory znaczników ArUco z określonymi etykietami ImageNet - szczegóły przedstawiono w Tabeli 3.11. Na



**(a)** **(b)** **(c)**

**Rysunek 3.20.** Fotografie przedstawiające środowisko i platformę testową. (a), (b) Środowisko eksperymentalne z robotem, (c) Wykorzystanie znacznika ArUco do fizycznego oznaczenia dużego obiektu.

Rysunku 3.20 można zobaczyć przygotowane środowisko testowe wraz z platformą testową (rzeczywisty robot) oraz przykładowy obiekt przemysłowy z naklejonym markerem ArUco (wózek widłowy).

**Tabela 3.11.** Kodowanie różnych obiektów za pomocą identyfikatorów markerów ArUco i odpowiadających im klas ImageNet. Skrót Id. oznacza identyfikator. W kolumnie „ImageNet" zastosowano oryginalne (angielskie) nazwy klas zbioru ImageNet odpowiadające wykorzystanym w pracy obiektom.

| **Obiekt** | **Id. AruCo** | **Klasa Imagenet** | **Id. Imagenet** |
|---|---|---|---|
| Wiatrak elektryczny | 1 | „electric fan" | 545 |
| Kosz na śmieci | 3 | „ashcan/trash can" | 412 |
| Butla gazowa | 4 | „gas pump" | 571 |
| Wózek widłowy | 8 | „forklift" | 561 |
| Karton | 9 | „carton" | 478 |

**Sprzęt.** W testach w czasie rzeczywistym zastosowano prawdziwy AGV: Robota Kobuki TurtleBot 2i wyposażonego w różnorodne czujniki, z których do zbierania danych wykorzystywana została kamera RGB-D (Orbbec Astra). Baza Kobuki jest również wyposażona w komputer pokładowy - Intel NUC i5-10210U. Robotem sterowano po wyznaczonym torze. Podczas jazdy system gromadził dane i testował sieć neuronową MobileNet w czasie rzeczywistym. Do generowania metryk na podstawie zebranych wyników wykorzystano standardowy komputer stancjonarny. Skrypty do przygotowywania wyników zostały uruchomione na komputerze z ośmiordzeniowym procesorem AMD Ryzen 7 2700X, 3,70 GHz i 32 GB pamięci RAM. Oprogramowanie potrzebne do zbudowania proponowanego systemu

oraz testów online i offline umieszczono w repozytorium GitHub pod linkiem `https://github.com/iitis/aruco_labeling`.

**Eksperymenty i prezentacja wyników.** Celem rozwiązania jest testowanie systemów wizyjnych opartych na rozpoznawaniu obiektów w realistycznych warunkach. Zastosowano dwa podejścia: testowanie online i offline. W testach online równolegle dokonuje się predykcji za pomocą wybranego modelu i równocześnie tworzy się zbiór danych (w czasie rzeczywistym). Wyniki (identyfikatory zdjęć, identyfikatory wykrytych markerów i predykcje sieci neuronowej - 5 wyników o najwyższej wartości prawdopodobieństwa) są zapisywane w pliku csv. Zapisywane są również same zdjęcia w folderze (z tymi samymi nazwami identyfikującymi, co w pliku csv). Taki plik csv i folder danych można następnie wykorzystać do przedstawienia ostatecznych wyników w postaci metryk i przetestowania dodatkowych modeli w trybie offline. W celu zebrania wyników przeprowadzono dwa eksperymenty (dwa oddzielne przejazdy robotem).

W badaniu, aby ocenić działanie modeli (zarówno dla wyników uzyskanych online, jak i offline), wykorzystano dwie powszechnie stosowane metryki opisujące skuteczność rozpoznawania sieci: dokładność i znormalizowane macierze pomyłek. Dlatego skupiono się na przewidywaniach o najwyższej wartości prawdopodobieństwa (klasy te nazywa się często w literaturze TOP1). Dokładność opisuje ogólną skuteczność rozpoznawania sieci - niezależnie od przewidywanej etykiety, podczas gdy znormalizowane macierze pomyłek przedstawiają bardziej szczegółowe wyniki - dla wszystkich testowanych obiektów.

**Opracowanie wyników testowania online** W testach online przewidywania modelu są prezentowane operatorowi w formie tekstowej w czasie rzeczywistym. Ponadto predykcje wraz z prawdziwymi etykietami (w postaci identyfikatorów ArUco) są zapisywane w pliku csv. Wyniki te można łatwo wykorzystać do wygenerowania wartości miar dokładności modelu (ogólnej dokładności i macierzy pomyłek). Najpierw należy odczytać plik csv i zmapować zapisane etykiety ArUco na etykiety ImageNet za pomocą słownika - szczegóły w Tabeli 3.11. Przewidywanym etykietom, które nie zostały uwzględnione w zestawie etykiet (ponieważ wykorzystano 5 ze wszystkich etykiet ImageNet), przyporządkowano wartość -1 (klasa „inne", „pozostałe"). Ta etykieta występuje jedynie w przewidywanych etykietach. Korzystając z prawdziwych i przewidywanych etykiet, obliczono wartości wskaźników dokładności.

**Opracowanie wyników testowania offline** W tej części zastosowano podzbiór kolumn wspomnianego wcześniej pliku csv: identyfikator obrazu oraz identyfikator ArUco. Najpierw zmapowano zapisane etykiety ArUco na etykiety ImageNet za pomocą słownika (patrz Tabela 3.11). Następnie zaimportowano wybrany model, wczytano zbiór danych za pomocą odpowiedniej funkcji przetwarzania wstępnego i dokonano predykcji. Przewidziane etykiety, które nie zostały uwzględnione w prawdziwym zestawie etykiet, zmapowano na -1. Korzystając z odpowiednio zmapowanych prawdziwych i przewidywanych etykiet, wyznaczono wartości różnych wskaźników opisujących działanie sieci w zakresie rozpoznawania. Podczas testowania offline, przeprowadzono testy z bazowymi (oryginalnymi) modelami głębokich sieci neuronowych, a także ze zmodyfikowanymi modelami. Modele zmodyfikowano w taki sposób, aby ich finalne klasyfikatory (warstwy ściśle połączone) uwzględniały jedynie pięć analizowanych klas docelowych. Taka specjalizacja klasyfikatora modelu może potencjalnie zwiększyć jego dokładność poprzez zmniejszenie liczby możliwych pomyłek modelu, w szczególności między klasami mniej związanymi z rozpatrywanych problemem. W tym celu odrzuca się wszystkie neurony ostatniej warstwy sieci, które nie należą do rozpatrywanego zbioru klas. Jak rezultat otrzymywana jest sieć o takiej samej strukturze jak oryginalna sieć z wyjątkiem ostatniej warstwy. Dodatkowym plusem takiego rozwiązania jest fakt, że odrzucenie nierozpatrywanych neuronów sieci prowadzi do znacznego zmniejszenia się jej liczby parametrów. Niech $F$ będzie liczbą cech w przedostatniej warstwie sieci, a $|O|$ reprezentuje liczbę klas ImageNet (1000). Niech $|S|$ będzie liczbą pożądanych klas. Całkowita liczba parametrów w ostatniej warstwie oryginalnego modelu wynosi zatem $(F \times |O|) + |O|$ (biorąc pod uwagę wagi i bias każdego neuronu). Całkowita liczba parametrów w ostatniej warstwie modelu specjalizowanego wynosi $(F \times |S|) + |S|$. Dlatego też redukcję liczby parametrów ($\Delta P$) można obliczyć w następujący sposób:

$$\Delta P = (1 + F)(|O| - |S|) \tag{3.24}$$

Taka modyfikacja modelu nie wymaga dodatkowego trenowania, więc może zostać przeprowadzona niskim kosztem pod kątem obliczeń i czasu. Przedstawiona metoda specjalizacji modeli (manualna) zainspirowała autorkę pracy do stworzenia mechanizmu automatycznej (oraz semi-automatycznej) metody do ekstrakcji modeli specjalizowanych oraz tworzenia z nich dokładniejszych modeli zespołowych. Metodę oraz wyniki przedstawiono w publikacji [226]. Metodę przedstawioną w publikacji można wykorzystać w przypadku modeli wizyjnych sieci głębokich, których klasy są węzłami struktury leksykalnej WordNet [227], a ich podzbiór stanowi grupę klas podrzędnych określonej nadrzędnej kategorii w tej strukturze. Metoda ta korzysta z relacji semantycznych hiperonimii oraz hiponimii (związek typu i nadtypu).

(a) (b) (c) (d) (e)

**Rysunek 3.21.** Zdjęcia różnych obiektów zebrane za pośrednictwem stworzonego systemu

### 3.4.3 Wyniki

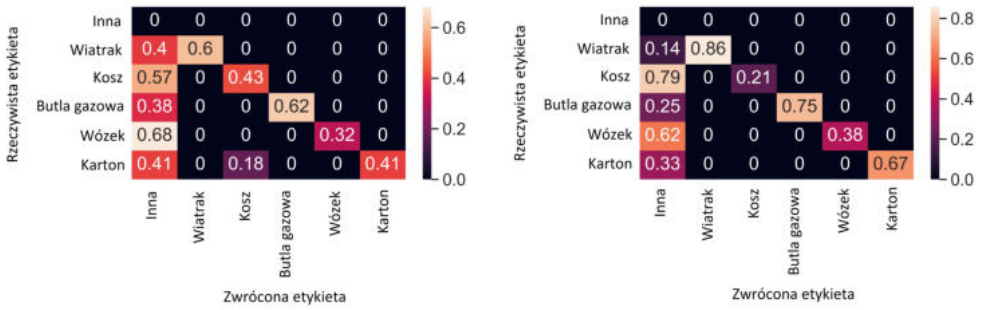
W celu prezentacji działania systemu przeprowadzono dwa eksperymenty: w Eksperyment 1 i Eksperyment 2, odpowiadające dwóm osobnym przejazdom robotem po hali przemysłowej. Na Rysunku 3.21 zaprezentowano kilka przykładowych zdjęć ze zgromadzonych na potrzeby pracy zbiorów danych. Dla badanych obrazów uzyskano 100-procentową dokładność identyfikacji markerów ArUco. Można zauważyć, że dane binarne markerów ArUco zostały pomyślnie usunięte ze zdjęć. Zdjęcia koncentrują się na analizowanych obiektach, ale zawierają niejednorodne obiekty tła - podobnie jak w rzeczywistych obiektach przemysłowych. Pomimo występowania tych obiektów, większość testowanych modeli gotowych do użycia (ang. off-the-shelf) uzyskała stosunkowo wysoką dokładność w rozpoznawaniu obiektów.

W Tabeli 3.12 zaprezentowano wyniki ogólnej dokładności uzyskane dla sieci testowanej online (MobileNet) i sieci testowanych offline. Wyraźnie widać, że większe modele uzyskały lepszą dokładność niż mniejsze, bardziej wydajne modele (np. sieć Xception uzyskała do 87 % dokładności dla zbioru danych Eksperymentu 2, a model MobileNet uzyskał 43 % w obu eksperymentach). Widać, że wydajność mniejszych gotowych modeli wiąże się z pewnym kosztem - gorszą dokładnością. Niemniej jednak nawet te modele uzyskały pewną moc predykcyjną bez dodatkowego treningu, dlatego można oczekiwać, że te gotowe modele można ulepszyć poprzez dodatkowy trening (przy użyciu konwolucyjnych ekstraktorów cech modelu do uczenia transferowego, ang. transfer learning), a następnie wykorzystać w praktycznych zastosowaniach. W nawiasach podano również wyniki ogólnej dokładności uzyskanej dla modeli specjalizowanych. Wyraźnie widać, że specjalizacja modeli znacznie zwiększa ogólną dokładność modeli (nawet do 100% w 4 przypadkach dla Eksperymentu 2).

Oprócz wyników ogólnej dokładności, przebadano także dokładność modeli dla poszczególnych klas. W przypadku testów offline wyniki wygenero-

**Tabela 3.12.** Wyniki dokładności uzyskane w testach online i offline dla rozpatrywanych modeli rozpoznawania obiektów. W nawiasach podano wyniki uzyskane dla specjalizowanych modeli testowanych offline.

| Testowanie | Model | Eksperyment 1 | Eksperyment 2 |
|---|---|---|---|
| Online | MobileNet | 43 | 43 |
| Offline | Xception | 64 (95) | 87 (100) |
| | InceptionV3 | 65 (96) | 70 (100) |
| | DenseNet121 | 50 (96) | 39 (87) |
| | ResNet50 | 68 (95) | 48 (96) |
| | ResNet101 | 62 (89) | 63 (100) |
| | MobileNetV2 | 51 (90) | 59 (100) |



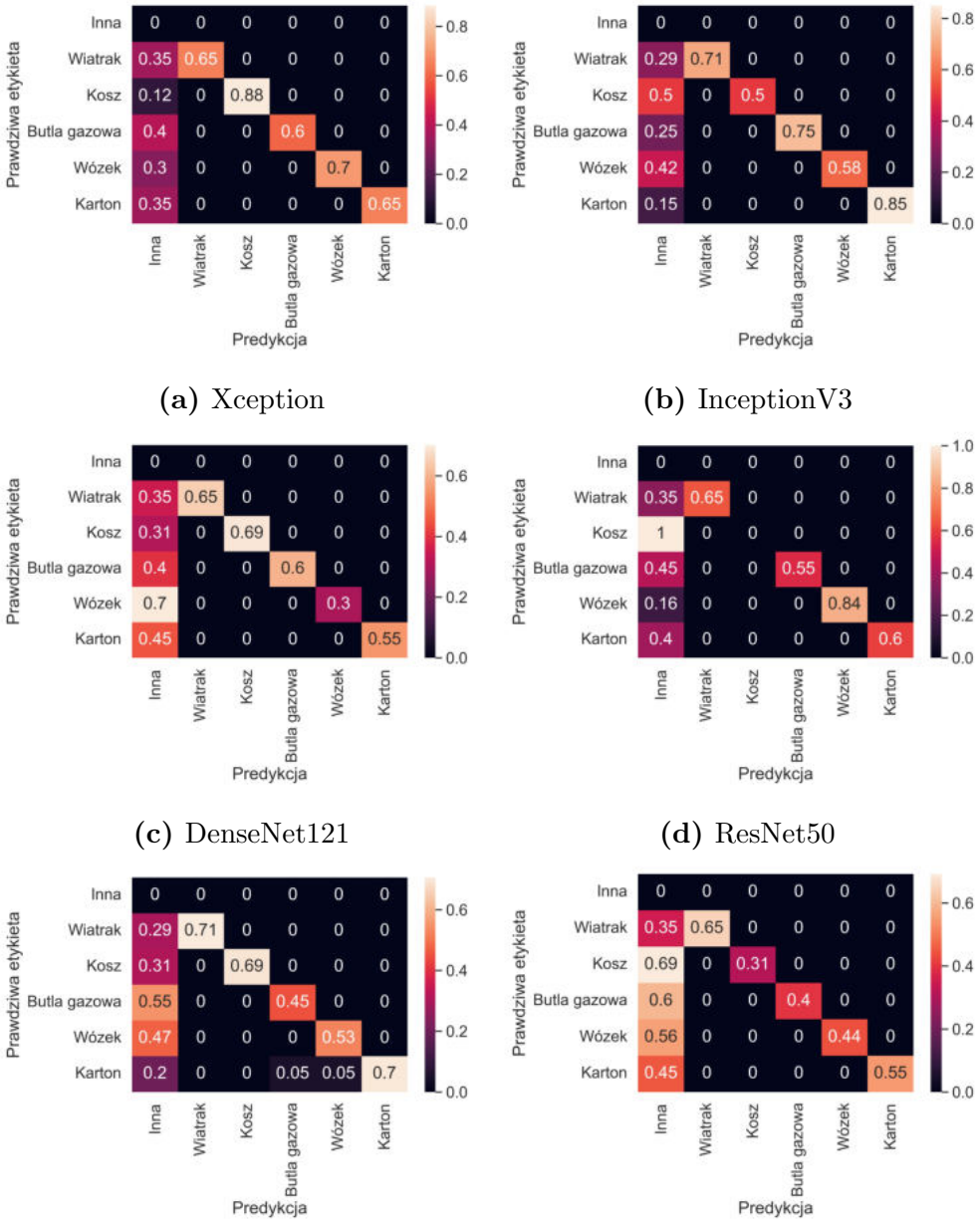

**(a)** Eksperyment 1 **(b)** Eksperyment 2

**Rysunek 3.22.** Macierze pomyłek uzyskane na podstawie wyników uzyskanych dla MobileNet w czasie rzeczywistym.

wano dla danych z Eksperymentu 1 i 2 łącznie. Na Rysunkach 3.22, 3.23 i 3.24 przedstawiono macierze pomyłek uzyskane dla testów online (MobileNet) oraz testów offline odpowiednio dla modeli oryginalnych i specjalizowanych. Widać wyraźnie, że działanie poszczególnych sieci znacznie różni się od siebie. Niektóre bazowe modele (np. Xception) mogą z łatwością rozpoznać kosz na śmieci w kadrze, a dla innych (np. MobileNetV2 lub ResNet50) obiekt ten wydaje się być bardzo wymagający. Robot był sterowany manualnie, aby zapewnić większą różnorodność między eksperymentami w postrzeganiu poszczególnych obiektów. Pozwoliło to zobaczyć, że dokładność predykcji zależy również od perspektywy, z której model obserwuje dany obiekt (co powinno być uwzględnione w zbiorach testowych) - w eksperymencie 2 dokładność rozpoznawania kartonu wyniosła 67 % w testach online, podczas gdy w eksperymencie 1 było to tylko 41 %. Niemniej jednak wszystkie modele bazowe były w stanie rozpoznać przynajmniej część zdjęć dla każdego obiektu (jedynym wyjątkiem jest ResNet50 dla klasy kosz na śmieci). Pokazuje to potencjał wszystkich sieci do wykorzystania w środowiskach przemysłowych po dodatkowym treningu (np. w oparciu o automatycznie oznakowane dane zebrane za pomocą proponowanego systemu). Eksperymenty przeprowadzone z modelami specjalizowanymi pokazały, że poprzez ograniczenie liczby klas (bez dodatkowego trenowania) znacząco można poprawić dokładność modeli. W tym przypadku wszystkie dokładności wykrywania poszczególnych kategorii z wyjątkiem klasy „Butla gazowa" były wyższe od 75% (a połowa odczytów dokładności wyniosła 100%).

### 3.4.4 Wnioski

W pracy wykazano, że markery ArUco mogą być z powodzeniem wykorzystywane do testowania systemów rozpoznawania obiektów opartych na uczeniu głębokim w czasie rzeczywistym bezpośrednio na pojazdach AGV. Takie testowanie wydaje się kluczowe dla poprawy jakości i bezpieczeństwa usług oferowanych przez głębokie sieci neuronowe wykorzystywane w praktycznych zastosowaniach Zaprezentowany system pozwolił na wykorzystanie danych zebranych podczas procedury testowania online do przeprowadzenia dodatkowych testów offline z różnymi modelami sieci konwolucyjnych na innej platformie. W ramach rozwiązania zastosowano prostą procedurę maskowania markerów, dzięki czemu utworzone za jej pomocą zbiory danych mogą być potencjalnie wykorzystywane również do trenowania modeli głębokich sieci neuronowych. Wyniki eksperymentów pokazują, że mniejsze gotowe modele wytrenowane na ImageNet uzyskują niskie wyniki dokładności w realistycznej konfiguracji przemysłowej. Lepszą dokładność można uzyskać przy użyciu większych modeli sieci. Koszt obliczeniowy tych modeli jest dość wysoki w

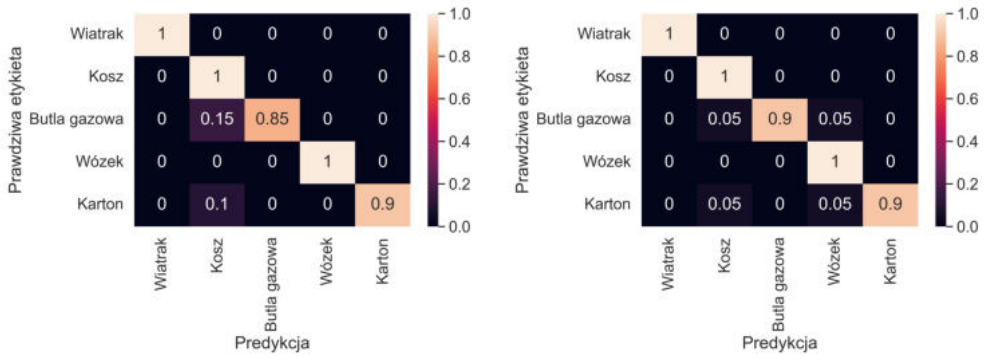



**(a)** Xception **(b)** InceptionV3

**(c)** DenseNet121 **(d)** ResNet50

**(e)** ResNet101 **(f)** MobileNetV2

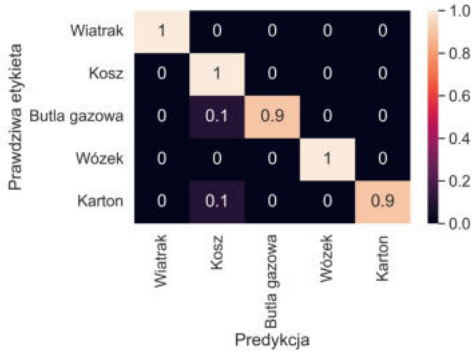
**Rysunek 3.23.** Macierze pomyłek uzyskane dla sieci testowanych offline na podstawie danych zebranych podczas testów online (dane z Eksperymentów 1 i 2 łącznie). W tym przypadku brane są pod uwagę sieci oryginalne (1000 klas ImageNet w klasyfikatorze).



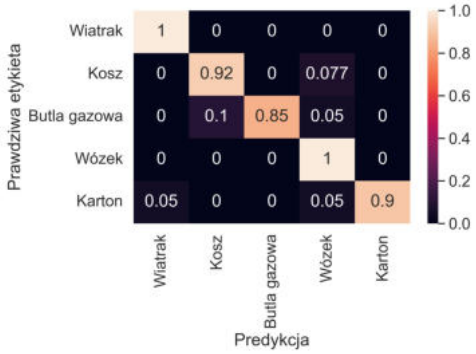

**(a)** Xception



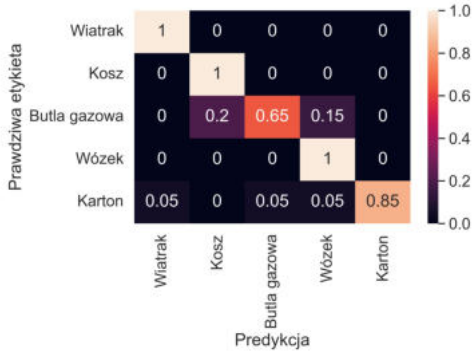

**(b)** InceptionV3



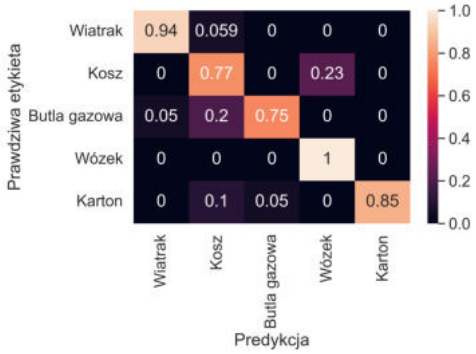

**(c)** DenseNet121




**(d)** ResNet50




**(e)** ResNet101




**(f)** MobileNetV2

**Rysunek 3.24.** Macierze pomyłek uzyskane dla sieci testowanych offline na podstawie danych zebranych podczas testów online (dane z Eksperymentów 1 i 2 łącznie). W tym przypadku brane są pod uwagę sieci zmodyfikowane w taki sposób, aby klasyfikowały obrazy jedynie do klas docelowych (z odrzuceniem reszty kategorii).

porównaniu z mniejszymi testowanymi modelami, niemniej jednak większe modele mogą być wykorzystywane jako odniesienie dla testowanych systemów rozpoznawania obiektów w procedurach testowania. Specjalizacja bazowych modeli poprzez odrzucenie klas spoza pożądanego zbioru klas znacząco poprawia dokładność sieci neuronowych pomimo braku dodatkowego trenowania modeli. Podejście to jest również proste w interpretacji i intuicyjne, a także pozwala na bardziej „modularne" traktowanie wytrenowanych głębokich sieci neuronowych. Jako rozwinięcie systemu testowania, w przyszłych pracach planowane jest stworzenie analogicznego mechanizmu testowania dla innego rodzaju sieci neuronowych - sieci do wykrywania obiektów. W tym celu planowane jest wykorzystanie dodatkowych danych - odczytów z kamery głębi. Dodatkowo planowane jest stworzenie automatycznej procedury nawigacji robota po środowisku, w którym zbierane są dane.

**Opisany w niniejszej sekcji system testowania zaprezentowano w pracy [228]**, która uzyskała częściowe finansowanie z projektu „Automated Guided Vehicles Integrated with Collaborative Robots for Smart Industry Perspective" (program Fundusze Norweskie 2014-2021, którym zarządza Narodowe Centrum Badań i Rozwoju).

## 3.5 Netsat - intuicyjny atak na pośrednie warstwy sieci głębokich

W tej części pracy przeanalizowano ostatni badany aspekt bezpieczeństwa systemów inteligentnych, obejmujący dedykowane zagrożenia związane z głębokimi sieciami neuronowymi - atakami adwersarza (ang. adversarial attacks). Pomimo doskonałych zdolności do rozwiązywania licznych problemów wizji komputerowej, głębokie Konwolucyjne Sieci Neuronowe (ang. Convolutional Neural Networks, CNN) da się w stosunkowo prosty sposób oszukać za pomocą nielosowych zakłóceń obrazu [229], które są często niewidoczne dla ludzi lub postrzegane tylko jako nieznaczne szumy. Są one niezwykle skuteczne przeciwko nawet najbardziej zaawansowanym systemom opartym na sieciach CNN [230]. Negatywnie wpływa to na bezpieczeństwo systemów wykorzystujących głębokie sieci neuronowe. Z tego powodu nie jest już wystarczające testowanie dokładności modeli sieci neuronowych i ich odporności na różne warunki środowiskowe (propozycję rozwiązania tego typu przedstawiono w sekcji 3.4), ale równie istotne stało się sprawdzanie ich odporności na ataki adwersarza. Jest to szczególnie ważne przed wdrożeniem głębokich sieci w aplikacjach wrażliwych z perspektywy bezpieczeństwa (np. w rozwiązaniach przemysłowych, czy też związanych z transportem).

Aby umożliwić efektywne testowanie odporności modeli na ataki adwersarza, środowisko związane ze sztuczną inteligencją koncentruje się aktualnie na opracowywaniu nowych skutecznych algorytmów ataku. Takie ataki pozwalają zidentyfikować słabości głębokich sieci przed ich wdrożeniem w praktycznych rozwiązaniach. Podczas gdy ataki w realnym świecie są zazwyczaj przeprowadzane w scenariuszu czarnej skrzynki, ataki typu białej skrzynki zwykle generują silniejsze próbki [231]. Ponadto, w wielu przypadkach ataki typu białej skrzynki oraz techniki transferu ataków są wykorzystywane w celu oszukania docelowego modelu bez wiedzy o jego strukturze i parametrach [141, 142]. W związku z tym, istotne jest wprowadzanie nowych skutecznych ataków typu białej skrzynki do celów testowych.

W domenie ataków adwersarza na głębokie sieci neuronowe do klasyfikacji dominują podejścia biorące pod uwagę wynik ostatniej warstwy sieci - warstwy ściśle połączonej z nieliniowością softmax [20, 142, 147]. Jedynie nieliczne podejścia skupiają się na warstwach pośrednich sieci [148]. Głębokie reprezentacje obrazów generowane na podstawie obrazów zmodyfikowanych atakami opartymi na wynikach ostatniej warstwy mają tendencję do zachowywania większości informacji semantycznych na temat obrazu, przez co sieci mogą być nadal przydatne w innych zadaniach, np. jako szkielety dla zadań takich jak detekcja obiektów [148]. Dlatego istotne jest tworzenie nowych ataków, które będą w stanie efektywnie zaburzać głębokie reprezentacje danych i potencjalnie sprawić, że więcej części sieci nie będzie w stanie poprawnie przetwarzać obrazów wygenerowanych za pośrednictwem takiego ataku. Takie ataki mogą być potencjalnie przydatne do uniemożliwienia działania modeli, które używają ekstraktorów cech modeli bazowych. Takie rozwiązania są rzadkością w literaturze. Ze względu na niewielką liczbę takich ataków, w pracy zaprezentowano nowy typ ataku - Atak Nasycenia Sieci (ang. Network Saturation Attack, NetSat). Jego celem jest zniszczenie reprezentacji danych z ostatniej warstwy konwolucyjnej sieci neuronowej (przed klasyfikatorem). Jest to realizowane poprzez nasycenie map poprzez wprowadzanie odpowiednich wzorców do oryginalnego obrazu. Celem jest sprawienie, że wszystkie kanały finalnej mapy cech są zapełnione równą wartością (można to nazwać nasyceniem sieci konwolucyjnej), a faktyczne, rzeczywiste wzorce zostają ukryte. Prezentowany atak jest wysoce modyfikowalny i może być wykorzystany w różnych wariantach: nieiteracyjnym/iteracyjnym, wykorzystującym wartości gradientu (NetSat) lub jego znak (SignedNetSat). Co więcej, atak ten jest niezależny od klasy i klasyfikatora, co odróżnia go od większości powszechnych ataków, takich jak Fast Gradient Signed Method (FGSM), a zatem w przyszłości może być potencjalnie zastosowany do zadań innych niż klasyfikacja.

Ocena wpływu ataku adwersarza na działanie sieci jest zazwyczaj mie-

rzona za pomocą miary Fooling Rate (FR) [232]. Wadą tej metryki jest fakt, że nie mierzy ona stopnia szkód spowodowanych przez atak, a jedynie ocenia sam fakt wystąpienia ataku w sposób binarny. Często może się zdarzyć, że zmiana predykcji to tylko zmiana etykiety pomiędzy bardzo podobnymi semantycznie etykietami (np. dwie rasy psów). Aby zmierzyć stopień szkodliwości ataku na działanie sieci, wprowadzono nieliczne metryki, np. [148, 233]. Charakteryzują się one jednak niespójną i nieintuicyjną interpretacją co w negatywny sposób wpływa na wyjaśnialność testowania. Dodatkowo w przypadku metryk wprowadzonych w pracy [233] nie jest trywialne zdefiniowanie ich progów tolerancji. Dlatego też zdecydowano się stworzyć Metrykę Niepodobieństwa (ang. Dissimilarity Metric, DM). Wprowadzona metryka opisuje, jak daleko jest przewidywana etykieta pod wpływem ataku od prawdziwej w przestrzeni klas (metryka opisuje stopień szkód spowodowanych atakiem). Jest to szczególnie istotne w domenach, w których pomyłki sieci neuronowych mogą skutkować wypadkami zagrażającymi bezpieczeństwu (np. pomylenie osoby z hydrantem). Aby obliczyć wartość metryki, wykorzystywana jest wyłącznie wiedza już obecna w klasyfikatorze sieci. Z tego powodu szkodliwość jest mierzona z perspektywy samej sieci neuronowej.

### 3.5.1 NetSat - Network Saturation Attack

Network Saturation Attack (NetSat) jest proponowanym atakiem adwersarza niezależnym od klasyfikatora sieci i klasy. Celem ataku jest wypełnienie finalnych konwolucyjnych map cech wysokimi aktywacjami spowodowanymi rozpoznaniem pewnych wzorców na całym wymiarze przestrzennym obrazu wejściowego. Intuicyjnie skutkuje to nasyceniem map cech wzorcami i ukryciem faktycznych wzorców, które z dużym prawdopodobieństwem zostałyby rozpoznane na czystym obrazie. Powinno to sprawić, że sieć nie będzie już w stanie rozpoznać rzeczywistych informatywnych wzorców, a co za tym idzie - obiektów.

Aby wygenerować perturbacje za pomocą NetSat, korzysta się z podejścia optymalizacyjnego opartego na gradientach sieci względem obrazów wejściowych. Perturbacje można generować w sposób nieiteracyjny oraz iteracyjny. W wariancie iteracyjnym, wynik jednej iteracji staje się wejściem do następnej, a pojedynczy krok optymalizacji pozostaje taki sam jak w wariancie nieiteracyjnym. Podczas generowania perturbacji można brać pod uwagę wartości gradientu (NetSat) lub jego znaki (SignedNetSat). Ponadto, przez dodanie kolejnego kroku obliczeniowego do sformułowania ataku, ten typ ataku może być łatwo zmieniony na PGD. Możliwe jest także wprowadzenie do ataku dodatkowych modyfikacji algorytmu najszybszego spadku, np.

momentum. Poniżej sformułowano pojedynczy krok optymalizacji obrazu:

$$x' = x - \eta, \quad \eta = \epsilon \times fun(\nabla_x J(\theta, x)), \tag{3.25}$$

gdzie $x'$ jest obrazem uzyskanym za pośrednictwem modyfikacji oryginalnego obrazu perturbacjami wygenerowanymi za pośrednictwem ataku adwersarza. $x'$ należy przeskalować do właściwego zakresu dla sieci. W pracy zdecydowano się na normalizację do zakresu przyjmowanego przez daną sieć neuronową, ale możliwe jest także zastosowanie operacji przycięcia (ang. clip). $x$ jest natomiast oryginalnym obrazem, a $\eta$ oznacza perturbacje. $\epsilon$ jest współczynnikiem stopnia perturbacji, $\nabla_x$ to gradient funkcji straty $J$ względem $x$, a $\theta$ oznacza parametry modelu. $fun()$ oznacza transformację wynikowych gradientów. Dla wariantu ze znakiem, dla każdego piksela zwraca ona znak gradientu, a dla wariantu z normalizacją - wartość znormalizowaną do wcześniej określonego zakresu (w badanym przypadku - $\langle -1, 1 \rangle$, aby minimalne i maksymalne wartości były takie same jak w metodzie ze znakiem).

Aby zdefiniować funkcję straty ($J(\theta, x)$), oznaczmy mapy cech konwolucyjnych z ostatniej warstwy konwolucyjnej sieci jako $FM$:

$$FM \in \mathbb{R}^{H \times W \times C}, \tag{3.26}$$

gdzie $H$ reprezentuje wysokość obrazu, $W$ jego szerokość (wymiary przestrzenne), natomiast $C$ oznacza liczbę kanałów. Niech $FM_{pred}$ oznacza przewidywane mapy cech (z ostatniej warstwy konwolucyjnej sieci) z aktywacjami uzyskanymi dla obrazu wejściowego $x$. Niech $FM_{max}$ będzie strukturą danych o takim samym rozmiarze jak $FM_{pred}$, ale wypełnioną maksymalną wartością zaobserwowaną w $FM_{pred}$. Formalnie można to sformułować w sposób:

$$FM_{max_{ijk}} = \max(FM_{pred}) \quad \forall i \in [1, H], j \in [1, W], k \in [1, C], \tag{3.27}$$

Każdy element $FM_{max}$ w danej lokalizacji przestrzennej $(i, j)$ i dla kanału $k$ można zdefiniować jako maksymalną wartość zaobserwowaną w mapie $FM_{pred}$. $FM_{max}$ można traktować jako mapę cech „nasyconą" maksymalną obserwowaną wartością aktywacji w tym kroku.

Funkcja straty ($J(\theta, x)$) sformułowana w przedstawiony sposób zależy wyłącznie od obrazu wejściowego $x$ i parametrów modelu $\theta$ - w przeciwieństwie do ataków takich jak FGSM. Model wykorzystany w procesie generowania perturbacji to zmodyfikowany model bazowy. Model należy zmodyfikować w następujący sposób: należy usunąć część sieci po ostatniej warstwie konwolucyjnej i zostawić w ten sposób jedynie jej konwolucyjny ekstraktor cech. Funkcję straty używaną do minimalizacji można przedstawić w następujący sposób:

$$J(\theta, x) = MSE(FM_{pred}, FM_{max}), \tag{3.28}$$

gdzie funkcja $MSE()$ oznacza błąd średniokwadratowy między mapami $FM_{pred}$ i $FM_{max}$. Jako że $FM_{pred}$ i $FM_{max}$ posiadają trzy wymiary, błąd średniokwadratowy można zdefiniować jako:

$$MSE(FM_{pred}, FM_{max}) = \frac{1}{HWC} \sum_{i=1}^{H} \sum_{j=1}^{W} \sum_{k=1}^{C} (FM_{max_{ijk}} - FM_{pred_{ijk}})^2, \quad (3.29)$$

### 3.5.2 Metryka Niepodobieństwa - Dissimilarity Metric (DM)

Metryka Niepodobieństwa (ang. Dissimilarity Metric, DM) stanowi autorską reformulację średniego wizualnego zmieszania (ang. mean visual confusion) - QI-Vis, metryki wprowadzonej w pracy [233]. Celem DM jest zmierzenie jak daleko od prawdziwej etykiety znajdują się przewidywania uzyskane na zakłóconym zestawie danych w przestrzeni cech - stopnia szkody. Odległości w przestrzeni cech są mierzone za pośrednictwem wizualnego podobieństwa między kategoriami. Ponieważ ręczne oznaczanie wizualnych podobieństw między kategoriami nie jest praktyczne, w pracy wykorzystano informacje z wytrenowanego modelu sieci neuronowej do zmierzenia tego podobieństwa. Wykorzystano w tym celu wagi związane z ostatnią warstwą ściśle połączoną sieci - warstwą klasyfikacji. Jest to warstwa sieci z aktywacją softmax. Każdy neuron tej warstwy odpowiada konkretnej klasie, a wagi łączące dany neuron z poprzednią warstwą mogą być traktowane jako szablon tej klasy. Do porównania wektorów wag odpowiadających poszczególnym klasom wykorzystano podobieństwo cosinusowe (ang. Cosine Similarity, $CS$):

$$CS(i, j) = \frac{w_i^T w_j}{||w_i|| ||w_j||}, \quad (3.30)$$

gdzie $i$ i $j$ są indeksami $i$-tej i $j$-tej klasy, a $w_i$ i $w_j$ są wagami reprezentującymi te klasy.

Aby obliczyć wartość metryki DM, należy najpierw przygotować określone struktury danych. Najpierw wykorzystywane jest podobieństwo cosinusowe i wyznaczane są wartości macierzy podobieństwa cosinusowego (ang. Cosine Similarity Matrix, CSM) z elementami określonymi w następujący sposób:

$$CSM_{i,j} = CS(i, j) \quad (3.31)$$

Następnie każdy wiersz CSM jest sortowany w malejącej kolejności i tworzona jest macierz z indeksami klas odpowiadającymi poszczególnym wartościom - posortowana macierz podobieństwa klas (ang. Sorted Class Similarity Matrix, SCSM). Każdy wiersz $c$ w tej macierzy zawiera indeksy klas najbardziej

podobnych do danej klasy $c$. Elementem na pierwszej pozycji w wierszu jest zawsze klasa $c$.

Po dokonaniu predykcji na zakłóconym atakiem zbiorze danych, można wykorzystać DM do zmierzenia szkód spowodowanych przez ten atak. Dla każdego obrazu w zbiorze należy odczytać etykietę prawdziwą $i$ i etykietę przewidzianą po ataku $j$. Należy odczytać indeks $j$ w wierszu $i$ w SCSM. Im większa jest ranga odczytanego elementu (kolejność w wierszu), tym większe szkody spowodował atak (prognoza po ataku jest bardziej niepodobna do etykiety prawdziwej). Należy powtórzyć tę operację dla wszystkich obrazów i obliczyć średnią wartość odczytanych rang. Dzieląc średnią wartość przez liczbę wszystkich klas pomniejszoną o 1, uzyskiwana jest wartość metryki DM, która mieści się w stałym zakresie $\langle 0, 1\rangle$. Poniżej zaprezentowano interpretację wartości metryki:

$$\begin{cases} 1 & \textit{maksymalna szkoda} \Rightarrow \textit{accuracy} = 0 \\ 0 \leqslant DM \leqslant 1 & \textit{wyższa wartość} \Rightarrow \textit{większa szkoda} \\ 0 & \textit{minimalna szkoda} \Longleftrightarrow \textit{accuracy} = 1 \quad (\textit{lub} \quad 100\%) \end{cases}$$

Ze względu na swoje powiązanie z ogólną dokładnością, metryka DM może być również używana wraz ze standardową dokładnością do testowania wytrenowanych sieci na czystych danych. Celem jest uzyskanie bardziej szczegółowych informacji na temat tego, jak blisko etykiet prawdziwych są etykiety przewidziane przez sieć, nawet w przypadku błędów. Należy zauważyć, że metryka działa w przeciwny sposób do dokładności. Należy również zauważyć, że $\textit{accuracy} = 0 \not\Longrightarrow DM = 1$.

Można również sformułować pewne modyfikacje metryki. Dla wysoce skutecznych modeli, podstawowa metryka DM wydaje się wystarczająca do mierzenia szkodliwości ataku i do porównywania różnych typów ataków (takie modele przeważnie zwracają prawidłowe predykcje na czystym zbiorze danych). Można również wziąć pod uwagę tylko te obrazy, dla których predykcja po ataku była nieprawidłowa. Tę modyfikację można nazwać *metryką rozbieżności dla sukcesu (ang. Dissimilarity Metric for Success, DMS)*. Można ją uzyskać w prosty sposób wykorzystując bazową metrykę DM. W przypadku gdy zwrócona klasa jest taka sama jak prawdziwa etykieta, wartość DM dla tej klasy wynosi 0, dlatego też:

$$DMS = \frac{DM \cdot N}{N(1 - \textit{accuracy})}, \tag{3.32}$$

gdzie N to liczba próbek, a dokładność jest mierzona w zakresie $\langle 0, 1\rangle$.

### 3.5.3 Metodologia badań

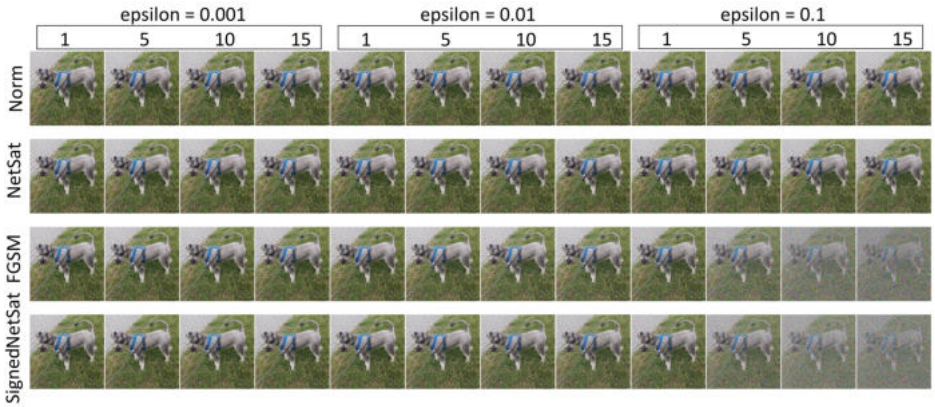
W celu zaprezentowania wyników ilościowych wykorzystano zbiór danych NIPS 2017 Adversarial Competition [234] w eksperymentach numerycznych [234]. Zbiór ten jest kompatybilny ze zbiorem ImageNet [235] (zbiory obrazów w obu zbiorach danych są rozłączne). Zbiór ImageNet to jeden z najważniejszych zbiorów danych dużej skali w domenie wizji komputerowej i klasyfikacji obrazów. Imagenet obejmuje miliony obrazów i tysiące kategorii. Kategorie te są zorganizowane według semantycznej hierarchii rzeczowników leksykalnej bazy danych WordNet [227]. ImageNet Large Scale Visual Recognition Competition (ILSVRC) lub inaczej **Imagenet-1k** - to podzbiór oryginalnego zestawu stworzony na potrzeby konkursu ILSVRC. Składa się on z tysiąca losowo wybranych klas, wybranych w celu zredukowania niejednoznaczności. Zbiór zawiera naturalne obiekty o bogatych relacjach semantycznych, ma strukturę hierarchiczną i zawiera nawet bardzo szczegółowe etykiety. Dostępna jest także duża liczba modeli wytrenowanych na tym zestawie danych. Zbiór NIPS 2017 Adversarial Competition stworzono do testowania skuteczności ataków adwersarza na sieci wytrenowane na zbiorze ImageNet. Jest on często wykorzystywany do tego celu [142, 236], ponieważ stanowi naturalne rozszerzenie testów działania modeli trenowanych na ImageNet o aspekt bezpieczeństwa. Przez to zdecydowano się go zastosować także na potrzeby niniejszej pracy wraz z modelami wytrenowanymi na zbiorze ImageNet. Do prezentacji wizualnej wykorzystano kilka dodatkowych przykładowych obrazów. Uwzględniono 2 rodzaje perturbacji: stworzone za pomocą znaku gradientu i jego znormalizowanej wartości (do zakresu obrazu obsługiwanego przez daną sieć) - daje to 4 podstawowe typy testowanych ataków (znormalizowany gradient, FGSM, NetSat i SignedNetSat) oraz dodatkowo ich wersje iteracyjne/nieiteracyjne.

Na potrzeby eksperymentów wykorzystano trzy wiodące sieci neuronowe wytrenowane na zbiorze danych IamageNet:

1. **Xception** [224] - model zainspirowany architekturą sieci neuronowych Inception, który zastępuje faktoryzowane konwolucje warstwami konwolucji separowalnej głębokościowo.

2. **MobileNetV2** [220] - model wykorzystujący odwróconą strukturę rezydualną, w której wejście i wyjście bloków rezydualnych są warstwami wąskiego gardła. Model opiera się na lekkich konwolucjach głębokościowych.

3. **ResNet50V2** [237] - następca sieci ResNet50. Oprócz klasycznych połączeń rezydualnych, w modelu zastosowano odwzorowania tożsamościowe jako połączenia omijające i aktywacje po operacji dodawania.




**Rysunek 3.25.** Wygląd próbek wygenerowanych przy użyciu różnych typów ataków i MobileNetV2: Znormalizowany gradient, NetSat, FGSM i Signed NetSat. Przetestowano różne wartości współczynnika epsilon i liczby iteracji - celem było sprawdzenie wpływu szerokiego spektrum perturbacji - od bardzo słabych do tych, które są widoczne (z założeniem, że główny obiekt jest nadal rozpoznawalny dla człowieka).

Zastosowane w celu testów sieci reprezentują modele od małych do średnich rozmiarów, wykorzystują różne techniki oraz wykorzystują różne rozmiary filtrów konwolucyjnych. Są one również często używane w różnych pracach w literaturze, np. modele z rodziny ResNet są jednymi z najpopularniejszych modeli wykorzystywanych jako szkielety w celu wykonywania innych zadań [222].

W badaniu zadano różne wartości parametrów optymalizacji w procesie tworzenia perturbacji. Przetestowano różną liczbę iteracji ($\in 1, 5, 10, 15$, gdzie 1 daje wariant nieiteracyjny. Wykorzystano też różne wartości $\epsilon$: $\epsilon \in 0.001, 0.01, 0.1$. Aby jak najpełniej przedstawić spektrum próbek zaatakowanych pod względem parametrów optymalizacji i widoczności perturbacji, generowane próbki obejmują zakres od bardzo słabych (niewidocznych) perturbacji, aż do widocznych perturbacji (z założeniem, że główny obiekt jest nadal wyraźnie widoczny). Spektrum to zostało przedstawione na Rysunku 3.25.

W celu zbadania efektywności ataku NetSat, porównano uzyskane przez niego wyniki z dwoma referencyjnymi typami ataku - FGSM oraz ze znormalizowanym gradientem. Ponieważ prezentowany atak jest atakiem nieukierunkowanym, w celu porównania z tymi typami ataków, wykorzystano maksymalizację funkcji straty (Kategoryczna Entropia Krzyżowa) względem etykiety prawdziwej(podejście klasyczne). W przypadku wariantów NetSat, minimalizuje się błąd średniokwadratowy pomiędzy przewidywanymi końcowymi

| **(a)** NetSat Predykcja: butelka wody (0.04) DM: 0.98 | **(b)** Norm Predykcja: Seter angielski (1.0) DM: 0.44 | **(c)** NetSat Predykcja: waza (0.09) DM: 0.92 | **(d)** Norm Predykcja: Border terrier (1.0) DM: 0.2 | **(e)** NetSat Predykcja: zegar ścienny (0.1) DM: 0.52 | **(f)** Norm Predykcja: Wilczarz irlandzki (1.0) DM: 0.08 |
|---|---|---|---|---|---|

**Rysunek 3.26.** Porównanie wyników NetSat i metody referencyjnej (znormalizowany gradient) dla 15 iteracji/ $\epsilon = 0.05$/MobileNetV2. We wszystkich przypadkach model zwrócił prawidłową prognozę dla czystego obrazu: Niemiecki Owczarek, Golden Retriever i Sznaucer miniaturowy odpowiednio. Zakłócenia dla obu ataków są praktycznie niewidoczne. Próbki generowane za pomocą NetSat pozwalają na uzyskanie etykiet, które są bardzo różne od prawdziwej etykiety. Ten brak podobieństwa jest odzwierciedlony przez metrykę DM. () - zaobserwowana wartość pewności sieci dla predykcji.

konwolucyjnymi mapami cech a mapą o tych samych wymiarach wypełnioną maksymalną wartością końcowej konwolucyjnej mapy cech. W ten sposób na każdym etapie optymalizacji nie jest potrzebna etykieta klasy, a wykorzystuje się tylko reprezentacje danych uzyskane z sieci. W pracy prezentowane są wyniki uzyskane dla kilku przykładów oraz wyniki analizy ilościowej w postaci wykresów dla metryk FR, DM, DMS.

### 3.5.4 Wyniki

Na Rysunku 3.26 przedstawiono porównanie jakościowe wyników dla ataku NetSat oraz znormalizowanego gradientu. Można zauważyć, że próbki wygenerowane za pomocą NetSat mogą skutecznie oszukać model MobileNetV2 oraz spowodować niepodobne do prawdziwej etykiety predykcje o niskiej pewności sieci. Próbki wygenerowane za pośrednictwem metody referencyjnej - znormalizowanego gradientu - skutkowały znacznie bardziej podobnymi do prawdziwych predykcjami (różne rasy psów). Podobieństwo predykcji do tych prawdziwych zmierzono za pomocą metryki Dissimilarity Metric. Uzyskane wyniki są wysokie dla różnych klas (np. para Owczarek niemiecki - butelka wody uzyskała DM = 0,98) i niskie dla bardziej podobnych (np. DM = 0,08 dla pary Sznaucer miniaturowy - Wilczarz irlandzki).

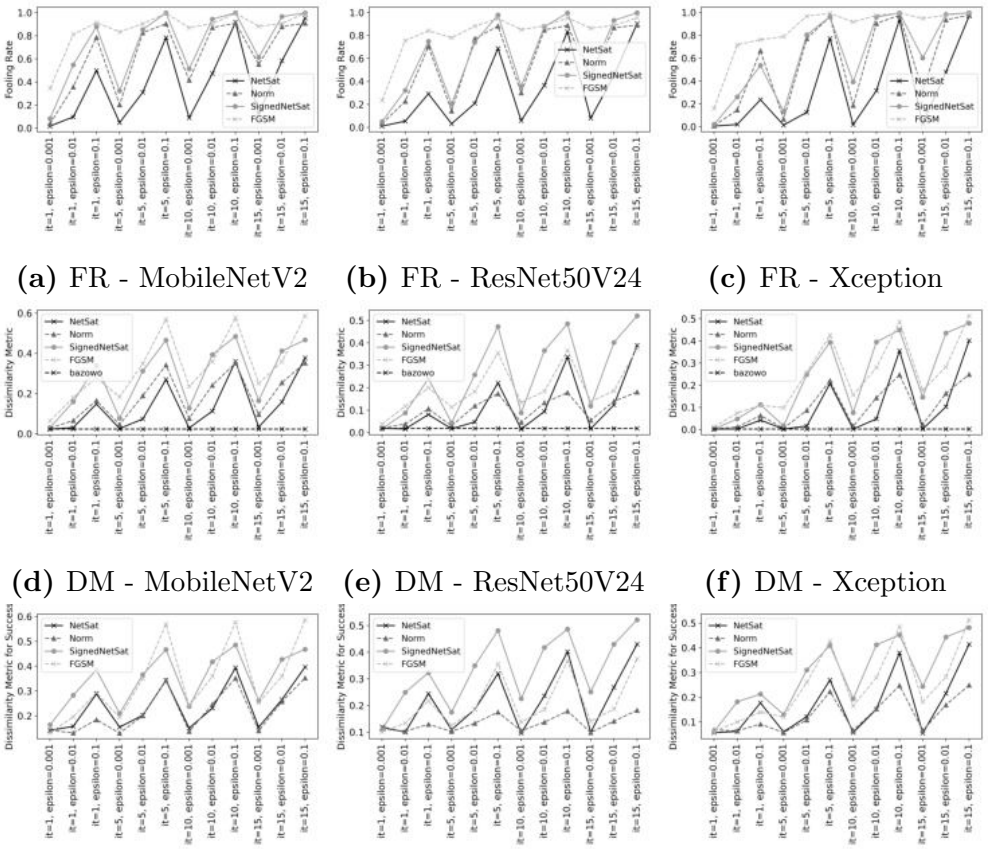
Na Rysunku 3.27 zaprezentowano wykresy analizy ilościowej (odpowied-

nio dla metryk FR, DM oraz DMS). Widać, że jeżeli brać pod uwagę jedynie metrykę Fooling Rate, to atak FGSM daje najlepsze wyniki w większości przypadków (niemniej jednak w niektórych przypadkach, np. dla $\epsilon = 0.01$ oraz 10 i 15 iteracji dla sieci MobileNetV2, SignedNetSat przewyższa FGSM). Pod względem FR, NetSat jest słabszy od swojego tradycyjnego odpowiednika - Normalized Gradient, jednak wariant NetSat ze znakiem w większości przypadków przewyższa atak referencyjny ze znormalizowanym gradientem.

Przewaga ataku NetSat jest najbardziej widoczna, gdy pod uwagę brana jest metryka DM. Pokazuje ona, że badane ataki referencyjne dla niskich wartości $\epsilon$ i dużej liczby iteracji generują słabsze próbki pod względem niepodobieństwa (nawet FGSM, który osiąga najwyższe wartości FR dla tych wartości). Oznacza to, że inne ataki, pomimo że powodują zmianę etykiety, to zwracają one predykcje bardzo zbliżone pod względem podobieństwa wizualnego do prawdziwej wartości. DM pokazuje, że w niektórych przypadkach można uzyskać znacznie bardziej niepodobne predykcje, korzystając z SignedNetSat, niż z innych ataków. Różnica pomiędzy SignedNetSat a drugim najlepszym atakiem - FGSM - może wynosić nawet około 0,2. Co więcej, na Rysunkach 3.27d - 3.27f przedstawiono wyniki uzyskane dla każdej sieci w przypadku czystego zbioru danych (bez ataku). Oznaczono je etykieta na wykresie „bazowo" Można zauważyć, że wszystkie sieci zwracają bardzo zbliżone do prawdziwych etykiet predykcje na czystych danych (pomimo że nie osiągają one 100% dokładności na czystych danych, ale ok. 83% dla MobileNetV2, 84% dla ResNet50V2 i 97% dla Xception). Przewaga wariantów NetSat jest jeszcze bardziej widoczna w przypadku zastosowania metryki DMS. Dla wszystkich sieci można zaobserwować przypadki, w których wariant NetSat ze znakiem był w stanie spowodować najbardziej niepodobne do prawdziwych predykcje. Również wariant znormalizowany NetSat jest lepszy od standardowego znormalizowanego gradientu, który daje najsłabsze wyniki ze wszystkich ataków pod względem DMS. Dla ResNet50V2, SignedNetSat był we wszystkich przypadkach lepszy (DMS) od reszty ataków (także w większości przypadków dla innych sieci). Pokazuje to, że sieci różnią się pod względem odporności na stopień szkód spowodowanych atakami adwersarza.

### 3.5.5 Wnioski

W pracy zaprezentowano nowy atak adwersarza, który w przeciwieństwie do większości istniejących ataków jest niezależny od klasy i klasyfikatora. Jego celem jest nasycenie reprezentacji obrazu i ukrycie faktycznych wzorców wizualnych w danych ostatniej warstwy konwolucyjnej sieci neuronowej. Chociaż wyniki uzyskiwane przez prezentowany atak są niższe niż dla ataków referencyjnych z perspektywy metryki Fooling Rate, to wyniki uzyskane za




**(a)** FR - MobileNetV2 **(b)** FR - ResNet50V24 **(c)** FR - Xception

**(d)** DM - MobileNetV2 **(e)** DM - ResNet50V24 **(f)** DM - Xception

**(g)** DMS - MobileNetV2 **(h)** DMS - ResNet50V24 **(i)** DMS - Xception

**Rysunek 3.27.** Wartości metryk Fooling rate, Dissimilarity Metric oraz Dissimilarity Metric for Success uzyskane dla analizowanych ataków oraz sieci.

pośrednictwem proponowanych metryk stopnia szkody pokazują, że atak jest w stanie spowodować większą szkodę wizualną w przypadku sukcesu ataku. Może to umożliwić testowanie sieci neuronowych z innej perspektywy. Jako że proponowana metoda jest elastyczna i może być zastosowana w różnych wariantach: z wartościami lub znakami gradientu, w sposób nieiteracyjny i iteracyjny sprawia, że istnieje duża szansa wykorzystania jej w procesie praktycznego testowania rzeczywistych systemów opartych na głębokich sieciach neuronowych. Wskazuje także na to fakt, że metoda jest również prosta w modyfikacji, np. poprzez zastosowanie różnych technik optymalizacji gradientu, takich jak momentum czy przyspieszony gradient Nesterova, a także łatwość przekształcenia na ataku na wariant PGD. Dodatkowo przedstawiona metryka także wydaje się być dobrze przystosowana do praktycznego testowania skuteczności ataków oraz odporności sieci neuronowych. Metryka jest prosta do wyznaczenia i w interpretacji, a jej wykorzystanie nie wymaga ustawiania żadnych progów wartości definiujących sukces ataku. Interpretowalność metryki ma pozytywny wpływ na aspekt wyjaśnialności sztucznej inteligencji w kontekście testowania i bezpieczeństwa. Chociaż metryka została w pracy zaliczona do części związanej z bezpieczeństwem, to uwzględnienie w niej aspektu wyjaśnialności pokazuje, że bezpieczeństwo i wyjaśnialność przenikają się i wzajemnie na siebie oddziałują. Podczas tworzenia rozwiązań mających na celu ocenę i poprawę bezpieczeństwa, to jeśli to możliwe, powinno się uwzględniać aspekt wyjaśnialności. **Wyniki opracowane na potrzeby niniejszej sekcji przedstawiono w pracy [238].** Przedstawione metody można wykorzystać jako rozszerzenie metody testowania systemów wizyjnych pojazdów sterowanych automatycznie (AGV) przedstawionej w Sekcji 3.4.

# Rozdział 4

# Wyjaśnialność

W niniejszym rozdziale opisano proponowane rozwiązania mające na celu poprawę wyjaśnialności systemów inteligentnych. Chociaż głównym tematem rozdziału jest wyjaśnialność, to w przypadku jednej z prezentowanych metod poruszono również możliwość jej wykorzystania w zakresie bezpieczeństwa.

## 4.1 Network Activation Mapping - metoda wyjaśnialności dla głębokich sieci konwolucyjnych

Choć głębokie sieci konwolucyjne oferują wysoką dokładność i zdolność do rozwiązywania szerokiego zakresu problemów, ich złożoność i wykorzystywanie dużej liczby nieliniowych przekształceń danych sprawiają, że ich działanie wydaje się nieintuicyjne, a same modele i podejmowane przez nie decyzje - trudne do zinterpretowania.

W pracy [239] wykazano, że filtry konwolucyjne stosowane w warstwach konwolucyjnych zachowują się jak detektory obiektów, mimo że nie zapewniono nadzoru nad lokalizacją obiektów występujących na obrazach. Ta zdolność lokalizacji jest tracona wewnątrz warstw ściśle połączonych wykorzystywanych do klasyfikacji [58]. Pokazuje to, że sercem (lub w tym przypadku oczami) sieci konwolucyjnej jest w rzeczywistości jej konwolucyjny ekstraktor cech. Percepcja/brak percepcji danego obiektu przez wstępnie wytrenowaną sieć konwolucyjną zależy wyłącznie od ekstraktora cech, a nie od klasyfikatora. Można tu użyć analogii: w uproszczeniu, konwolucyjny ekstraktor cech to oczy, a klasyfikator to mózg (bardzo uproszczony), który przypisuje informacje semantyczne do widzianych obiektów (reprezentowanych przez wyodrębnione cechy). Jeśli ekstraktor nie wykryje określonych cech, to bez względu

na to, jak dobry jest końcowy klasyfikator (lub inny model), to nie będzie on miał na czym pracować - informacje opisujące kluczowe cechy obrazu zostaną bezpowrotnie utracone.

Ze względu na ograniczoną interpretowalność systemów inteligentnych, w przypadku ich awarii (spowodowanej przyczynami naturalnymi lub atakiem adwersarza), zazwyczaj trudno jest znaleźć przyczynę i rozwiązanie problemu. Ponieważ coraz więcej aplikacji polega na takich systemach, konieczne jest dostarczanie „przejrzystych" modeli i metod, które nie tylko wyjaśniają predykcje, ale także pomagają ocenić zdolność konwolucyjnych ekstraktorów cech do znajdowania wzorców. Istniejące mechanizmy, takie jak Class Activation Mapping (CAM), koncentrują się na odpowiedzi na pytanie kierowane do sieci *„Dlaczego to powiedziałaś?"* (ang. „Why did you say that?") [59], co narzuca zależność od klasy i ma za zadanie wyjaśnić decyzję podjętą przez końcowy klasyfikator. W przeciwieństwie do tych rozwiązań, zaproponowano prosty mechanizm, który odpowiada na pytanie *„Co widzisz na tym obrazie?"* lub *„Na co patrzysz?"*.

W niniejszej pracy przedstawiono mechanizm wizualizacji CNN o nazwie Network Activation Mapping (NAM). Jest to prosty, ale informatywny sposób wizualizacji aktywacji sieci. Mechanizm ten działa wyłącznie na konwolucyjnych mapach cech, dzięki czemu jest niezależny od klasyfikatora sieci i kompatybilny ze wszystkimi modelami CNN. Obliczając standardowe statystyki punktowe konwolucyjnych map cech, uzyskiwane są wysoce informatywne i łatwe w interpretacji mapy aktywacji, które można wykorzystać do zrozumienia konwolucyjnego ekstraktora cech i oceny jego zdolności do reprezentowania obrazu za pomocą wzorców. Wykazano, że NAM może być także użytecznym narzędziem do badania wpływu ataków adwersarza na aktywacje w końcowej konwolucyjnej mapie cech sieci. Zostało to pokazane na przykładzie NetSat - ataku przeciwnika zaproponowanego na potrzeby tej pracy. Sama metoda NAM oraz fakt jak istotne są reprezentacje danych w finalnej warstwie konwolucyjnej przyczyniły się do stworzenia wspomnianego ataku NetSat. NAM stał się również inspiracją dla stworzenia autorskiej metody globalnego poolingu (ang. global pooling) jako alternatywy dla Global Average Pooling w celu przygotowania cech opisujących obraz w postaci wektorowej dla finalnej warstwy klasyfikacyjnej sieci do rozpoznawania obiektów. Ideę tę oraz wyniki przedstawiono w publikacji [240].

### 4.1.1 Proponowana metoda wizualizacji

Network Activation Mapping (NAM) - to prosty, ale informatywny sposób wizualizacji aktywacji sieci dla konwolucyjnych sieci neuronowych. W przeciwieństwie do najczęściej stosowanych metod mapowania aktywacji (np.

CAM, GRAD-CAM), opracowana metoda jest niezależna od klasy (konkretnej predykcji) i zapewnia bardziej ogólny wgląd w to, na czym koncentruje się sieć podczas procesu ekstrakcji cech, a tym samym jakie cechy/obiekty można za jej pomocą rozpoznawać. Aby wygenerować mapy aktywacji, stosowane są trzy standardowe statystyki: średnia, maksimum i wariancja (dlatego można wyróżnić trzy warianty NAM: NAM-mean, NAM-max i NAM-var). Poniżej przedstawiono wszystkie kroki niezbędne do wykorzystania NAM do wizualizacji:

1. Obliczenie punktowej wartości maksymalnej/średniej/wariancji dla wszystkich map cech (najlepiej z ostatnich warstw konwolucyjnych, ale jest to możliwe dla dowolnej warstwy konwolucyjnej).

2. Skalowanie wynikowej mapy do zakresu (0, 1).

3. Zmiana rozmiaru (przeskalowanie) mapy do oryginalnego rozmiaru badanego obrazu.

Poszczególne kroki zostały przedstawione w formie pseudokodu na Listingu 2. Należy zauważyć, że w wariancie NAM-mean zamiast średniej można obliczyć sumę punktową (wynikowa mapa będzie miała te same właściwości).

Najprostsza interpretacja różnych wariantów NAM jest następująca: NAM-max określa pokrycie, tj. **wszystkie potencjalnie informatywne regiony** obrazu (regiony, dla których mocno aktywowano którykolwiek z filtrów ostatniej warstwy - co oznacza, że znaleziono tam jakieś silne wzorce), podczas gdy NAM-mean lub NAM-var określają intensywność aktywacji filtrów w danym regionie, tj. podkreślają **najbardziej dyskryminacyjne regiony** obrazu (regiony, w których albo aktywowano wiele filtrów, albo kilka filtrów zostało bardzo mocno aktywowanych).

Zaprezentowany mechanizm NAM może być stosowany zarówno w sieciach z warstwą GAP, jak i z warstwami spłaszczającymi i gęstymi. Co więcej, może być również stosowany z innymi typami sieci, np. autokoderami. Proponowana metoda Network Activation Mapping ma wiele potencjalnych zastosowań. Po pierwsze, można ją wykorzystać do lepszego zrozumienia tego, co dzieje się w CNN i jakie cechy są reprezentowane w jej mapach cech. Analiza uzyskanych map może dostarczyć cennych informacji dotyczących przydatności badanego ekstraktora cech CNN, a także do kontroli wpływu różnych ataków przeciwnika na działanie sieci.

### 4.1.2 Metodologia badań

Aby przetestować NAM, wykorzystano pięć zaawansowanych wytrenowanych modeli. Te konkretne modele zostały wybrane ze względu na różnorodność

**Algorithm 2:** Pseudokod przedstawiający wyznaczanie wszystkich wariantów map metody wizualizacyjnej NAM.

**Input:** Mapy cech $FM$ o wymiarach $H \times W \times C$, $W$ - wariant NAM, możliwe wartości to $MAX$, $MEAN$ i $VAR$, $H_{\text{orig}}$ - oryginalna wysokość obrazu, $W_{\text{orig}}$ - oryginalna szerokość obrazu.

**Output:** Mapa podsumowująca $S$ o wymiarach $H \times W$

**switch** $W$ **do**
  **case** $MAX$ **do**
    **foreach** $i, j$ **do**
      $S_{ij} = \max_k^C FM_{ijk}$ `// Punktowe maksimum`
    **end**
  **end**
  **case** $MEAN$ **do**
    **foreach** $i, j$ **do**
      $S_{ij} = \frac{1}{C} \sum_k^C FM_{ijk}$ `// Punktowa średnia`
    **end**
  **end**
  **case** $VAR$ **do**
    **foreach** $i, j$ **do**
      $S_{ij} = \frac{1}{C} \sum_k^C (FM_{ijk} - \frac{1}{C} \sum_k^C FM_{ijk})^2$ `// Punktowa wariancja`
    **end**
  **end**
**end**
$S = \frac{S - \min(S)}{\max(S) - \min(S)}$ `// Skalowanie elementów mapy do zakresu (0, 1)`
$S = \text{resize}(S, H_{\text{orig}}, W_{\text{orig}})$ `// Zmiana rozmiaru mapy do pierwotnego rozmiaru obrazu`
**return** $S$

ich architektury. Poniżej znajduje się ich krótki opis i główny wkład.

1. **VGG16** [27] - jeden ze starszych modeli sieci CNN, w którym do klasyfikacji wykorzystywano stos warstw gęstych. Jego największym wkładem było zastąpienie dużych filtrów wieloma mniejszymi filtrami działającymi jeden po drugim.

2. **ResNet50** [28] - W tym modelu, aby umożliwić tworzenie głębszych modeli (co było problematyczne ze względu na problem zanikającego gradientu), wprowadzono połączenia rezydualne (pomijanie połączeń w niektórych stosach warstw).

3. **InceptionV3** [223] - model inspirowany koncepcją sieci w sieci. Wykorzystuje on odpowiednio faktoryzowane konwolucje i agresywną regularyzację, aby zmaksymalizować wydajność operacji w sieci i skalować jej rozmiar.

4. **Xception** [224] - częściowo zainspirowany architekturą Inception, model ten wykorzystuje warstwy konwolucyjne z separacją głębokościową. Udowodniono, że przewyższa ona InceptionV3 na zbiorze Imagenet pomimo podobnej liczby parametrów, co sugeruje bardziej efektywne ich wykorzystanie.

5. **InceptionResNetV2** [241] - model ten jest hybrydą modeli Inception i ResNet. Aby zrekompensować redukcję wymiarowości spowodowaną przez moduły Inception i wprowadzić połączenia szczątkowe, dodano warstwy rozszerzające.

Wszystkie te modele wykorzystują 3-kanałowe obrazy (RGB) przeskalowane do prawidłowego wymiaru ($299 \times 299$ dla Xception, InceptionV3 i InceptionResNetV2, $224 \times 224$ dla VGG16 i ResNet50) i wstępnie przetworzone w odpowiedni sposób. Ponieważ NAM jest metodą wizualną, przeprowadzono analizę jakościową w oparciu o przykładowe obrazy uzyskane dla wszystkich badanych sieci. W pierwszym kroku analizy wykorzystano cztery obrazy: kościoła, miniaturowego sznaucera (rasa psa), koni i narciarza biegowego. Zestaw obrazów jest niewielki, ale reprezentatywny (obrazy przedstawiają budynki, zwierzęta, osobę, grupę obiektów i obiekty w zróżnicowanych sceneriach). Dla każdego obrazu wygenerowano 3 wersje NAM: NAM-var, NAM-mean i NAM-max. Dodatkowo zostały wygenerowane wyniki GRAD-CAM dla klasy z największą wartością funkcji softmax. Przewidywania te ujęto w Tabeli 4.1. GRAD-CAM został wykorzystany jako metoda wizualna do porównania z NAM. GRAD-CAM pokazuje regiony obrazu, które

są odpowiedzialne za zwrócone przewidywania, podczas gdy NAM pokazuje ogólne skupienie sieci na wzorcach. Eksperyment porównawczy pokazuje, czy te spostrzeżenia są zgodne, czy nie. Pokazuje on również różnicę między mapowaniem zależnym od klasy a mapowaniem niezależnym od klasy.

W drugim etapie analizy wygenerowano mapy NAM-max dla Xception. Celem było przeanalizowanie stopnia pokrycia obiektów (i maksymalnego skupienia) na pięciu przykładowych obrazach przedstawiających wiele obiektów (Rys. 4.5). Zestaw obrazów jest niewielki, ale reprezentatywny dla tego zadania: zawiera zróżnicowane obiekty (ludzie, psy, samochód, kubki) i krajobraz. Do zbadania, które obiekty/obszary na obrazach są odzwierciedlone w mapach cech, wykorzystano mapy NAM-max.

W trzecim kroku analizy zastosowano atak adwersarza NetSat i wygenerowano trzy próbki dla trzech różnych ras psów. Wykorzystano 50 iteracji i $\epsilon = 0,1$, aby wygenerować silny atak. Następnie wygenerowano wyniki NAM-mean dla oryginalnych i zakłóconych obrazów i porównano je, aby pokazać wpływ ataku adwersarza, który ma na celu nasycenie analizowanych map cech.

### 4.1.3 Wyniki

#### Porównanie różnych wariantów NAM oraz GRAD-CAM

Dla każdego z 4 obrazów wygenerowano 3 wersje NAM: NAM-var, NAM-mean i NAM-max. Wygenerowano również wyniki GRAD-CAM dla klasy o największej wartości funkcji softmax (w Tabeli 4.1 zawarto te przewidywania - oznaczono je jako TOP1). Na Rysunkach 4.1, 4.2, 4.3 i 4.4 przedstawiono wyniki uzyskane dla wszystkich badanych sieci.

**Tabela 4.1.** Predykcje TOP1 uzyskane za pomocą badanych wytrenowanych modeli. Predykcje te są wykorzystywane do generowania wyników GRAD-CAM. IncResNetV2 oznacza InceptionResNetV2, a min. - miniaturowy.

| | VGG16 | Xception | InceptionV3 | ResNet | IncResNetV2 |
|---|---|---|---|---|---|
| Kościół | klasztor | kościół | klasztor | kościół | klasztor |
| Sznaucer miniaturowy | sznaucer min. | sznaucer min. | sznaucer min. | sznaucer min. | sznaucer min. |
| Konie Konie | bawół wodny | bawolec krowi | bawolec krowi | bawół wodny | bawół wodny |
| Narciarz biegowy | narty | narty | narty | psi zaprzęg | narty |

Na Rysunkach 4.1-4.4 można zaobserwować, że mapy uzyskane za pomocą NAM-mean i NAM-var uwydatniają głównie charakterystyczne obiekty

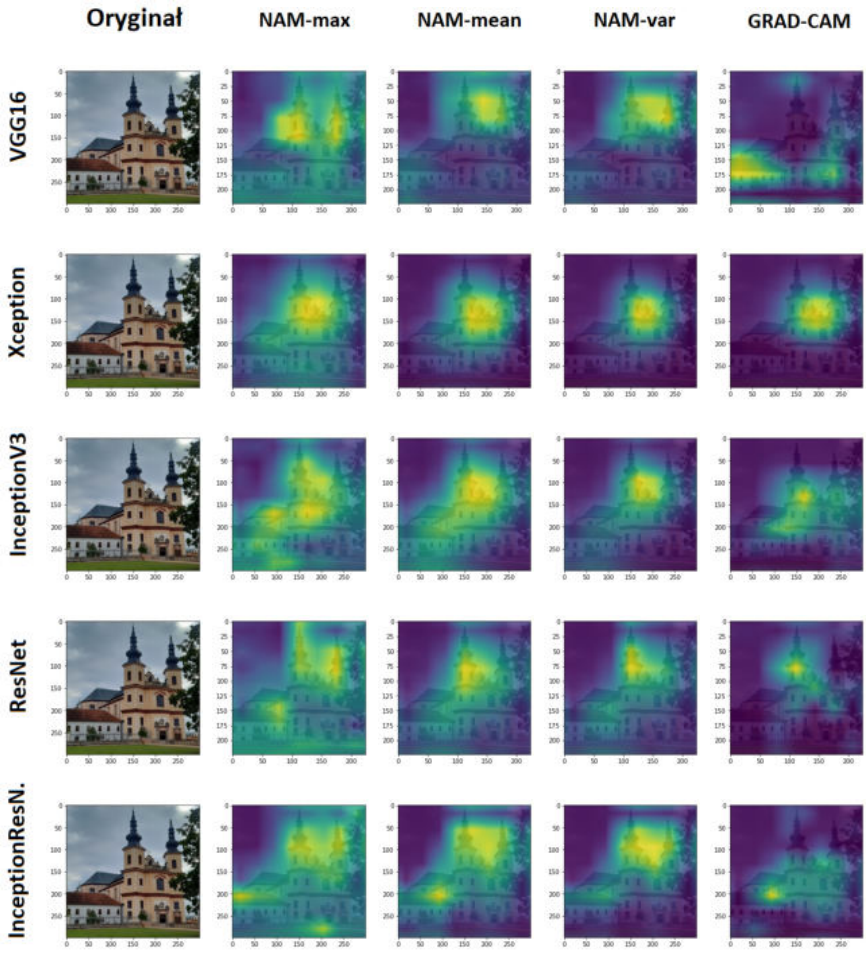
pierwszego planu badanych obrazów. W przypadku kościoła, dla wszystkich sieci obszar obejmujący wieże (charakterystyczny element architektoniczny) został wyróżniony - obserwując wyniki uzyskane za pomocą GRAD-CAM, można zauważyć, że klasyfikatory, które wykorzystały ten właśnie obszar do zwrócenia przewidywania TOP1 (Xception i ResNet) zwróciły klasę „kościół". Inne sieci (szczególnie widoczne w przypadku VGG16) skupiły się na budynku w lewym dolnym rogu i sklasyfikowały obraz jako „klasztor". Można zauważyć, że w wielu przypadkach obszary wyróżnione przez GRAD-CAM mieszczą się w tych zdefiniowanych przez NAM-mean i NAM-var. Wyjątkami są głównie przykłady, w których wykorzystano sieć VGG16. Może to być spowodowane faktem, że użycie konwolucyjnych map cech nie jest tak bezpośrednie dla tego modelu, jak w przypadku architektur opartych na warstwie Global Average Pooling. Niemniej jednak, NAM pokazuje, że sam ekstraktor cech zachowuje się podobnie do modeli opartych na GAP. Jedynym modelem z GAP, którego mapa predykcji GRAD-CAM TOP1 podkreśliła inne obszary obrazu niż NAM-mean i NAM-var, jest ResNet dla zdjęcia narciarza - klasyfikator skupił się na otoczeniu narciarza (używając NAM-max można zobaczyć, że środowisko silnie aktywowało niektóre filtry konwolucyjne) - i sklasyfikował obraz jako „psi zaprzęg" - przypuszczalnie z powodu śniegu i drzew - podczas gdy inne sieci zwróciły klasę „narty". Pokazuje to, że podczas treningu sieci uczą się również klasyfikować obiekty na podstawie ich otoczenia. Znaczy to, że sieci mogą zawierać filtry, które są w stanie znaleźć cechy otoczenia (opisujące kontekst), a nie tylko obiekty na pierwszym planie (pokazują to wizualnie mapy NAM-max).

W przypadku innych obrazów - przedstawiających psa i konie - mapy uzyskane za pomocą NAM-mean i NAM-var także skupiły się na charakterystycznych obiektach pierwszoplanowych, a równocześnie te same obszary zostały wykorzystane do zwrócenia prognozy (TOP1). W przypadku koni sama prognoza była nieprawidłowa (sieci zwróciły wartości „bawół wodny" i „bawolec krowi"). Prawdopodobnie wynikało to z faktu, że rysunek zawierał niszową rasę koni - Koniki polskie (co skutkowało charakterystyczną sierścią i inną budową ciała) oraz otaczające środowisko (sucha trawa). Niemniej jednak to właśnie obszary zawierające zwierzęta zostały wykorzystane w procesie klasyfikacji (a wynikiem były zwierzęta kopytne) - a fakt, że były to najbardziej dyskryminujące obszary, był wyraźnie widoczny na mapach uzyskanych za pomocą NAM-mean i NAM-var.

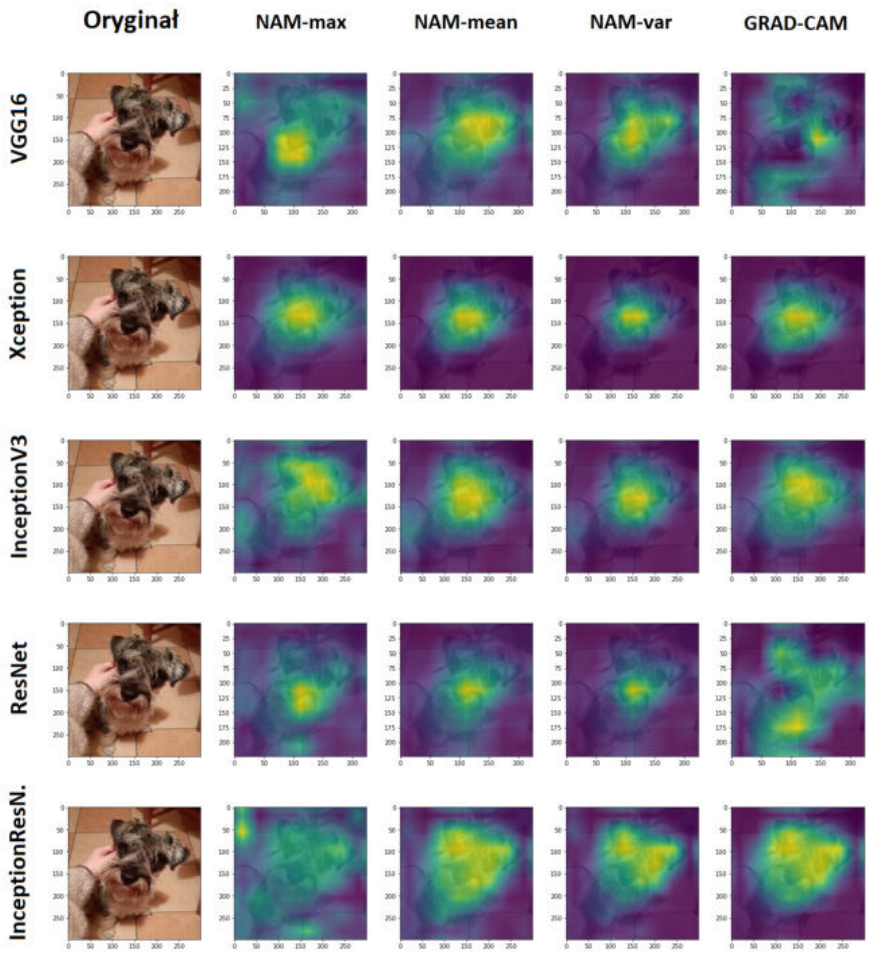
Podobieństwo map uzyskanych za pomocą NAM-mean i NAM-var dla różnych modeli sugeruje, że pomimo różnych architektur, wszystkie modele koncentrują się na podobnych obszarach obrazu w procesie ekstrakcji cech. Znaczy to, że wykorzystują one podobne wzorce do opisu tych samych obiektów. Aktywacje występujące jedynie na niewielu mapach są tłumione w pro-

cesie obliczania średniej i wariancji (NAM-mean i NAM-var), dzięki czemu NAM-max jest bardziej informatywny do obserwowania bardziej „subtelnych" zmian w postrzeganiu obiektów i skupienia. NAM-mean i NAM-var określają głównie zdolność do znajdowania najbardziej dyskryminatywnych obszarów obrazu i dobrze znanych obiektów - zazwyczaj są to obiekty na pierwszym planie (modele zostały wyszkolone do rozpoznawania obiektów na poziomie obrazu - stąd więcej filtrów jest zaprojektowanych do wykrywania charakterystycznych cech obiektów, a nie środowiska - potwierdzają to wyniki NAM-mean i NAM-var).

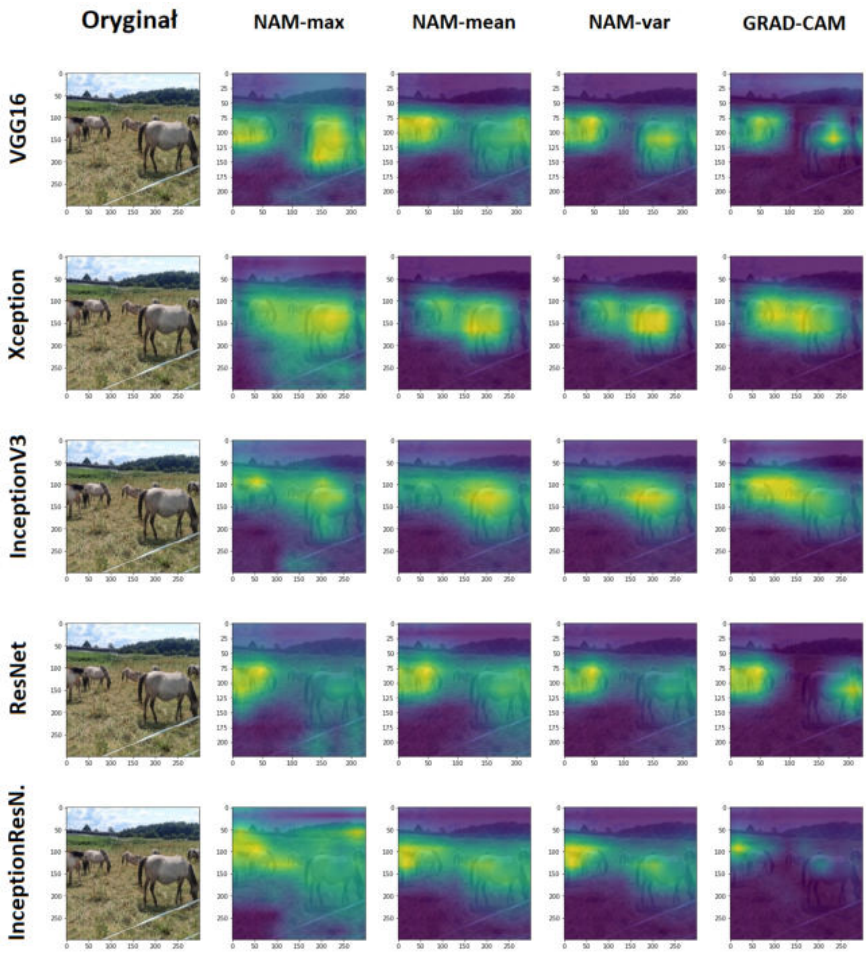
NAM-max daje wgląd w zdolność sieci do reprezentowania różnych części obrazu wejściowego - pozwala zaobserwować, które regiony obrazu aktywowały filtr jakiejkolwiek ostatniej warstwy konwolucyjnej, a które regiony nie są mocno reprezentowane. Wyniki pokazują, że Xception jest mniej wrażliwy na wzorce tła i koncentruje się głównie na regionach pierwszoplanowych. Jest to szczególnie widoczne na Rysunkach 4.2 (pies) i 4.4 (narciarz). NAM-mean ignoruje regiony, dla których tylko bardzo ograniczona liczba map cech zawiera znalezione wzorce. Mapa utworzona przy użyciu NAM-mean podkreśla regiony, dla których wiele filtrów znalazło wzorce - można je interpretować jako regiony dobrze rozpoznawane przez sieć - pokazuje to na przykład jasny pysk psa na Rysunku 4.2 (jest to bardzo dobrze znany obiekt dla sieci - zbiór danych ImageNet składa się ze 118 klas reprezentujących psy z łącznie tysiąca klas). Ta właściwość sprawia, że NAM-mean dobrze odzwierciedla właściwości sieci konwolucyjnych w zakresie rozpoznawania obiektów. NAM-var również wydaje się być trafnym mechanizmem opisującym umiejętność sieci do lokalizacji/wykrywania obiektów. Jego wyniki są jednak bardziej „rygorystyczne" niż w przypadku NAM-mean. Wariancja aktywacji sieci w różnych mapach cech może również wskazywać najbardziej dyskryminatywne regiony obrazu. Prostym wyjaśnieniem wydaje się być fakt, że duże wartości wariancji należą do punktów, dla których wiele filtrów znalazło cechy, a jednocześnie wiele różnych filtrów nie znalazło wzorców (można to zinterpretować w następujący sposób: sieć jest w stanie bardzo dokładnie rozróżnić poszczególne wzorce w tym regionie - np. widać jak wyraźnie podkreśla kijki na Rys. 4.4 - dla przypadków Xception i InceptionV3). Z tego powodu wydaje się, że NAM-mean i NAM-var są bardziej odpowiednie do analizy umiejętności sieci do wykonywania zadań takich jak klasyfikacja obiektów, wykrywanie lub lokalizacja, podczas gdy NAM-max jest lepszy do analizy ukierunkowanej na segmentację lub autokodery (w których wszystkie części obrazu powinny być silnie reprezentowane przynajmniej przez niektóre mapy).



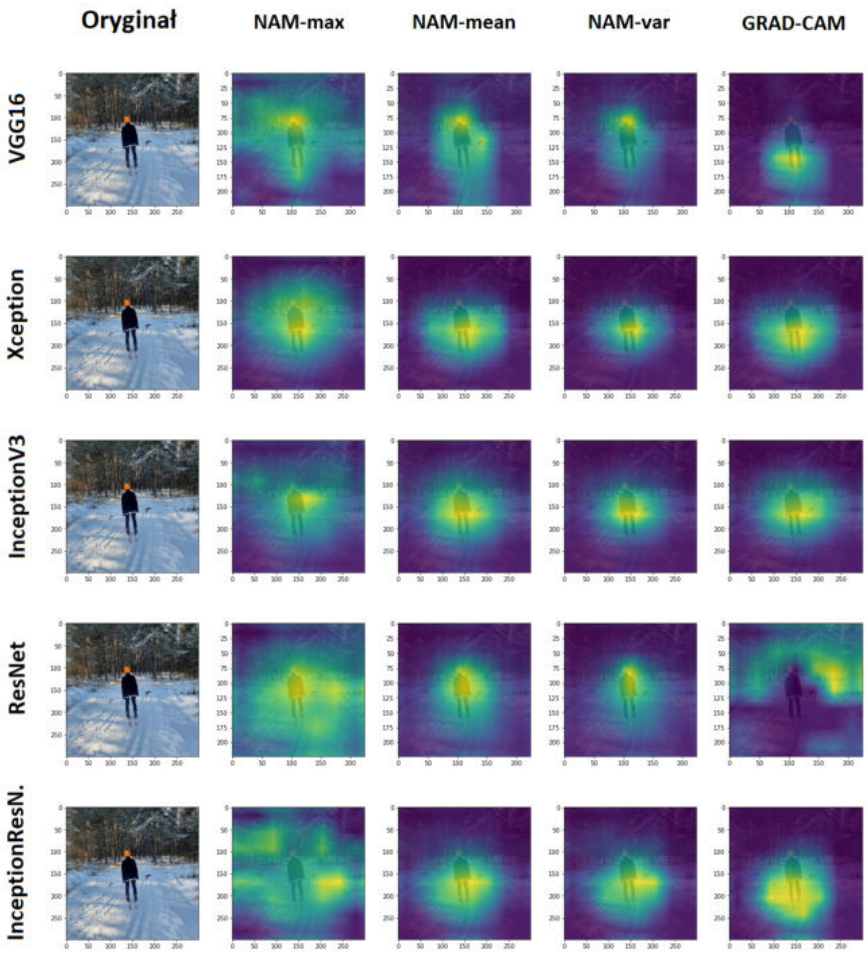

**Rysunek 4.1.** Wykorzystanie NAM-max, NAM-mean i NAM-var do wizualizacji aktywacji wytrenowanych modeli CNN. Wyniki GRAD-CAM dla klasy TOP1 zostały podane jako odniesienie. Ilustracja przedstawiająca kościół.




**Rysunek 4.2.** Wykorzystanie NAM-max, NAM-mean i NAM-var do wizualizacji aktywacji wytrenowanych modeli CNN. Wyniki GRAD-CAM dla klasy TOP1 zostały podane jako odniesienie. Ilustracja przedstawiająca sznaucera miniaturowego.




**Rysunek 4.3.** Wykorzystanie NAM-max, NAM-mean i NAM-var do wizualizacji aktywacji wytrenowanych modeli CNN. Wyniki GRAD-CAM dla klasy TOP1 zostały podane jako odniesienie. Ilustracja przedstawiająca stado koni.




**Rysunek 4.4.** Wykorzystanie NAM-max, NAM-mean i NAM-var do wizualizacji aktywacji wytrenowanych modeli CNN. Wyniki GRAD-CAM dla klasy TOP1 zostały podane jako odniesienie. Ilustracja przedstawiająca narciarza biegowego.

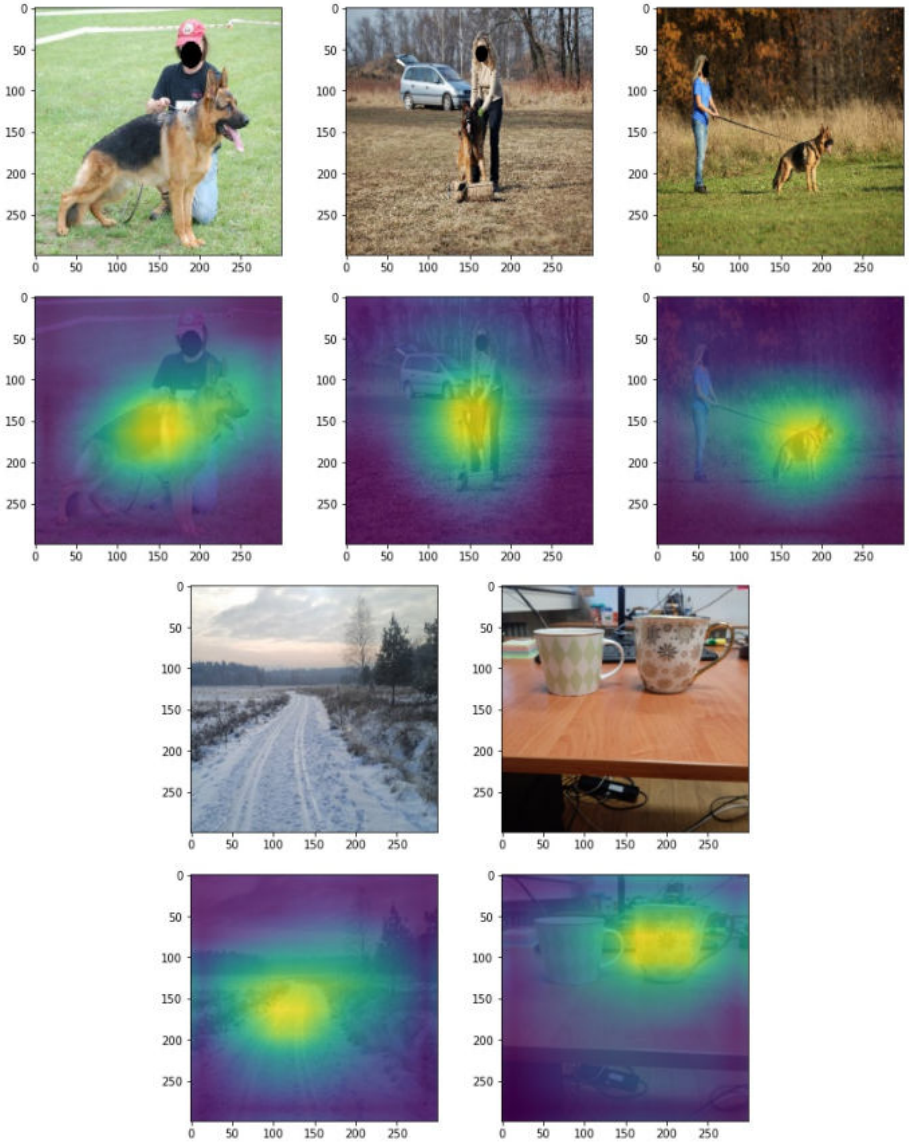
### Analiza obszarów maksymalnego skupienia sieci przy wykorzystaniu NAM-MAX

Na potrzeby analizy stopnia pokrycia (mapami cech uzyskanymi za pomocą Xception) obiektów dostępnych na pięciu przykładowych obrazach przedstawiających wiele obiektów (patrz rys. 4.5) wygenerowano mapy NAM-max. Wykorzystano mapy NAM-max do zbadania, które obiekty na obrazach są najbardziej odzwierciedlone w mapach cech (z maksymalnymi aktywacjami w jakiejkolwiek mapie cech), a zatem mają wyróżniającą się reprezentację jako głębokie cechy (które można wykorzystać w innych zadaniach). Xception koncentruje się głównie na niektórych ograniczonych widocznych obiektach na obrazie, ignorując tło i mniej znane obiekty. Na pierwszych trzech obrazach wyraźnie widać, że Xception ledwo „patrzy" na ludzi, co może być znaczącym problemem w niektórych zastosowaniach (np. wykrywanie ludzi na obrazach z kamer samochodowych. Ponadto wydaje się, że złym pomysłem jest użycie Xception do segmentacji sceny (patrz zdjęcie z zaśnieżoną drogą - sieć skupia się tylko na niewielkiej części obrazu, blisko horyzontu, i ignoruje np. drzewa po prawej stronie). Ponadto sieć silnie wykrywa jeden z kubków na ostatnim zdjęciu, a tylko niewielką w porównaniu aktywację można zaobserwować w pobliżu drugiego (ponieważ obiekty te reprezentują tę samą kategorię, aktywacje idealnie powinny być na tym samym poziomie). Na Rysunku 4.6 przedstawiono przykłady wyników lepszej jakości uzyskanych z wykorzystaniem pozostałych sieci. W tych przykładach można zaobserwować, że VGG16, ResNet50 i InceptionV3 są bardzo dokładne w wykrywaniu obiektów na pierwszym planie, a ich maksymalne aktywacje są bardziej zrównoważone między obiektami (psy, ludzie i oba kubki). Może to sugerować mniejszą tendencyjność w stosunku do określonych kategorii (istotne dla wykrywania obiektów). Z drugiej strony przykład z InceptionResNetV2 pokazuje, że sieć ta zapewnia lepsze pokrycie elementów sceny (np. las w lewym górnym rogu, drzewo w prawym rogu), co wydaje się istotne dla segmentacji).

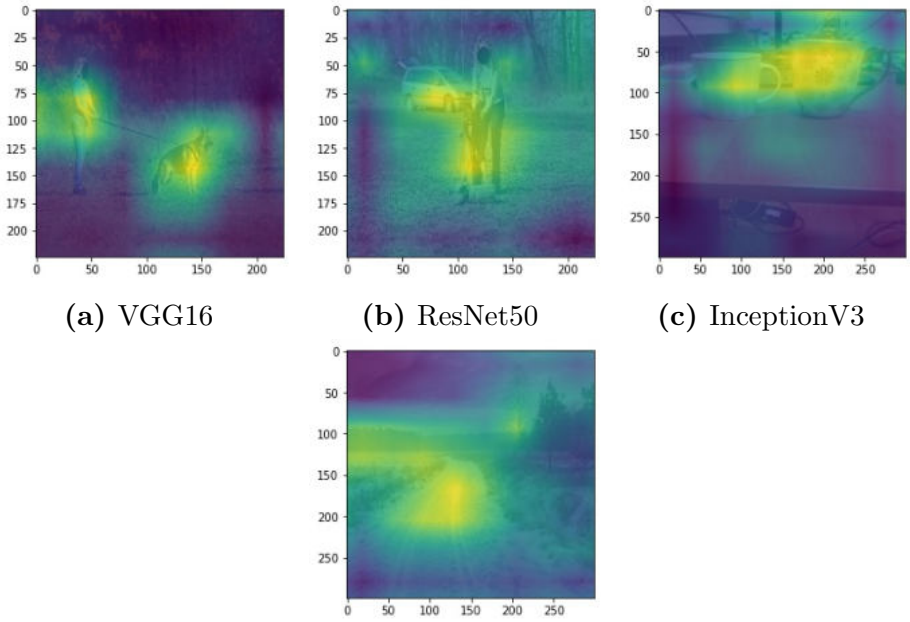
### Analiza wpływu ataku NetSat na finalne mapy cech

Zastosowano również opracowaną technikę wizualizacji - Network Activation Mapping (NAM-mean) [242], która oblicza podstawowe statystyki punktowe wszystkich końcowych aktywacji map konwolucyjnych - w celu inspekcji wpływu proponowanego w Sekcji 3.5 ataku NetSat na końcowe mapy cech.

Uzyskane wyniki pokazują, że atak NetSat wywiera wyraźny wpływ na finalne mapy konwolucyjne sieci, co pokazuje porównanie wyników NAM-mean dla obrazów oryginalnych oraz z wprowadzonymi perturbacjami. Zgodnie z



**Rysunek 4.5.** Wykorzystanie NAM-max do ręcznej analizy przydatności Xception do wyodrębniania cech opisujących różne obiekty na obrazach.



**(a)** VGG16 **(b)** ResNet50 **(c)** InceptionV3

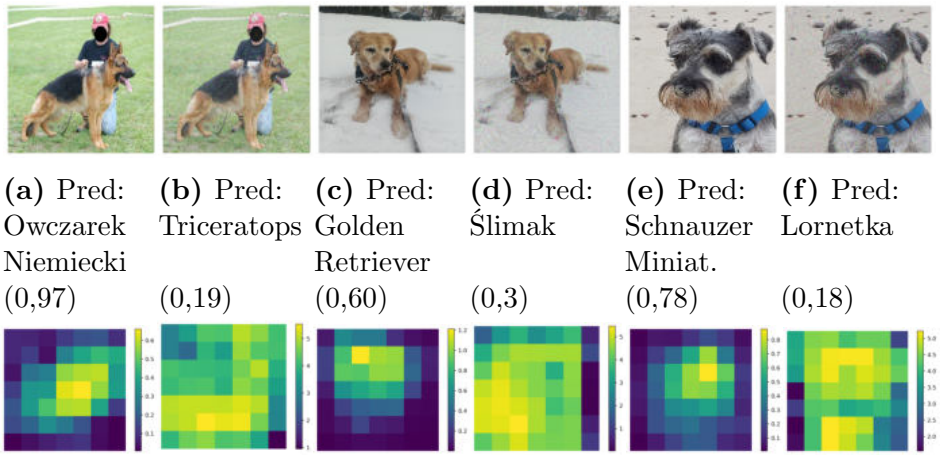
**(d)** InceptionResnetV2

**Rysunek 4.6.** Lepsze pokrycie obiektów cechami dla przykładowych obrazów za pomocą VGG16, ResNet50, InceptionV3 i InceptionResNetV2.

założeniem, atak „nasyca" wzorcami finalne mapy cech i w odróżnieniu od wizualizacji uzyskanych dla obrazów oryginalnych uniemożliwia określenie najważniejszych obszarów skupienia sieci (główne elementy pierwszoplanowe - psy). Pokazuje to potencjał NAM dla inspekcji wpływu ataków adwersarza na finalne mapy konwolucyjne cech, które są odpowiedzialne za informatywną reprezentację informacji zawartych w wejściowych obrazach.

### 4.1.4 Wnioski

Przedstawiona metoda wizualizacji dostarcza wielu przydatnych informacji na temat analizowanej sieci i tego na czym skupia się w danym obrazie. Wyniki pokazują, że wszystkie warianty NAM wykazują potencjał do wizualizacji działania sieci konwolucyjnych, interpretacji i oceny wydajności ekstraktora cech CNN. Przeprowadzone badania pokazały, że metoda może być wykorzystana do badania umiejętności ekstrakcji wzorców, stronniczości, a także wpływu ataków adwersarza na wewnętrzne reprezentacje sieci. Ze względu na swoją prostotę metoda może być z powodzeniem wykorzystana w praktycznych zastosowaniach jako szybka metoda ogólnej inspekcji działania sieci. Metoda wykazuje potencjał w inspekcji wpływu ataków adwersarza na



**(a)** Pred: Owczarek Niemiecki (0,97) **(b)** Pred: Triceratops (0,19) **(c)** Pred: Golden Retriever (0,60) **(d)** Pred: Ślimak (0,3) **(e)** Pred: Schnauzer Miniat. (0,78) **(f)** Pred: Lornetka (0,18)

**(g)** Oryginał **(h)** NetSat **(i)** Oryginał **(j)** NetSat **(k)** Oryginał **(l)** NetSat

**Rysunek 4.7.** Wpływ ataku NetSat na mapy cech z ostatniej warstwy konwolucyjnej modelu MobileNetV2. W celu generacji próbek wykorzystano 50 iteracji i $\epsilon = 0,1$. Wyniki pokazują, że NetSat nasyca konwolucyjne mapy cech i ukrywa znaczące wzorce - uniemożliwiając w ten sposób prawidłową predykcję. Pierwszy wiersz przedstawia oryginalne i zakłócone obrazy wraz z przewidywaniami TOP1, podczas gdy drugi wiersz przedstawia wizualizację NAM-mean dla odpowiednich obrazów z pierwszego wiersza. Pod obrazami (a) - (f) znajduje się predykcja (Pred:) oraz jej pewność (w nawiasie).

finalne mapy konwolucyjne sieci w procesie testowania. W przyszłych pracach planowane jest opracowanie metod w sposób numeryczny opisujących pokrycie wzorców filtrami. **Wyniki opracowane na potrzeby niniejszej sekcji przedstawiono w pracy [242].**

## 4.2 Wykorzystanie intuicyjnych metod filtrowania do stworzenia wyjaśnialnego systemu lokalizacji

Środowiska wewnętrzne są wyzwaniem w dziedzinie systemów opartych na lokalizacji. Chociaż globalny system pozycjonowania (GPS) może zaoferować bardzo dokładne szacowanie pozycji w środowiskach zewnętrznych, jest on bardzo niedokładny w środowiskach wewnętrznych z powodu blokowania sygnałów o częstotliwości radiowej przez elementy konstrukcyjne budynków [243]. W związku z tym podjęto wiele wysiłków w celu stworzenia rozwiązań,

które nie wykorzystują nawigacji satelitarnej. Takie rozwiązania opierają się między innymi na technologii Bluetooth [244, 245], ultraszerokopasmowej (ang. ultra-wideband, UWB) [246, 247] i ZigBee [248, 249], a także na podejściach hybrydowych (np. oparte na Wi-Fi i Bluetooth [250]). W ostatnich latach Bluetooth Low Energy (BLE) był najczęściej używaną technologią do estymacji dystansu i lokalizacji w pomieszczeniach ze względu na niski koszt, niskie zużycie energii, wszechobecną dostępność w urządzeniach mobilnych i zależność wskaźnika siły odbieranego sygnału (ang. Received Signal Strength Indicator, RSSI) od odległości między nadawanymi sygnałami nawigacyjnymi BLE [245, 251].

Chociaż domena estymacji bliskości i lokalizacji w pomieszczeniach jest zdominowana przez podejścia wykorzystujące wskaźnik siły odbieranego sygnału Bluetooth Low Energy, sygnał Bluetooth w środowisku wewnętrznym ma niską stabilność i jest wrażliwy na zakłócenia (przy występowaniu wielu przeszkód i odbić sygnału itp.) [252]. Aby zminimalizować te problemy, w systemach proponowanych w literaturze stosowane są zaawansowane metody matematyczne (np. filtry wygładzające i filtry falkowe), które sprawiają, że są one niepraktyczne w warunkach dużej liczby węzłów. Ponadto, gdy obiekt się porusza, jakość sygnału dodatkowo spada, a z praktycznego punktu widzenia, lokalizacja i wykrywanie bliskości może prowadzić do bardzo mylących wyników [253].

Rozwiązania lokalizacyjne są niezbędne w zastosowaniach takich jak przemysł spotkań (konferencje, targi, muzea, galerie itp.). Są one wykorzystywane w tym obszarze głównie do celów analitycznych, ale także jako pomoc dla użytkowników (np. do nawigacji w przestrzeni). Dla takich środowisk istotne jest tworzenie systemów, które oprócz dobrej dokładności lokalizacji charakteryzują się łatwością obsługi i przejrzystością działania, ponieważ często obsługują je osoby niezwiązane z branżą techniczną. Dlatego ważne jest, aby zaproponować takie właśnie rozwiązania w tym obszarze. Interpretowalność jest również ważnym kryterium w dziedzinie analizy danych, która wymaga metod, w wyniku których powstaje wiedza (np. metryki), która jest łatwo zrozumiała dla osób podejmujących decyzje [254]. Taka „praktyczna" interpretowalność wyników metod jest kolejnym ważnym aspektem wyjaśnialności systemów inteligentnych.

Celem niniejszej pracy jest stworzenie praktycznego rozwiązania lokalizacji użytkowników z telefonami wewnątrz pomieszczeń w obecności kotwic BLE. Docelowym zastosowaniem jest branża spotkań (konferencje, targi, muzea, galerie itp.). Aby osiągnąć dobrą wydajność i zminimalizować podatność na błędy, ograniczona została liczba kosztownych operacji poprzez zastosowanie prostej metody określania najbliższej kotwicy i odfiltrowania potencjalnych niewiarygodnych wyników. W celu filtrowania zaproponowano trzy

autorskie metryki bazowe: dwie opisujące aktywność osoby (statyczną/dynamiczną - przemieszczanie się/brak przemieszczania się) oraz wspólną metrykę opisującą wiarygodność danych wyników lokalizacji. Proponowane podejście do wykrywania przemieszczania opiera się zarówno na danych ze świata zewnętrznego (RSSI dla statycznych punktów - kotwic), jak i na informacji o stanie wewnętrznym badanego obiektu (wyrażonym przez odczyty z inercyjnej jednostki pomiarowej - IMU - telefonu komórkowego). Ponieważ proponowane metryki opisują stopień ruchu osoby lub wiarygodność wykrywania bliskości, są one proste w interpretacji nawet dla nietechnicznych operatorów aplikacji. Jest to istotny aspekt proponowanego rozwiązania ze względu na jego praktyczne zastosowanie.

### 4.2.1 Lokalizacja w środowiskach wewnętrznych

W sekcji opisano koncepty oraz prace powiązane z problemem lokalizacji wewnątrz pomieszczeń. Koncepty te w większości nie są wprost związane z wyjaśnialnością systemów inteligentnych, więc nie zostały one uwzględnione w analizie tematu w Rozdziale 2. Niemniej jednak, zagadnienia te są kluczowe dla samej lokalizacji, więc postanowiono je opisać w niniejszej sekcji.

Technologia BLE stała się w ostatnich latach najbardziej popularnym rozwiązaniem w dziedzinie oceny odległości i bliskości, a także lokalizacji w środowiskach wewnętrznych [251]. Podstawowym podejściem jest dopasowywanie krzywej w oparciu o zebrane pomiary RSSI w celu znalezienia odpowiedniego równania do oszacowania odległości [255]. Stosowane są również różne modele aproksymacyjne, np. [256]. Technika trilateracji oparta na RSSI jest najczęściej wykorzystywanym algorytmem do lokalizacji w sensie współrzędnych na płaszczyźnie euklidesowej [257, 258]. Dokładność obliczeń opartych na RSSI można poprawić poprzez kalibrację, użycie dedykowanych urządzeń i analizę propagacji sygnału radiowego. Takie podejście wiąże się jednak z wysokimi kosztami sprzętu i wymaga przygotowania środowiska, jak pokazano w [259].

Aby zwiększyć dokładność, rozwiązania z zakresu lokalizacji i pozycjonowania są czasami rozszerzane o dane z dodatkowych czujników [259, 260, 261]. Niektóre podejścia wykorzystują także głębokie uczenie w celu poprawy dokładności (np. uogólnioną sieć neuronową regresyjną wraz z filtrem Kalmana w pracy [262] lub autokodery odszumiające w pracy [245]). Podejścia te wprowadzają dodatkowe koszty obliczeniowe i często nie mogą być wdrażane w środowiskach, w których analizowane są dane z wielu węzłów.

W zastosowaniach takich jak analiza danych dla aplikacji logistycznych (np. targi, muzea, galerie, przemysł), śledzenie trajektorii ludzi jest zwykle wystarczające w kontekście lokalizacji. W takich zastosowaniach problem

można zdefiniować jako określanie wzajemnego położenia i wykrywanie interakcji między obiektami statycznymi i użytkownikami, a nie jako określanie lokalizacji w sensie współrzędnych. Jest to podejście stosowane w niniejszej pracy. W pracy [263] opisano metodę wykrywania trajektorii zwiedzających muzeum względem eksponatów. Podobny problem został rozwiązany w pracy [264], w której wykorzystano dedykowane kotwice BLE. W pracy skupiono się na sygnale BLE i zmienności kanału. Dane dotyczące odległości do obiektu mogą być również wykorzystywane do analizy typowych zachowań użytkowników i profili aktywności. Tego typu problem został poruszony w artykule [265], w którym wyodrębniono grupy odwiedzających reprezentujących określone profile zachowań społecznych.

Osobnym problemem ważnym dla badania jest analiza aktywności (ang. Human Activity Recognition, HAR). W tym obszarze dostępnych jest wiele prac badawczych. Można wyróżnić wiele podejść i obszarów zastosowań. Jako źródło danych wykorzystywane są zazwyczaj czujniki. Czujniki te mogą należeć do inercyjnej jednostki pomiarowej telefonu (IMU), która jest wyposażona w różne typy czujników inercyjnych (np. żyroskopy i akcelerometry). W pracy [266], autorzy zaproponowali system rozpoznawania aktywności przy użyciu klasyfikatora Naiwnego Bayesa i algorytmu K Najbliższych Sąsiadów. W artykule [267] opisano metodę rozpoznawania podstawowych aktywności, opartą odczytach z akcelerometru i żyroskopu. Wykorzystano metody uczenia głębokiego. Celem było rozpoznanie dwunastu różnych aktywności fizycznych (w tym stania, siedzenia, leżenia, chodzenia, wchodzenia i schodzenia po schodach). Średni wskaźnik rozpoznawania wyniósł 89,61%. W pracy [268] zastosowano model głębokiego uczenia (sieć głęboką z warstwami rekurencyjnymi). Do analizy wykorzystano zbiór danych z surowymi odczytami czujników noszonych na ciele. W innym badaniu, [269], autorzy wykorzystali sieci konwolucyjne do wyodrębnienia lokalnych cech z danych akcelerometru wraz z kilkoma prostymi cechami statystycznymi.

Artykuły wykorzystujące metody rozpoznawania aktywności człowieka w celu poprawy dokładności usług opartych na lokalizacji są rzadkie w porównaniu z artykułami skupiającymi się tylko na jednym z tych aspektów (lokalizacji lub rozpoznawaniu aktywności). Dlatego w niniejszej pracy zaproponowano system, który ma na celu lokalizowanie użytkowników ze smartfonami w środowisku ze statycznymi kotwicami z wykorzystaniem filtrowania niewiarygodnych estymacji na podstawie aktywności. Zaproponowano trzy różne metryki, aby odfiltrować użytkowników, którzy poruszają się między kotwicami, a tym samym poprawić dokładność lokalizacji, która jest naturalnie słaba w przypadku zmian lokalizacji [253]. Nacisk kładziony jest na prostotę metod (w przeciwieństwie do ogólnego kierunku prac w literaturze, które proponują coraz to bardziej wyrafinowane rozwiązania, które okazują

się niepraktyczne w rzeczywistych zastosowaniach). Także temat interpretowalności i intuicyjności wyników metryk nie jest powszechnym tematem w literaturze związanej z lokalizacją, jednak można znaleźć kilka przykładów w innych dziedzinach, takich jak [270, 271]. Ponieważ charakter proponowanego rozwiązania jest praktyczny, celem jest uczynienie działania systemu prostym w interpretacji dla nietechnicznych operatorów systemu. Dlatego w procesie projektowania metryk wzięto pod uwagę aspekty interpretowalności i intuicyjności zwracanych przez nie wyników.

### 4.2.2 Proponowany system lokalizacji z uwzględnieniem prostych w interpretacji metryk

Celem proponowanej metody było stworzenie systemu przybliżonej lokalizacji osób za pomocą smartfonów w oparciu o odczyty Bluetooth i filtrowanie niewiarygodnych oszacowań lokalizacji. W celu przeprowadzenia filtrowania zaproponowano trzy autorskie metryki oparte na aktywności danej osoby (aktywności rozumianej jako przemieszczanie się). Celem przy formułowaniu metryk było zapewnienie prostej interpretacji ich wartości nawet dla nietechnicznych operatorów. Aby wykryć bliskość i zbudować metryki, które można wykorzystać do oceny niezawodności wykrywania bliskości, należy najpierw przetworzyć surowe odczyty RSSI oraz pochodzące z czujników, aby uzyskać dane niezbędne do działania opisanych metod. W Tabeli 4.2 przedstawiono niezbędne kolumny tabeli danych wraz z opisem poszczególnych kolumn.

**Tabela 4.2.** Opis kolumn używanych w zbiorze danych wykorzystanym do testów obejmujących wykrywanie bliskości za pośrednictwem prezentowanej metody. W nawiasach umieszczono faktyczne stosowane nazwy kolumn. Te, faktyczne nazwy kolumn i jednocześnie oznaczenia poszczególnych nadawców zostały wykorzystane na finalnych wykresach prezentujących działanie systemu.

| **Kolumna** | **Opis** |
|---|---|
| znacznik czasu (timestamp) | Wspólny znacznik czasu dla wszystkich nadawców |
| nadawca 1 (sender_1) | Średnie RSSI dla nadawcy 1 |
| nadawca $i$-ty (sender_i) | Średnie RSSI dla nadawcy $i$ |
| nadawca $n$-ty (sender_n) | Średnie RSSI dla nadawcy $n$ |

Aby zbudować taką tabelę z danymi, dla każdej ramki Bluetooth należy pobrać odpowiednie odczyty danych czujnika (ten sam znacznik czasu).

Dane te są wykorzystywane w celu określania najbliższego stoiska oraz do wyznaczenia wartości metryk filtrujących. Następnie tworzy się tabelę, w której łączone są najbliższe ramki dla wszystkich nadawców (biorąc pod uwagę znacznik czasu nadejścia ramki). Taka tabela przechowuje odczyty RSSI dla nadawcy_1 ... nadawcy_i ... nadawcy_n dla danego znacznika czasu. Następnie wyznacza się średnie wartości RSSI dla poszczególnych nadawców w oknie czasowym, dla którego ma nastąpić określanie lokalizacji. Kolumny nadawca_1 ... nadawca_i ... nadawca_n reprezentują średnie wartości RSSI odczytane przez odbiornik dla każdego statycznego nadawcy (kotwicy) dostępnego w eksperymencie w danym oknie czasowym. W pracy wykorzystano okno czasowe o długości 5 sekund.

Wykorzystując dane w takiej formie, można łatwo określić najbliższe stanowiska w danym oknie czasowym (lub N najbliższych stanowisk). W każdym badanym oknie czasowym należy posortować nadawców malejąco według odpowiadającej im średniej wartości RSSI. Pierwszym elementem listy jest nadawca z najwyższą wartością RSSI (najniższą bezwzględną wartością RSSI). **Nadawca ten jest traktowany jako najbliższa statyczna kotwica i informacja ta jest wykorzystywana do tworzenia trajektorii użytkowników ze smartfonami.**

Na potrzeby pracy zaproponowano trzy metryki, które można wykorzystać do odfiltrowania ruchomych punktów w procesie wykrywania najbliższego stoiska oraz do oceny niezawodności tego wykrywania. Te metryki to metryka aktywności, metryka RSSI i metryka wiarygodności. Zaproponowane metryki są łatwe w interpretacji - wyższe wartości dwóch pierwszych metryk sugerują, że dana osoba porusza się, podczas gdy wyższe wartości metryki wiarygodności sugerują, że nie zaobserwowano ruchu między punktami, a zatem wyniki wykrywania bliskości można uznać za wiarygodne.

**Metryka aktywności** Aby wyznaczyć wartość metryki aktywności, pierwszym krokiem jest stworzenie binarnego modelu rozpoznawania aktywności. Taki model powinien rozważać dwie aktywności: statyczną (np. stanie, siedzenie, leżenie) i dynamiczną (np. chodzenie, bieganie). W badaniu na potrzeby pracy rozważono 4 aktywności, które zostały pogrupowane w dwie podstawowe aktywności:

- Telefon leżący na stole (aktywność statyczna)
- Osoba z telefonem stoi (aktywność statyczna)
- Osoba z telefonem powoli przechodzi z jednego punktu do drugiego (aktywność dynamiczna)

- Osoba z telefonem szybko przechodzi/biegnie z jednego punktu do drugiego (aktywność dynamiczna)

Ze względu na takie sformułowanie problemu, rozważane zadanie klasyfikacji binarnej można traktować jako wykrywanie ruchu (dlatego klasie oznaczającej ruch przypisuje się wartość pozytywną - 1). Modele wykorzystywane w pracy zostały wytrenowane na zbiorze danych stworzonym na potrzeby projektu AIMeet. Surowy zbiór danych obejmuje odczyty z jednostki IMU telefonu (ang. Inertial Measurement Unit) - akcelerometru, żyroskopu i magnetometru. Odczyty te były zależne od przybycia ramki BLE. Przetworzony zbiór danych obejmuje podstawowe statystyki odczytów w oknie czasowym 250ms (średnia, odchylenie standardowe, minimum, maksimum, mediana i skośność). Zbiór danych jest niezrównoważony, więc odrzucono losowe próbki (technika losowego odrzucania próbek - ang. random downsampling) ze zbioru treningowego dla klasy większościowej i wyrównano tym sposobem liczbę próbek z klasą mniejszościową. Powodem był fakt, że modele uczenia maszynowego są na ogół wrażliwe na niezrównoważone zestawy danych i zbiory tego typu mogą powodować ich stronniczość w kierunku klas większościowych.

Na tak przygotowanym zbiorze danych przeprowadzono selekcję cech. Redukcja wymiarowości spowodowana selekcją cech może skrócić czas potrzebny do trenowania modeli uczenia maszynowego, a także pozwala na tworzenie mniejszych, lekkich i efektywnych modeli. Takie modele są szybkie w predykcji ze względu na ich mniejszy rozmiar i mogą działać w czasie rzeczywistym. Dla takich małych modeli wystarczające są zasoby obliczeniowe typowe dla telefonów komórkowych.

W celu selekcji cech mogą być wykorzystywane różnorodne techniki. Istnieją trzy główne podejścia do tego zadania: metody oparte na filtrach, metody oparte na „opakowaniu" (ang. Wrapper-Based) i metody oparte na „wbudowaniu" (ang. Embedded-Based) [272]. Metody typu Wrapper-Based i Embedded-Based obejmują wykorzystanie pewnego modelu w procesie selekcji (np. wybierają najlepsze cechy podczas procesu uczenia danego modelu). Metody oparte na filtrach dostarczają ranking wszystkich cech na podstawie konkretnej miary istotności takich współczynników korelacji (np. Pearsona, Spearmana, Kendalla). Podejścia oparte na korelacji oceniają istotność cech na podstawie stopnia zależności zmiennej docelowej (w formie liczbowej) od wartości cech. Współczynniki korelacji Spearmana lub Kendalla wskazują na występowanie monotonicznych związków (również nieliniowych) i są nieparametryczne. Współczynnik korelacji Pearsona określa liniowe związki i jest miarą parametryczną [273]. Chociaż korzystanie z korelacji Pearsona ze zmiennymi porządkowymi może wprowadzić potencjalnie nieprawidłową

estymację związków, metody selekcji cech oparte na korelacji Pearsona są odporne na niedotrzymanie głównych założeń metody i zazwyczaj mogą skutecznie znaleźć i ocenić związek, nawet gdy założenie dotyczące zmiennych ciągłych jest naruszone [183]. Z tego powodu zdecydowano się na użycie korelacji Pearsona jako metody opartej na filtrach w procesie selekcji cech.

Aby uzyskać etykiety numeryczne dla analizowanych aktywności, zakodowano etykiety w następujący sposób dla zadania klasyfikacji binarnej:

- **Brak przemieszczania się:** Telefon komórkowy leży na stole / osoba z telefonem komórkowym stoi $\longrightarrow$ 0.0
- **Przemieszczanie się:** Osoba z telefonem komórkowym powoli przechodzi z jednego punktu do drugiego / osoba z telefonem komórkowym szybko idzie lub biegnie z jednego punktu do drugiego $\longrightarrow$ 1.0

Taka reprezentacja jest najbardziej naturalna, ponieważ w ten sposób zmienna decyzyjna jest traktowana jako zmienna porządkowa opisująca stopnień ruchu (aktywności są ustawiane pod względem stopnia aktywności).

W pracy wykorzystano trzy modele oparte na drzewach decyzyjnych, w celu wybrania najlepszego model do rozpoznawania aktywności. Wszystkie te modele trenowano i testowano w zadaniu klasyfikacji binarnej.

Wykorzystano w szczególności trzy modele: drzewo decyzyjne, drzewa wzmacniane gradientowo oraz lasy losowe. Są to jedne z najbardziej znanych algorytmów uczenia maszynowego. Drzewo decyzyjne jest dobrym punktem startowym ze względu na swoją prostotę. Algorytm ten był często używany w dziedzinie rozpoznawania aktywności (np. w pracach [274] czy [275]). Modele zespołowe (drzewa wzmacniane gradientowo oraz lasy losowe) zyskały popularność dzięki swojej przydatności przy radzeniu sobie ze statystycznymi cechami w wielu różnych dziedzinach. Przykładami są finanse [276], wykrywanie nieprawidłowości serca [277], ale także aktywności (np. klasyfikacja stylów narciarstwa alpejskiego na podstawie danych z IMU [278]). Ze względu na wysoką skuteczność opisanych algorytmów w literaturze, zdecydowano się również na ich użycie w celu stworzenia modeli zwracających wartości wykorzystywane do wyznaczenia wartości metryki aktywności.

Dokładność tych modeli przetestowano na danych testowych pochodzących z innych eksperymentów niż dane wykorzystane w zbiorze treningowym, aby uzyskać bardziej reprezentatywną ocenę. Najlepszy model wybrano na podstawie dokładności predykcji konkretnych klas przez modele (czułość i specyficzność).

Najlepszy model wykorzystano do obliczenia wartości metryki aktywności. Finalnie, ***metrykę aktywności*** można zdefiniować jako uśrednioną wartość predykcji modelu aktywności w określonym oknie czasowym (w którym

to dokonywana jest lokalizacja). Ponieważ zadanie przewidywania aktywności jest traktowane jako zadanie klasyfikacji binarnej (surowe wyniki klasyfikatora to 0 dla aktywności statycznych i 1 dla dynamicznych, to tak uzyskana wartość jest liczbą rzeczywistą w zakresie $\langle 0; 1\rangle$). Można to interpretować jako „współczynnik ruchu danej osoby".

**Metryka RSSI** opiera się na bezwzględnych zmianach w pomiarach RSSI pomiędzy dwoma kolejnymi oknami czasowymi. Jej wartość może być obliczona w następujący sposób:

Niech $X^{(t)}$ oznacza wektor zawierający średnie odczyty RSSI dla wszystkich nadawców $(n)$ dla okna czasowego $t$, gdzie $x_i^{(t)}$ oznacza średnie RSSI dla i-tego nadawcy. Następnie należy obliczyć wartości bezwzględne dla każdego elementu wektora $X^{(t)}$ - wynikiem jest wektor $|X^{(t)}|$ z elementami $|x_i^{(t)}|$. Zdefiniujmy $X_{diff}$ jako wektor zawierający bezwzględne wartości różnic średnich wartości RSSI między dwoma kolejnymi oknami czasowymi:

$$X_{diff}^{(t)} = |X^{(t)} - X^{(t-1)}| \tag{4.1}$$

Następnie należy obliczyć wektor wag, który zostanie wykorzystany do obliczenia średniej ważonej zmian średnich wartości RSSI między dwoma kolejnymi oknami czasowymi. Aby określić wartości wag dla każdej różnicy RSSI, wykorzystano funkcję *softmax*:

$$W^{(t)} = \sigma(|X^{(t)}|) \tag{4.2}$$

Aby obliczyć wartość $\sigma(|x_i^{(t)}|)$, została wykorzystany następujący wzór matematyczny:

$$\sigma(|x_i^{(t)}|) = \frac{\beta^{|x_i^{(t)}|}}{\sum_j \beta^{|x_j^{(t)}|}}, \quad \beta \in (0; 1) \tag{4.3}$$

Aby uzyskać większe wartości dla wyższych wartości RSSI, które reprezentują bliższe statyczne punkty - charakteryzujące się bardziej niezawodnymi wartościami sygnału i potencjalnie większymi zmianami w przypadku zmiany pozycji - podstawa funkcji wykładniczej musi przyjmować wartości z przedziału $(0; 1)$, jako że wartości surowe RSSI są ujemne.

Następnie należy określić wektor ważonych wartości bezwzględnych zmian RSSI ($Y^{(t)}$ z elementami $y_i^{(t)}$) i obliczyć średnią - $\mu_Y^{(t)}$ ($*$ oznacza mnożenie wartości na odpowiadających pozycjach wektorów $W$ i $X_{diff}$):

$$Y^{(t)} = W^{(t)} * X_{diff}^{(t)} \tag{4.4}$$

$$\mu_Y^{(t)} = \frac{\sum_j y_j^{(t)}}{n} \tag{4.5}$$

Ostatecznie, aby uzyskać wartość metryki reprezentującej zmiany w RSSI ($metryka\_RSSI$) dla okna czasowego $t$, należy wykorzystać funkcję tangensa hiperbolicznego, która przekształca wartość metryki do zakresu (0, 1), gdzie mniejsze wartości reprezentują mniejsze zmiany w RSSI, a większe wartości - większe zmiany (wynikające z potencjalnych zmian pozycji):

$$metryka\_RSSI^{(t)} = tanh(\mu_Y^{(t)}) \tag{4.6}$$

**Metryka wiarygodności** jest określana na podstawie wartości metryk aktywności i RSSI. Jest ona obliczana w następujący sposób:

$$\begin{aligned} &metryka\_wiarygodnosci^{(t)} = \\ &= 1 - (\alpha \cdot metryka\_aktywnosci^{(t)} + \gamma \cdot metryka\_RSSI^{(t)}), \end{aligned} \tag{4.7}$$

gdzie $\alpha$ może być opisane jako współczynnik istotności metryki aktywności:

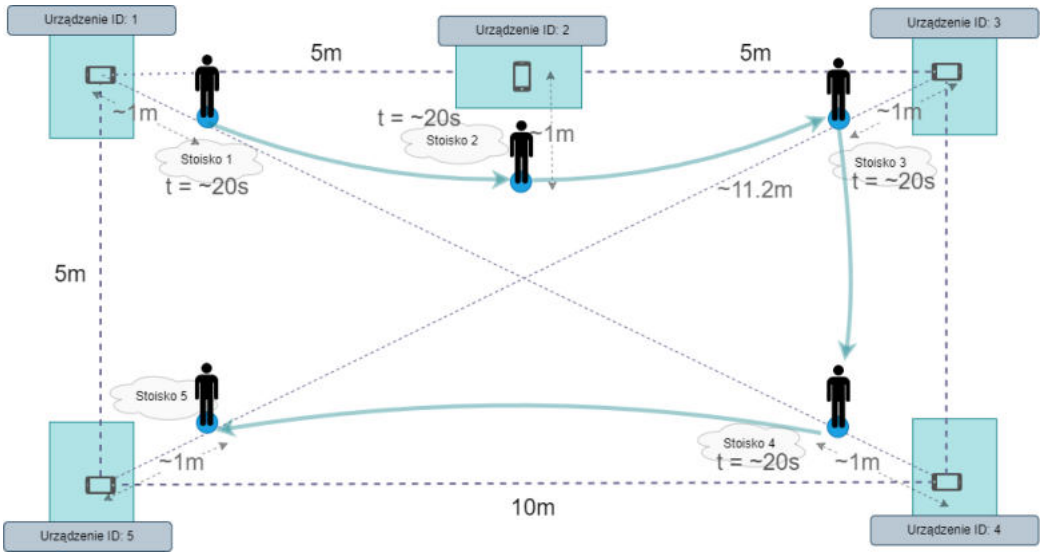
$$\alpha \in \langle 0; 1 \rangle, \quad \gamma \in \langle 0; 1 \rangle, \quad \alpha + \gamma = 1 \tag{4.8}$$

W rezultacie uzyskana metryka niezawodności przyjmuje wartości z zakresu $\langle 0; 1 \rangle$, przy czym mniejsze wartości oznaczają mniej wiarygodne estymacje bliskości, a wyższe - bardziej wiarygodne - ze względu na brak ruchu badanej osoby.

### 4.2.3 Metodologia badań

W celu przetestowania metod przedstawionych w badaniu, wykorzystano zbiór danych zebranych na potrzeby projektu AIMeet. W celu zebrania danych, zbudowano realistyczne środowisko testowe (generalizacja hali targowej/muzealnej ze stoiskami/dziełami sztuki). Środowisko składało się z 5 telefonów komórkowych (statycznych), leżących na 5 stołach, które tworzyły kształt prostokąta (4 smartfony jako wierzchołki, a pozostały na środku jednego z dłuższych boków. Środowisko to przedstawiono na Rysunku 4.8. Stoły ponumerowano zgodnie z ruchem wskazówek zegara.

W takim środowisku umieszczono osobę z telefonem komórkowym. Eksperyment trwał ok. 2 minut. Osoba biorąca udział w eksperymencie miała zachowywać się naturalnie (oglądać stoiska i poruszać się między nimi: $1 \rightarrow 2 \rightarrow 3 \rightarrow 4 \rightarrow 5$ (przejścia naniesiono na Rysunku 4.8). Osoba stała przy danym stoisku (rozpoczynając od pierwszego) przez około 20 sekund, a po danym sygnale (druga osoba mierzyła czas i wydawała polecenia) przemieszczała się do




**Rysunek 4.8.** Plan eksperymentu. Eksperyment obejmuje przechodzenie między pięcioma stoiskami, na których znajdują się telefony komórkowe wysyłające ramki BLE. Celem jest określenie w danym oknie czasu (5 sekund) najbliższego stoiska oraz ewentualne odfiltrowanie estymacji w przypadku zaobserwowania ruchu.

następnego stoiska. Krótsze przejścia zajmowały nieco ponad 5 sekund. Podczas ostatniego odcinka (dłuższego niż pozostałe) osoba poruszała się bardziej dynamicznie, aby droga trwała porównywalnie długo jak w innych przypadkach. Jednocześnie dostarczyło to bardziej dynamiczny przypadek ruchu. Z tego powodu, w eksperymentalnej części zaznaczono momenty ruchu z pewnym marginesem (2 okna czasowe w pierwszych dwóch eksperymentach i trzy w dwóch ostatnich) - po czasie trudniej było precyzyjnie określić rzeczywiste ramy czasowe okien czasowych statycznych i dynamicznych.

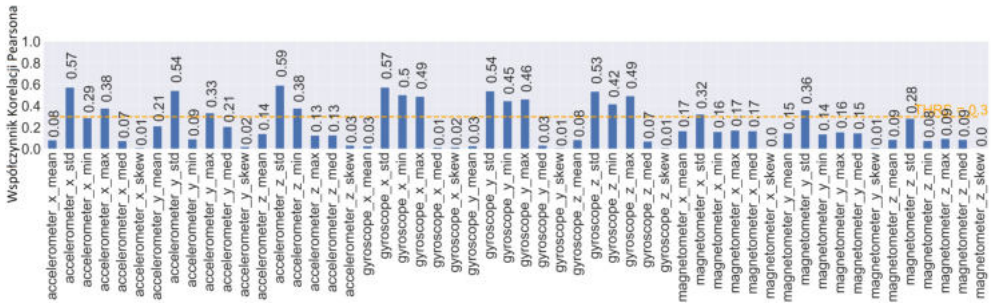
Do zbierania danych wykorzystano aplikację, która jednocześnie zbiera i wysyła ramki Bluetooth (rozgłoszeniowe) do innych zarejestrowanych urządzeń. Kiedy urządzenie odbiera ramkę Bluetooth, mierzy ono wartości zwracane przez wbudowane czujniki IMU: akcelerometr, żyroskop i magnetometr. Zbierając dane w sposób zależny od odbioru ramek Bluetooth, otrzymywane są wartości RSSI i odczyty z czujników powiązane z jednym znacznikiem czasu. Właściwość ta jest później wykorzystywana w eksperymentach. Aplikacja działa na wszystkich telefonach jednocześnie (rozpoczęcie i zakończenie zbierania danych jest zaplanowane na konkretny czas, zegary na telefonach są zsynchronizowane).

W eksperymentach wykorzystano tylko dane odbierane przez telefon trzymany przez poruszającą się osobę, a telefony leżące na stołach traktowano jako nadajniki (kotwice). Naturalnie możliwe jest użycie dwóch typów apli-

kacji - dla statycznych telefonów (leżących na stołach) i dla dynamicznych telefonów (trzymanych przez osoby poruszające się wewnątrz badanego środowiska). Proponowane rozwiązanie pozwala na wykorzystanie jednej aplikacji zarówno jako nadajnika, jak i odbiornika. Aplikacja zastosowana do pomiarów została zaimplementowana dla systemu iOS (używane były telefony firmy Apple od modeli XR do 12).

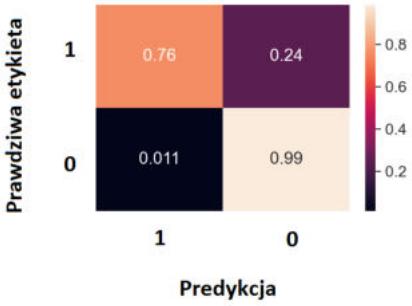
### 4.2.4 Wyniki

Na początku przeprowadzono selekcję cech bazując na wartościach współczynnika korelacji Pearsona. Wyniki pokazują, że uwzględniając tylko cechy z wartością współczynnika większą niż 0.3, możliwe było ograniczenie liczby cech uczących z 54 do 17 (wartości współczynnika dla wszystkich cech wraz z progiem odcięcia przedstawiono na Rysunku 4.9). Zdecydowana większość odrzuconych cech uzyskała bardzo niskie wartości współczynnika korelacji Pearsona - sugerując niski wpływ tych cech na wartość etykiety.

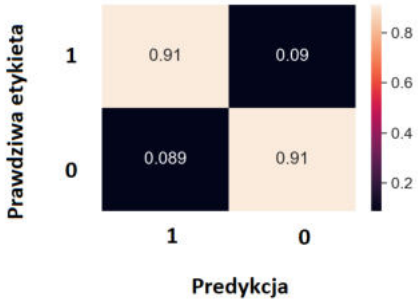



**Rysunek 4.9.** Selekcja cech z wykorzystaniem wartości absolutnej współczynnika korelacji Pearsona, ustalonej dla cech i etykiety w formie numerycznej.

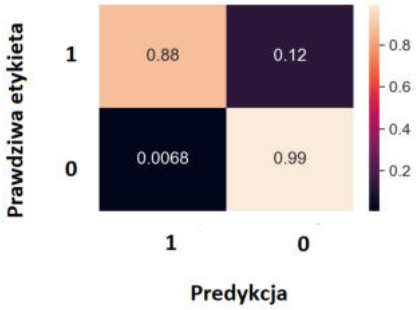
Zbiór danych treningowych został przetworzony w taki sposób, aby brać pod uwagę tylko te cechy, które zostały wybrane w procedurze selekcji cech. Tak przygotowany zbiór wykorzystano do wytrenowania trzech modeli binarnych: drzewa decyzyjnego, drzew wzmacnianych gradientowo oraz lasu losowego. Na Rysunku 4.10 zaprezentowano wyniki modeli w postaci macierzy pomyłek dla zbioru testowego. Można zauważyć, że modele zespołowe osiągnęły znacznie lepsze wyniki niż tradycyjne drzewo decyzyjne. Najprawdopodobniej wynika to z lepszej zdolności predykcyjnej i generalizacji modeli zespołowych. Najlepsze wyniki osiągnęły drzewa decyzyjne wzmacniane gradientowo. Zarówno wskaźnik predykcji prawdziwie pozytywnych (czułość), jak i prawdziwie negatywnych (specyficzność) wynosił ponad 90%, dlatego zdecydowano się na wykorzystanie tego konkretnego modelu w dalszych eksperymentach.




**(a)** Drzewo decyzyjne




**(b)** Drzewa wzmacniane



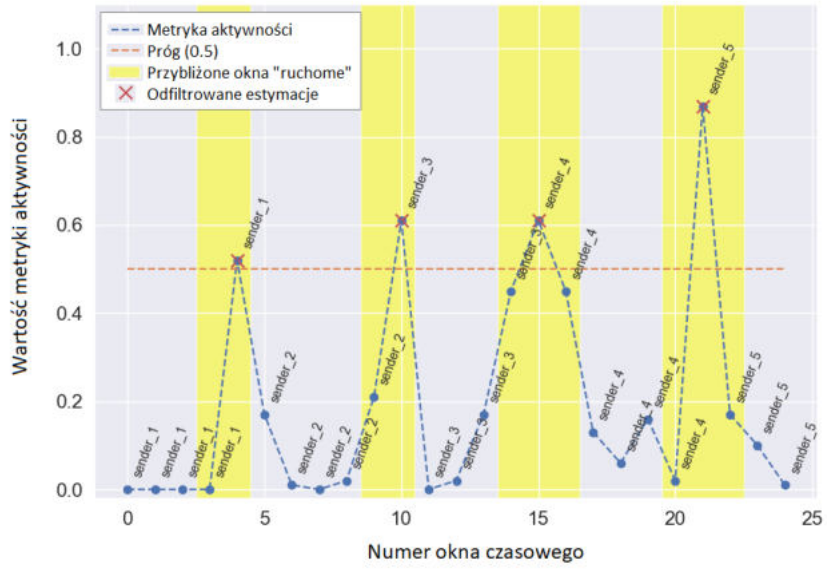

**(c)** Las losowy

**Rysunek 4.10.** Macierze pomyłek uzyskane dla testowanych binarnych klasyfikatorów aktywności.

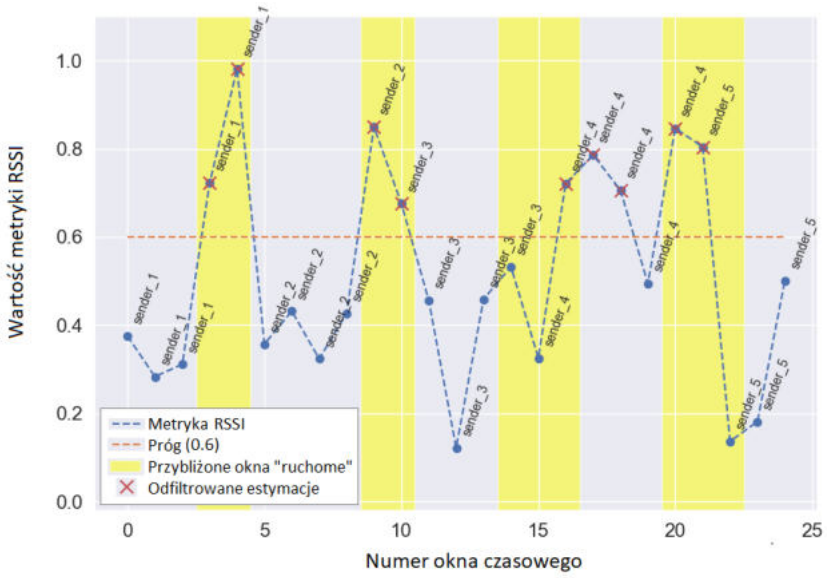
Następnie przetworzono dane zebrane w środowisku testowym. Sformatowano je w odpowiedni sposób oraz uruchomiono na nich model drzew decyzyjnych wzmacnianych gradientowo, aby uzyskać prognozy. Aby znaleźć optymalną wartość bazy funkcji softmax (używanej do obliczania wartości metryki opartej na RSSI), obliczono wartości współczynnika korelacji Pearsona pomiędzy wartościami metryki opartej na RSSI uzyskanymi przy różnych wartościach bazy (od 0.1 do 0.95 z krokiem 0.05) i wybrano tę bazę, dla której korelacja była najmocniejsza - 0.85. Podobnie wybrano optymalną wartość dla współczynnika ważności - również 0.85. Alternatywnie, wartości te mogą być wybrane przez operatora (szczególnie druga wartość - proporcja, w której brane są pod uwagę metryki oparte na RSSI i prognozach aktywności).

Na Rysunkach 4.11b, 4.11a i 4.12 przedstawiono wyniki filtrowania zwracania najbliższego stoiska dla trzech badanych metryk (*metryki aktywności*, *metryki RSSI* oraz *metryki wiarygodności* odpowiednio). W każdym badanym oknie czasowym określono najbliższe stoisko (kotwicę). Na wykresy naniesiono przybliżone obszary ruchu (okna czasowe). Wybrano również wartości progów (wartość ta powinna być wybrana zgodnie z tym, jak surowo ma być oceniana niezawodność danych) - ustawiono próg na 0,5 dla *metryki aktywności* i *metryki wiarygodności* oraz na 0,6 dla *metryki RSSI* (tutaj wykorzystano wyższą wartość, ponieważ wartości RSSI, które były używane do obliczenia metryki, charakteryzują się dużą niestabilnością). Okna czasowe, dla których wartość danej metryki jest wyższa niż określony próg (dla metryk opartych na RSSI i aktywności) lub niższa niż próg (metryka wiarygodności), są traktowane jako niewiarygodne i powinny być odfiltrowane (czerwone krzyżyki na rysunkach).

Można zaobserwować, że wszystkie metryki mogą być używane do oceny wiarygodności określania najbliższej kotwicy. Nawet *metryka RSSI* (Rys. 4.11a), która została obliczona na podstawie silnie niestabilnej siły sygnału Bluetooth, wykazuje dużą zdolność do wykrywania ruchu w trzech z czterech badanych obszarach ruchu (ruch w trzecim obszarze ruchu został przesunięty do obszaru bez ruchu). Pokazuje to, że nawet bez korzystania z odczytów z czujników IMU w smartfonie, można skutecznie wykryć ruch i odfiltrować niewiarygodne predykcje. Wyniki uzyskane dla *metryki aktywności* (oparte na odczytach z IMU, patrz Rysunek 4.11b) są bardziej dokładne i we wszystkich badanych obszarach ruchu, ruch został wykryty (a w obszarach bez ruchu wartości metryki są bliskie zeru, co sugeruje brak aktywności). W przypadku metryki łącznej (patrz Rysunek 4.12), można znacznie poprawić dokładność samej metryki opartej na RSSI i opierać wiedzę na obu źródłach informacji: zmianach w RSSI (informacja zewnętrzna) i czujnikach IMU (informacja wewnętrzna), dostarczając tym sposobem bardziej kompleksowej wiedzy na temat przemieszczania się użytkownika. Takie podejście może potencjalnie le-

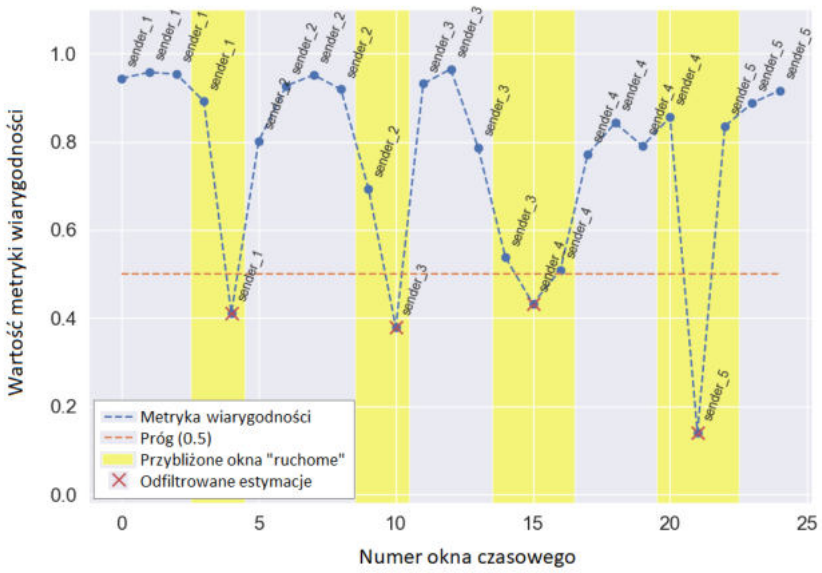



**(a)**




**(b)**

**Rysunek 4.11.** Filtrowanie niewiarygodnych wyników estymacji odległości opartych na proponowanych metrykach podstawowych: (a) metryka aktywności; (b) metryka RSSI.




**Rysunek 4.12.** Filtrowanie niewiarygodnych wyników estymacji odległości oparte na metryce wiarygodności w czasie.

piej działać dla użytkowników niestandardowych, np. takich, którzy chodzą bardzo spokojnie lub takich, którzy stojąc bardzo dynamicznie gestykulują ręką z telefonem (metryka aktywności może zwracać mylące wyniki).

### 4.2.5 Wnioski

Wyniki pokazują, że proponowany system jest skuteczny i może być wykorzystywany w realnych aplikacjach. Dzięki zastosowaniu prostych modeli do detekcji aktywności oraz prostych metod do określania wartości metryki opartej na RSSI, rozwiązanie jest łatwe do implementacji, odporne, efektywne pod względem pamięci i zasobów obliczeniowych oraz wysoce skalowalne. Utworzony system jest elastyczny i może być dostosowany do potrzeb użytkowników: poprzez wybór różnych wartości nieskopoziomowych (podstawa funkcji softmax), a także tych bardziej wysokopoziomowych (różne wartości długości okna czasowego oraz wartości współczynnika ważności metryki aktywności). Metryki prezentowane do filtrowania są proste w interpretacji, nawet dla użytkowników nietechnicznych. Dwie z nich opisują ruch osoby i przyjmują wartości ze stałego zakresu $\langle 0, 1\rangle$ (0 - brak ruchu, 1 - dynamiczny ruch). Trzecia metryka działa w odwrotny sposób - 1 oznacza bardzo wiarygodne wyniki (z powodu braku ruchu osoby z telefonem), a 0 oznacza nie-

wiarygodne wyniki (analizowana osoba najprawdopodobniej chodzi, więc jej lokalizacja dynamicznie zmienia się w czasie). Taka natura metryk umożliwia prostą w interpretacji wizualizację zmian w ich wartościach, co wydaje się niezwykle cenne z perspektywy analitycznej i wykorzystania proponowanych metryk w rzeczywistych systemach.

**Rozwiązanie opisane w niniejszym rozdziale zostało zaprezentowane w artykule [279]** i zaimplementowane w rzeczywistym rozwiązaniu na potrzeby projektu „AIMeet - zastosowanie metod uczenia maszynowego i sztucznej inteligencji na potrzeby lokalizacji indoor i analityki w przemyśle spotkań i innych". Projekt był finansowany w ramach programu Bridge Alfa przez Leonardo Fund ASI Sp. z o.o. i Narodowe Centrum Badań i Rozwoju.

# Rozdział 5

# Podsumowanie

Sztuczna inteligencja (AI) poprawia jakość codziennego życia ludzi. Odgrywa także kluczową rolę w cyfrowej transformacji dzięki możliwościom zautomatyzowanego podejmowania decyzji. Korzyści płynące z tej nowej technologii są znaczące, ale wiążą się z nią równie duże wyzwania. Wyzwania te obejmują między innymi bezpieczeństwo systemów inteligentnych oraz ich wyjaśnialność. W pracy zbadano te aspekty z różnorodnych perspektyw oraz zaproponowano rozwiązania mające na celu rozwiązywanie problemów związanych z tymi tematami.

W obszarze bezpieczeństwa zaproponowano i przedstawiono szereg nowych metod oraz praktycznych spostrzeżeń dotyczących poprawy bezpieczeństwa systemów inteligentnych. Zwrócono uwagę na kompleksową poprawę bezpieczeństwa tego typu systemów. W tym obszarze zaproponowano metody mające na celu wykrywanie podatności oprogramowania oraz ataków sieciowych. Stworzono również metody testowania głębokich sieci neuronowych pod kątem zagrożeń specyficznych dla tego typu algorytmów - ataków adwersarza. Zaproponowane metody pozwoliły uzyskać lepsze wyniki niż testowane w pracy metody referencyjne. W przypadku opisanej w pracy modyfikacji Losowych Sieci Neuronowych, zaproponowane rozwiązanie pozwoliło uzyskać lepszą dokładność wykrywania ataków w stosunku do wersji podstawowej. Wyższa ogólna dokładność była skutkiem znaczącej poprawy czułości modelu. Zastosowanie proponowanej metody selekcji cech dla zadania wykrywania podatności oprogramowania umożliwiło w wielu przypadkach uzyskanie wyższej dokładności klasyfikacji niż w przypadku rozwiązania bez selekcji cech, traktowanego jako rozwiązanie bazowe, przy znacząco mniejszej liczbie cech wykorzystanych w procesie decyzyjnym. Zaprezentowany sposób specjalizacji generycznych modeli uczenia głębokiego również pozwolił na uzyskanie znaczącej poprawy dokładności modeli poprzez zmniejszenie liczby klas między którymi model może się mylić, co pozytywnie wpływa na bezpieczeństwo. Za-

prezentowany nowy typ ataku adwersarza pozwolił uzyskać wyższy poziom szkodliwości próbek niż testowane referencyjne algorytmy ataków. Poziom ten wyrażono za pośrednictwem autorskiej metryki, która może być wykorzystywana do testowania odporności sieci głębokich na ataki adwersarza. Wszystkie zaprezentowane metody można wykorzystać do kompleksowej poprawy bezpieczeństwa systemów inteligentnych.

W celu poprawy wyjaśnialności systemów inteligentnych zaproponowano szereg metod mających na celu poprawę tego aspektu w przypadku istniejących systemów oraz stworzono nowe rozwiązania z jego uwzględnieniem. W przypadku zaprezentowanych metod z zakresu bezpieczeństwa w stosunkowo prosty sposób możliwe było przedstawienie poprawy w stosunku do rozwiązań bazowych w sposób numeryczny. W większości przypadków wykorzystano w tym celu dokładność (ang. accuracy) lub inne metody oceny skuteczności modeli lub ataków przeprowadzanych na nie. Wyjaśnialność oraz jej stopień to natomiast aspekty głównie jakościowe, trudne do uchwycenia w sposób czysto numeryczny. W tym przypadku poprawa może obejmować większą informatywność metod. Przykładem jest zaproponowana metryka oceny szkodliwości ataków adwersarza. Metryka może być wykorzystana również do oceny dokładności modeli na czystych danych. Szczególnie w przypadku modeli o dużej dokładności, może być wykorzystana jako rozszerzenie testowania. W przeciwieństwie do standardowej miary dokładności, proponowana metryka przyporządkowuje różne poziomy w zależności od tego jak daleko w znanej przestrzeni cech znajduje się klasa będąca predykcją w stosunku do klasy prawdziwej. W przypadku proponowanej metryki można również zauważyć poprawę w stosunku do metryki QI-Vis [233]. W tym kontekście poprawa obejmuje wyjaśnialność wyników. Przez to, że metryka DM nie wymaga ustalania żadnych nietrywialnych progów do działania, to jej wyniki są bardziej bezpośrednie w interpretacji przez swoją niezależność od tego typu progów. Poprawa może również obejmować bardziej przejrzyste w interpretacji działanie metody. Przedstawiona metoda inicjalizacji wag dla Losowej Sieci Neuronowej i jej stan przed rozpoczęciem procesu trenowania, mogą być w jednoznaczny sposób zinterpretowane (neutralność sieci w stosunku do danych na początku trenowania) w przeciwieństwie do najczęściej stosowanej metody - inicjalizacji losowej. Kolejnym przykładem poprawy aspektu wyjaśnialności względem dostępnych metod jest fakt, że prezentowana metoda wizualizacji ogólnej aktywacji sieci konwolucyjnej w przeciwieństwie do wizualizacji osobnych map konwolucyjnych zapewnia kompaktowość prezentowanej informacji. Dodatkowo, w przeciwieństwie do większości obecnych w literaturze metod, proponowana metoda jest niezależna od klasyfikatora sieci neuronowej, a więc może być wykorzystana dla każdej sieci z konwolucyjnym ekstraktorem cech.

Prostota i intuicyjność proponowanych metod, a także ich efektywność, sprawiają, że z powodzeniem mogą być one wykorzystane w rzeczywistych systemach informatycznych. Przedstawione w pracy wyniki i porównanie ich z metodami referencyjnymi potwierdzają ich adekwatność do rozwiązywania poszczególnych problemów oraz fakt, że **spełniono cel pracy, którym była poprawa wyjaśnialności i bezpieczeństwa systemów inteligentnych.**

W ramach kontynuacji badań przeprowadzonych w ramach doktoratu planowany jest rozwój metod przedstawionych w niniejszej pracy. W zakresie tradycyjnych zagrożeń bezpieczeństwa obejmujących podatności, planowane jest m.in. wykorzystanie sieci typu Transformer do przewidywania podatności na podstawie kodu źródłowego, jako że sieci te pozwoliły dokonać przełomu w domenie języka naturalnego. Planowane jest również dostosowanie przedstawionych w pracy rozwiązań dedykowanych sieciom konwolucyjnych, tak aby możliwe było ich wykorzystanie przez Transformery wizyjne. Jako że duża część proponowanych rozwiązań dla sieci konwolucyjnych może być wykorzystana z innymi typami sieci wizyjnych, planowane są wszechstronne testy obejmujące sieci do detekcji obiektów i segmentacji semantycznej. Celem jest uzyskanie jak największej ogólności i uniwersalności metod z zakresu bezpieczeństwa i wyjaśnialności.

# Spis skrótów

Poniżej przedstawiono listę najważniejszych skrótów, które wielokrotnie wykorzystano w niniejszej pracy doktorskiej.

- **SDLC** - Cykl Życia Oprogramowania (ang. Software Development Life Cycle)
- **AI** - Sztuczna Inteligencja (ang. Artificial Intelligence)
- **XAI** - Wyjaśnialna Sztuczna Inteligencja (ang. Explainable Artificial Intelligence)
- **IoT** - Internet Rzeczy (ang. Internet of Things)
- **CIA** - Poufność, Integralność, Dostępność do informacji (ang. Confidenciality, Integrity, Availability)
- **CWE** - ang. Common Weakness Enumeration
- **CVE** - ang. Common Vulnerabilities and Exposures
- **ODC** - Ortogonalna Klasyfikacja Defektów (ang. Orthogonal Defect Classification)
- **DoS** - Blokada Usług (ang. Denial of Service)
- **ENISA** - Agencja Unii Europejskiej ds. Cyberbezpieczeństwa (ang. European Union Agency for Cybersecurity)
- **ETSI** - Europejski Instytut Norm Telekomunikacyjnych (ang. European Telecommunications Standards Institute)
- **IEEE** - Instytut Inżynierów Elektryków i Elektroników (ang. Institute of Electrical and Electronics Engineers)
- **FGSM** - ang. Fast Gradient Sign Method
- **PGD** - Projected Gradient Descent
- **AGV** - Pojazd sterowany automatycznie (ang. Automated Guided Vehicle)

- **CNN** - Konwolucyjna Sieć Neuronowa (ang. Convolutional Neural Network)
- **RNN** - Losowa Sieć Neuronowa (ang. Random Neural Network)
- **t-SNE** - Stochastyczna metoda porządkowania sąsiadów w oparciu o rozkład t (ang. t-Distributed Stochastic Neighbor Embedding)
- **ROS** - Robotic Operating System
- **csv** - wartości rozdzielone przecinkiem (ang. comma-separated values)
- **(Signed)NetSat** - (Signed) Network Saturation Attack
- **DM** - Metryka Niepodobieństwa (ang. Dissimilarity Metric)
- **QI-Vis** - średnie wizualne zamieszanie (ang. mean visual confusion)
- **FR** - Wskaźnik Oszukania (ang. Fooling Rate)
- **CSM** - Macierz Podobieństwa Cosinusowego (ang. Cosine Similarity Matrix)
- **SCSM** - Posortowana Macierz Podobieństwa Klas (ang. Sorted Class Similarity Matrix)
- **ILSVRC** - ang. ImageNet Large Scale Visual Recognition Competition
- **NAM** - ang. Network Activation Mapping
- **CAM** - Mapowanie Aktywacji Klas (ang. Class Activation Mapping)
- **Grad-CAM** - Mapowanie Aktywacji Klas Ważone Gradientem (ang. Gradient-weighted Class Activation Mapping)
- **GAP** - ang. Global Average Pooling
- **BLE** - ang. Bluetooth Low Energy
- **RSSI** - Wskaźnik Siły Odbieranego Sygnału (ang. Received Signal Strength Indication)
- **IMU** - Inercyjna Jednostka Pomiarowa (ang. Inertial Measurement Unit)
- **HAR** - Rozpoznawanie Aktywności Ludzkich (ang. Human Activity Recognition)

# Bibliografia


[1] R. Abduljabbar, H. Dia, S. Liyanage, and S. A. Bagloee, "Applications of artificial intelligence in transport: An overview," *Sustainability*, vol. 11, no. 1, p. 189, 2019.

[2] R. S. Peres, X. Jia, J. Lee, K. Sun, A. W. Colombo, and J. Barata, "Industrial Artificial Intelligence in Industry 4.0 - Systematic Review, Challenges and Outlook," *IEEE Access*, vol. 8, pp. 220121–220139, 2020.

[3] S. Secinaro, D. Calandra, A. Secinaro, V. Muthurangu, and P. Biancone, "The role of artificial intelligence in healthcare: a structured literature review," *BMC Medical Informatics and Decision Making*, vol. 21, pp. 1–23, 2021.

[4] K. Jha, A. Doshi, P. Patel, and M. Shah, "A comprehensive review on automation in agriculture using artificial intelligence," *Artificial Intelligence in Agriculture*, vol. 2, pp. 1–12, 2019.

[5] Z. Ullah, F. Al-Turjman, L. Mostarda, and R. Gagliardi, "Applications of artificial intelligence and machine learning in smart cities," *Computer Communications*, vol. 154, pp. 313–323, 2020.

[6] **Filus, Katarzyna**, A. Domański, J. Domańska, D. Marek, and J. Szyguła, "**Long-Range Dependent Traffic Classification with Convolutional Neural Networks Based on Hurst Exponent Analysis**," *Entropy*, vol. 22, no. 10, p. 1159, 2020.

[7] P. Nineta and P. Isabel, "A multilayer framework for good cybersecurity practices for AI," tech. rep., European Union Agency for Cybersecurity (ENISA), 2023.

[8] M. Hamdi and H. Abie, "Game-based adaptive security in the Internet of Things for eHealth," in *IEEE International Conference on Communications (ICC)*, pp. 920–925, 2014.

[9] M. Krichen, W. Y. H. Adoni, A. Mihoub, M. Y. Alzahrani, and T. Nahhal, "Security challenges for drone communications: Possible threats, attacks and countermeasures," in *2nd International Conference of Smart Systems and Emerging Technologies (SMARTTECH)*, pp. 184–189, 2022.

[10] R. Nandhakumar and S. Mohanapriya, "Smart Baby Monitoring System using YOLO V7 Algorithm," in *International Conference on Information Technology Research and Innovation (ICITRI)*, pp. 42–47, 2022.

[11] M. Stoyanova, Y. Nikoloudakis, S. Panagiotakis, E. Pallis, and E. K. Markakis, "A Survey on the Internet of Things (IoT) Forensics: Challenges, Approaches and Open Issues," *IEEE Communications Surveys & Tutorials*, 2020.

[12] Veracode, "State of Software Security," tech. rep., Veracode, 2020.

[13] "CWE Top 25 Most Dangerous Software Weaknesses." [online], 2022. `https://cwe.mitre.org/top25/archive/2022/2022_cwe_top25.html` [Dostęp: 2023-06-18].

[14] J. G. Almaraz-Rivera, J. A. Perez-Diaz, and J. A. Cantoral-Ceballos, "Transport and Application Layer DDoS Attacks Detection to IoT Devices by Using Machine Learning and Deep Learning Models," *Sensors*, vol. 22, no. 9, p. 3367, 2022.

[15] M. A. Ferrag and L. Maglaras, "DeepCoin: A Novel Deep Learning and Blockchain-Based Energy Exchange Framework for Smart Grids," *IEEE Transactions on Engineering Management*, 2019.

[16] Z. Li, D. Zou, S. Xu, X. Ou, H. Jin, S. Wang, Z. Deng, and Y. Zhong, "VulDeePecker: A Deep Learning-Based System for Vulnerability Detection," 2018.

[17] M. Mittal, K. Kumar, and S. Behal, "Deep learning approaches for detecting DDoS attacks: A systematic review," *Soft computing*, pp. 1–37, 2022.

[18] B. Caroline, B. Christian, B. Stephan, B. Luis, , D. Giuseppe, D. Ernesto, H. Sven, L. Caroline, M. Jochen, N. Duy Cu, P. Nineta, P. Isabel, S. George, S. Vincent, and S. Ewelina, "Artificial Intelligence Cybersecurity Challenges," tech. rep., European Union Agency for Cybersecurity (ENISA), 2020.

[19] N. Stavros, G. Misuraca, and R. Pierre, "Artificial Intelligence and Cybersecurity Research," tech. rep., The European Union Agency for Cybersecurity (ENISA), 2023.

[20] I. J. Goodfellow, J. Shlens, and C. Szegedy, "Explaining and Harnessing Adversarial Examples," *arXiv preprint arXiv:1412.6572*, 2014.

[21] C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. Goodfellow, and R. Fergus, "Intriguing properties of neural networks," *2nd International Conference on Learning Representations*, 2013.

[22] D. Georgia, H. Ronan, J. Henrik, N. Rossen, M. Apostolos, and S. Ignacio, "Cybersecurity Challenges in the Uptake of Artificial Intelligence in Autonomous Driving," tech. rep., The European Union Agency for Cybersecurity (ENISA), 2021.

[23] "Sztuczna inteligencja: co to jest i jakie ma zastosowania?." [online]. Dostępne: `https://www.europarl.europa.eu/news/pl/headlines/society/20200827STO85804/sztuczna-inteligencja-co-to-jest-i-jakie-ma-zastosowania` [Dostęp: 2023-04-07].

[24] S. J. Russell, *Artificial Intelligence: A Modern Approach*. Pearson Education, Inc., 2010.

[25] L. Deng, "The MNIST Database of Handwritten Digit Images for Machine Learning Research," *IEEE Signal Processing Magazine*, vol. 29, no. 6, pp. 141–142, 2012.

[26] B. Wu, C. Xu, X. Dai, A. Wan, P. Zhang, Z. Yan, M. Tomizuka, J. Gonzalez, K. Keutzer, and P. Vajda, "Visual transformers: Token-based image representation and processing for computer vision," *arXiv preprint arXiv:2006.03677*, 2020.

[27] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.

[28] K. He, X. Zhang, S. Ren, and J. Sun, "Deep Residual Learning for Image Recognition," in *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 770–778, 2016.

[29] M. Tan and Q. Le, "EfficientNetV2: Smaller Models and Faster Training," in *International Conference on Machine Learning*, pp. 10096–10106, 2021.

[30] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," *Advances in Neural Information Processing Systems*, vol. 30, 2017.

[31] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[32] K. Cho, B. Van Merriënboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio, "Learning phrase representations using RNN encoder-decoder for statistical machine translation," *arXiv preprint arXiv:1406.1078*, 2014.

[33] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly, *et al.*, "An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale," *arXiv preprint arXiv:2010.11929*, 2020.

[34] S. Mehta and M. Rastegari, "MobileViT: Light-weight, General-purpose, and Mobile-friendly Vision Transformer," in *International Conference on Learning Representations*.

[35] S. Khan, M. Naseer, M. Hayat, S. W. Zamir, F. S. Khan, and M. Shah, "Transformers in Vision: A Survey," *ACM computing surveys*, vol. 54, no. 10s, pp. 1–41, 2022.

[36] Z. Liu, Y. Lin, Y. Cao, H. Hu, Y. Wei, Z. Zhang, S. Lin, and B. Guo, "Swin Transformer: Hierarchical Vision Transformer using Shifted Windows," in *IEEE/CVF International Conference on Computer Vision (ICCV)*, pp. 10012–10022, 2021.

[37] H. Wu, B. Xiao, N. Codella, M. Liu, X. Dai, L. Yuan, and L. Zhang, "CvT: Introducing Convolutions to Vision Transformers," in *IEEE/CVF International Conference on Computer Vision (ICCV)*, pp. 22–31, 2021.

[38] B. Graham, A. El-Nouby, H. Touvron, P. Stock, A. Joulin, H. Jégou, and M. Douze, "LeViT: a Vision Transformer in ConvNet's Clothing for Faster Inference," in *IEEE/CVF International Conference on Computer Vision (ICCV)*, pp. 12259–12269, 2021.

[39] Z. Liu, H. Mao, C.-Y. Wu, C. Feichtenhofer, T. Darrell, and S. Xie, "A ConvNet for the 2020s," in *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 11976–11986, 2022.

[40] E. Gelenbe, "Random Neural Networks with Negative and Positive Signals and Product Form Solution," *Neural Computation*, vol. 1, no. 4, pp. 502–510, 1989.

[41] E. Gelenbe, Y. Feng, and K. R. R. Krishnan, "Neural network methods for volumetric magnetic resonance imaging of the human brain," *Proceedings of the IEEE*, vol. 84, no. 10, pp. 1488–1496, 1996.

[42] K. Hussain and G. Moussa, "On-road vehicle classification based on random neural network and bag-of-visual words," *Probability in the Engineering and Informational Sciences*, vol. 30, no. 3, pp. 403–412, 2016.

[43] K. F. Hussain, E. Radwan, and G. S. Moussa, "Augmented Reality Experiment: Drivers' Behavior at an Unsignalized Intersection," *IEEE Transactions on Intelligent Transportation Systems*, vol. 14, no. 2, pp. 608–617, 2013.

[44] Y. Yin, L. Wang, and E. Gelenbe, "Multi-Layer Neural Networks for Quality of Service oriented Server-State Classification in Cloud Servers," in *IEEE International Joint Conference on Neural Networks (IJCNN)*, pp. 1623–1627, 2017.

[45] A. Saeed, A. Ahmadinia, A. Javed, and H. Larijani, "Intelligent Intrusion Detection in Low-Power IoTs," *ACM Transactions on Internet Technology (TOIT)*, vol. 16, no. 4, pp. 1–25, 2016.

[46] O. Brun, Y. Yin, and E. Gelenbe, "Deep Learning with Dense Random Neural Network for Detecting Attacks against IoT-connected Home Environments," *Procedia Computer Science*, vol. 134, pp. 458 – 463, 2018.

[47] K. Hussain, M. Yousef, and E. Gelenbe, "Accurate and Energy-Efficient Classification with Spiking Random Neural Network," *Probability in the Engineering and Informational Sciences*, p. 1–11, 2019.

[48] Y. Yin and E. Gelenbe, "A Classifier Based on Spiking Random Neural Network Function Approximator," 2018.

[49] Y. Yin, "Deep Learning with the Random Neural Network and its Applications," *ArXiv*, vol. abs/1810.08653, 2018.

[50] E. Gelenbe, "Learning in the Recurrent Random Neural Network," *Neural Computation*, vol. 5, pp. 154–164, 1993.

[51] S. Timotheou, "A novel weight initialization method for the random neural network," *Neurocomputing*, vol. 73, no. 1, pp. 160 – 168, 2009.

[52] S. Timotheou, "The Random Neural Network: A Survey," *The Computer Journal*, vol. 53, no. 3, pp. 251–267, 2010.

[53] S. Basterrech, S. Mohammed, G. Rubino, and M. Soliman, "Levenberg—Marquardt training algorithms for random neural networks," *The computer journal*, vol. 54, no. 1, pp. 125–135, 2011.

[54] O. Biran and C. Cotton, "Explanation and justification in machine learning: A survey," in *IJCAI-17 workshop on explainable AI (XAI)*, vol. 8, pp. 8–13, 2017.

[55] B. Kim, R. Khanna, and O. O. Koyejo, "Examples are not enough, learn to criticize! Criticism for interpretability," in *Advances in Neural Information Processing Systems*, vol. 29, 2016.

[56] M. Amrani and F. Jiang, "Deep feature extraction and combination for synthetic aperture radar target classification," *Journal of Applied Remote Sensing*, vol. 11, no. 4, p. 042616, 2017.

[57] J. T. Springenberg, A. Dosovitskiy, T. Brox, and M. Riedmiller, "Striving for Simplicity: The All Convolutional Net," *arXiv preprint arXiv:1412.6806*, 2014.

[58] B. Zhou, A. Khosla, A. Lapedriza, A. Oliva, and A. Torralba, "Learning Deep Features for Discriminative Localization," in *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 2921–2929, 2016.

[59] R. R. Selvaraju, A. Das, R. Vedantam, M. Cogswell, D. Parikh, and D. Batra, "Grad-CAM: Why did you say that?," *arXiv preprint arXiv:1611.07450*, 2016.

[60] M. B. Muhammad and M. Yeasin, "Eigen-CAM: Class Activation Map using Principal Components," in *IEEE International Joint Conference on Neural Networks (IJCNN)*, pp. 1–7, 2020.

[61] P.-T. Jiang, C.-B. Zhang, Q. Hou, M.-M. Cheng, and Y. Wei, "LayerCAM: Exploring Hierarchical Class Activation Maps for Localization," *IEEE Transactions on Image Processing*, vol. 30, pp. 5875–5888, 2021.

[62] B. N. Patro, M. Lunayach, and V. P. Namboodiri, "Uncertainty Class Activation Map (U-CAM) Using Gradient Certainty Method," *IEEE Transactions on Image Processing*, vol. 30, pp. 1910–1924, 2021.

[63] S. Belharbi, A. Sarraf, M. Pedersoli, I. Ben Ayed, L. McCaffrey, and E. Granger, "F-CAM: Full Resolution Class Activation Maps via Guided Parametric Upscaling," in *IEEE/CVF Winter Conference on Applications of Computer Vision (WACV)*, pp. 3490–3499, 2022.

[64] X. Glorot and Y. Bengio, "Understanding the difficulty of training deep feedforward neural networks," in *International Conference on Artificial Intelligence and Statistics (AISTATS)*, pp. 249–256, JMLR Workshop and Conference Proceedings, 2010.

[65] C. Manresa-Yee and S. Ramis, "Assessing gender bias in predictive algorithms using explainable AI," in *Proceedings of the XXI International Conference on Human Computer Interaction*, pp. 1–8, 2021.

[66] Enisa, "ENISA Threat Landscape 2020 - List of top 15 threats," tech. rep., European Union Agency for Cybersecurity (ENISA), 2020.

[67] I. Ahmad, M. S. Niazy, R. A. Ziar, and S. Khan, "Survey on IoT: Security Threats and Applications," *Journal of Robotics and Control (JRC)*, vol. 2, no. 1, pp. 42–46, 2020.

[68] J. Sengupta, S. Ruj, and S. D. Bit, "A comprehensive survey on attacks, security issues and blockchain solutions for IoT and IIoT," *Journal of Network and Computer Applications*, vol. 149, p. 102481, 2020.

[69] "What is a SYN flood attack." [online]. Dostępne: `https://www.imperva.com/learn/ddos/syn-flood/` [Dostęp: 2021-04-07].

[70] "What is a UDP flood attack." [online]. Dostępne: `https://www.imperva.com/learn/ddos/udp-flood/` [Dostęp: 2021-04-07].

[71] "What is an HTTP flood attack." [online]. Dostępne: `https://www.imperva.com/learn/application-security/http-flood/` [Dostęp: 2021-04-07].

[72] "What is Slowloris?." [online]. Dostępne: `https://www.imperva.com/learn/ddos/slowloris/` [Dostęp: 2021-04-07].

[73] F. M. Cortés and N. G. Gómez, "A hybrid alarm management strategy in signature-based intrusion detection systems," in *2019 IEEE Colombian Conference on Communications and Computing (COLCOM)*, pp. 1–6, 2019.

[74] W. Li, S. Tug, W. Meng, and Y. Wang, "Designing collaborative blockchained signature-based intrusion detection in IoT environments," *Future Generation Computer Systems*, vol. 96, pp. 481–489, 2019.

[75] A. Guerra-Manzanares, J. Medina-Galindo, H. Bahsi, and S. Nõmm, "MedBIoT: Generation of an IoT Botnet Dataset in a Medium-sized IoT Network," in *6th International Conference on Information Systems Security and Privacy*, pp. 207–218, 2020.

[76] A. Aldribi, I. Traoré, B. Moa, and O. Nwamuo, "Hypervisor-based cloud intrusion detection through online multivariate statistical change tracking," *Computers & Security*, vol. 88, p. 101646, 2020.

[77] C. Xiang, P. C. Yong, and L. S. Meng, "Design of multiple-level hybrid classifier for intrusion detection system using bayesian clustering and decision trees," *Pattern Recognition Letters*, vol. 29, no. 7, pp. 918 – 924, 2008.

[78] T. Shon, X. Kovah, and J. Moon, "Applying genetic algorithm for classifying anomalous TCP/IP packets," *Neurocomputing*, vol. 69, no. 16, pp. 2429 – 2433, 2006.

[79] M. Ge, X. Fu, N. Syed, Z. Baig, G. Teo, and A. Robles-Kelly, "Deep Learning-Based Intrusion Detection for IoT Networks," in *2019 IEEE 24th Pacific Rim International Symposium on Dependable Computing (PRDC)*, pp. 256–25609, 2019.

[80] C. Yin, Y. Zhu, J. Fei, and X. He, "A Deep Learning Approach for Intrusion Detection Using Recurrent Neural Networks," *IEEE Access*, vol. 5, pp. 21954–21961, 2017.

[81] "Information technology — Security techniques — Information security management systems — Overview and vocabulary," standard, International Organization for Standardization, Geneva, CH, Feb. 2018.

[82] Z. Zhioua, S. Short, and Y. Roudier, "Static Code Analysis for Software Security Verification," in *IEEE 38th International Computer Software and Applications Conference Workshops*, pp. 102–109, 2014.

[83] Y. Shin, A. Meneely, L. Williams, and J. A. Osborne, "Evaluating Complexity, Code Churn, and Developer Activity Metrics as Indicators of Software Vulnerabilities," *IEEE transactions on software engineering*, vol. 37, no. 6, pp. 772–787, 2010.

[84] "CWE - Common Weakness Enumeration." [online]. `https://cwe.mitre.org/index.html` [Dostęp: 2023-06-18].

[85] "CVE." [online]. `https://cve.mitre.org/` [Dostęp: 2023-06-18].

[86] "CWE-119: Improper Restriction of Operations within the Bounds of a Memory Buffer." [online]. `https://cwe.mitre.org/data/definitions/119.html` [Dostęp: 2023-06-18].

[87] "CVE-2022-26763." [online]. `https://www.cvedetails.com/cve/CVE-2022-26763/` [Dostęp: 2023-06-18].

[88] "Computer Emergency Response Team Coordination Center." [online]. `https://www.kb.cert.org/vuls/` [Dostęp: 2023-06-18].

[89] "Open Web Application Security Project (OWASP)." [online]. `https://owasp.org/` [Dostęp: 2023-06-18].

[90] "Information Security Training - SANS Cyber Security Certifications & Research." [online]. `https://www.sans.org/` [Dostęp: 2023-06-18].

[91] "National Vulnerability Database (NVD)." [online]. `https://nvd.nist.gov/` [Dostęp: 2023-06-18].

[92] "CVE Details." [online]. `https://www.cvedetails.com/` [Dostęp: 2023-06-18].

[93] "OWASP Top Ten." [online]. `https://owasp.org/www-project-top-ten/` [Dostęp: 2023-06-18].

[94] "OWASP Secure Coding Practices Quick Reference Guide." [online]. `https://owasp.org/www-pdf-archive/OWASP_SCP_Quick_Reference_Guide_v1.pdf` [Dostęp: 2023-06-18].

[95] Veracode, "State of Software Security," tech. rep., Veracode, 2018.

[96] S. M. Ghaffarian and H. R. Shahriari, "Software Vulnerability Analysis and Discovery Using Machine-Learning and Data-Mining Techniques: A Survey," *ACM Computing Surveys (CSUR)*, vol. 50, no. 4, pp. 1–36, 2017.

[97] Veracode, "State of Software Security," tech. rep., Veracode, 2016.

[98] S. Kwon, S. Park, H. Cho, Y. Park, D. Kim, and K. Yim, "Towards 5G-based IoT security analysis against Vo5G eavesdropping," *Computing*, pp. 1–23.

[99] O. A. Wahab, J. Bentahar, H. Otrok, and A. Mourad, "Resource-Aware Detection and Defense System against Multi-Type Attacks in the Cloud: Repeated Bayesian Stackelberg Game," *IEEE Transactions on Dependable and Secure Computing*, 2019.

[100] R. Croft, D. Newlands, Z. Chen, and M. A. Babar, "An Empirical Study of Rule-Based and Learning-Based Approaches for Static Application Security Testing," in *ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*, pp. 1–12, 2021.

[101] A. Kaur and R. Nayyar, "A comparative study of static code analysis tools for vulnerability detection in C/C++ and JAVA source code," *Procedia Computer Science*, vol. 171, pp. 2023–2029, 2020.

[102] T. Riom, A. Sawadogo, K. Allix, T. F. Bissyand'e, N. Moha, and J. Klein, "Revisiting the VCCFinder approach for the identification of vulnerability-contributing commits," *Empirical Software Engineering*, vol. 26, no. 3, pp. 1–30, 2021.

[103] J. Zhao, S. Guo, and D. Mu, "DouBiGRU-A: Software defect detection algorithm based on attention mechanism and double BiGRU," *Computers & Security*, vol. 111, p. 102459, 2021.

[104] X. Chen, Z. Yuan, Z. Cui, D. Zhang, and X. Ju, "Empirical studies on the impact of filter-based ranking feature selection on security vulnerability prediction," *IET Software*.

[105] X. Chen, Y. Zhao, Z. Cui, G. Meng, Y. Liu, and Z. Wang, "Large-Scale Empirical Studies on Effort-Aware Security Vulnerability Prediction Methods," *IEEE Transactions on Reliability*, vol. 69, no. 1, pp. 70–87, 2019.

[106] J. Walden, J. Stuckman, and R. Scandariato, "Predicting vulnerable components: Software metrics vs text mining," in *IEEE 25th international symposium on software reliability engineering*, pp. 23–33, 2014.

[107] J. Cui, L. Wang, X. Zhao, and H. Zhang, "Towards predictive analysis of android vulnerability using statistical codes and machine learning for iot applications," *Computer Communications*, 2020.

[108] Y. Fang, S. Han, C. Huang, and R. Wu, "TAP: A static analysis model for PHP vulnerabilities based on token and deep learning technology," *PloS one*, vol. 14, no. 11, p. e0225196, 2019.

[109] Z. Li, D. Zou, S. Xu, H. Jin, Y. Zhu, and Z. Chen, "SySeVR: A framework for using deep learning to detect software vulnerabilities," *IEEE Transactions on Dependable and Secure Computing*, 2021.

[110] A. Pechenkin and R. Demidov, "Applying Deep Learning and Vector Representation for Software Vulnerabilities Detection," in *International Conference on Security of Information and Networks*, pp. 1–6, 2018.

[111] Q. Xiao, K. Li, D. Zhang, and W. Xu, "Security Risks in Deep Learning Implementations," in *IEEE Security and Privacy Workshops (SPW)*, pp. 123–128, 2018.

[112] J. D. Pereira, N. Ivaki, and M. Vieira, "Characterizing Buffer Overflow Vulnerabilities in Large C/C++ Projects," *IEEE Access*, vol. 9, pp. 142879–142892, 2021.

[113] M. Siavvas, E. Gelenbe, D. Kehagias, and D. Tzovaras, "Static Analysis-Based Approaches for Secure Software Development," in *International ISCIS Security Workshop*, pp. 142–157, 2018.

[114] H. Alves, B. Fonseca, and N. Antunes, "Software Metrics and Security Vulnerabilities: Dataset and Exploratory Study," in *2016 12th European Dependable Computing Conference (EDCC)*, pp. 37–44, 2016.

[115] S. Moshtari, A. Sami, and M. Azimi, "Using complexity metrics to improve software security," *Computer Fraud & Security*, vol. 2013, no. 5, pp. 8–17, 2013.

[116] I. Chowdhury and M. Zulkernine, "Using complexity, coupling, and cohesion metrics as early indicators of vulnerabilities," *Journal of Systems Architecture*, vol. 57, no. 3, pp. 294–313, 2011.

[117] M. Sherriff, S. S. Heckman, M. Lake, and L. Williams, "Identifying fault-prone files using static analysis alerts through singular value decomposition," in *Conference of the Center for Advanced Studies on Collaborative Research (CASCON)*, pp. 276–279, 2007.

[118] M. Gegick and L. Williams, "Toward the Use of Automated Static Analysis Alerts for Early Identification of Vulnerability- and Attack-prone Components," in *Second International Conference on Internet Monitoring and Protection (ICIMP 2007)*, pp. 18–18, 2007.

[119] S. Neuhaus, T. Zimmermann, C. Holler, and A. Zeller, "Predicting vulnerable software components," in *ACM Computer and Communications Security Conference (CCS)*, pp. 529–540, 2007.

[120] M. Jimenez, M. Papadakis, and Y. Le Traon, "Vulnerability Prediction Models: A Case Study on the Linux Kernel," in *IEEE 16th International Working Conference on Source Code Analysis and Manipulation (SCAM)*, pp. 1–10, 2016.

[121] R. Scandariato, J. Walden, A. Hovsepyan, and W. Joosen, "Predicting Vulnerable Software Components via Text Mining," *IEEE Transactions on Software Engineering*, vol. 40, no. 10, pp. 993–1006, 2014.

[122] P. K. Kudjo, J. Chen, M. Zhou, S. Mensah, and R. Huang, "Improving the Accuracy of Vulnerability Report Classification Using Term Frequency-Inverse Gravity Moment," in *2019 IEEE 19th International Conference on Software Quality, Reliability and Security (QRS)*, pp. 248–259, 2019.

[123] Y. Zhang, D. Lo, X. Xia, B. Xu, J. Sun, and S. Li, "Combining Software Metrics and Text Features for Vulnerable File Prediction," in *2015 20th International Conference on Engineering of Complex Computer Systems (ICECCS)*, pp. 40–49, 2015.

[124] X. Du, B. Chen, Y. Li, J. Guo, Y. Zhou, Y. Liu, and Y. Jiang, "Leopard: Identifying vulnerable code for vulnerability assessment through program metrics," in *2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE)*, pp. 60–71, 2019.

[125] M. Zhang, X. d. C. de Carnavalet, L. Wang, and A. Ragab, "Large-Scale Empirical Study of Important Features Indicative of Discovered Vulnerabilities to Assess Application Security," *IEEE Transactions on Information Forensics and Security*, vol. 14, no. 9, pp. 2315–2330, 2019.

[126] I. Chowdhury and M. Zulkernine, "Can complexity, coupling, and cohesion metrics be used as early indicators of vulnerabilities?," in *ACM Symposium on Applied Computing*, pp. 1963–1969, 2010.

[127] H. Hanif and S. Maffeis, "VulBERTa: Simplified Source Code Pre-Training for Vulnerability Detection," in *IEEE International Joint Conference on Neural Networks (IJCNN)*, pp. 1–8, 2022.

[128] K. A. Jackson and B. T. Bennett, "Locating SQL injection vulnerabilities in Java byte code using natural language techniques," in *SoutheastCon 2018*, pp. 1–5, 2018.

[129] K. W. Nafi, B. Roy, C. K. Roy, and K. A. Schneider, "A universal cross language software similarity detector for open source software categorization," *Journal of Systems and Software*, vol. 162, p. 110491, 2020.

[130] Y. Pang, X. Xue, and H. Wang, "Predicting Vulnerable Software Components through Deep Neural Network," in *International Conference on Deep Learning Technologies*, pp. 6–10, 2017.

[131] A. Arusoaie, S. Ciobâca, V. Craciun, D. Gavrilut, and D. Lucanu, "A Comparison of Open-Source Static Analysis Tools for Vulnerability Detection in C/C++ Code," in *2017 19th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing (SYNASC)*, pp. 161–168, 2017.

[132] S. Shiraishi, V. Mohan, and H. Marimuthu, "Test Suites for Benchmarks of Static Analysis Tools," in *IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW)*, pp. 12–15, 2015.

[133] Y. Nong, H. Cai, P. Ye, L. Li, and F. Chen, "Evaluating and comparing memory error vulnerability detectors," *Information and Software Technology*, vol. 137, p. 106614, 2021.

[134] L. Khaled and N. Abdelbaki, "Evaluation of Software Static Analyzers," in *International Conference on Software and Information Engineering (ICSIE)*, pp. 11–17, 2020.

[135] A. Schwarzschild, M. Goldblum, A. Gupta, J. P. Dickerson, and T. Goldstein, "Just How Toxic is Data Poisoning? A Unified Benchmark for Backdoor and Data Poisoning Attacks," in *International Conference on Machine Learning*, pp. 9389–9398, PMLR, 2021.

[136] A. Madry, A. Makelov, L. Schmidt, D. Tsipras, and A. Vladu, "Towards Deep Learning Models Resistant to Adversarial Attacks," *arXiv preprint arXiv:1706.06083*, 2017.

[137] N. Carlini and D. Wagner, "Towards Evaluating the Robustness of Neural Networks," in *IEEE Symposium on Security and Privacy*, pp. 39–57, 2017.

[138] J. Chen, M. I. Jordan, and M. J. Wainwright, "Hopskipjumpattack: A query-efficient decision-based attack," in *IEEE Symposium on Security and Privacy (SP)*, pp. 1277–1294, 2020.

[139] H. Li, X. Xu, X. Zhang, S. Yang, and B. Li, "Qeba: Query-efficient boundary-based blackbox attack," in *IEEE/CVF conference on computer vision and pattern recognition*, pp. 1221–1230, 2020.

[140] A. Rahmati, S.-M. Moosavi-Dezfooli, P. Frossard, and H. Dai, "Geoda: a geometric framework for black-box adversarial attacks," in *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 8446–8455, 2020.

[141] W. Zhou, X. Hou, Y. Chen, M. Tang, X. Huang, X. Gan, and Y. Yang, "Transferable Adversarial Perturbations," in *European Conference on Computer Vision*, pp. 452–467, 2018.

[142] J. Byun, S. Cho, M.-J. Kwon, H.-S. Kim, and C. Kim, "Improving the Transferability of Targeted Adversarial Examples through Object-Based Diverse Input," in *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 15244–15253, 2022.

[143] S.-M. Moosavi-Dezfooli, A. Fawzi, O. Fawzi, and P. Frossard, "Universal adversarial perturbations," in *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 1765–1773, 2017.

[144] K. R. Mopuri, A. Ganeshan, and R. V. Babu, "Generalizable Data-free Objective for Crafting Universal Adversarial Perturbations," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 41, no. 10, pp. 2452–2465, 2018.

[145] Z. Zhang and T. Wu, "Learning Ordered Top-k Adversarial Attacks via Adversarial Distillation," in *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*, pp. 776–777, 2020.

[146] A. Kurakin, I. J. Goodfellow, and S. Bengio, "Adversarial examples in the physical world," in *Artificial intelligence safety and security*, pp. 99–112, Chapman and Hall/CRC, 2018.

[147] Y. Dong, F. Liao, T. Pang, H. Su, J. Zhu, X. Hu, and J. Li, "Boosting Adversarial Attacks with Momentum," in *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 9185–9193, 2018.

[148] A. Ganeshan, V. BS, and R. V. Babu, "FDA: Feature Disruptive Attack," in *IEEE/CVF International Conference on Computer Vision*, pp. 8069–8079, 2019.

[149] S. Sabour, Y. Cao, F. Faghri, and D. J. Fleet, "Adversarial Manipulation of Deep Representations," *arXiv preprint arXiv:1511.05122*, 2015.

[150] S. Lu, B. Nott, A. Olson, A. Todeschini, H. Vahabi, Y. Carmon, and L. Schmidt, "Harder or Different? A Closer Look at Distribution Shift in Dataset Reproduction," in *ICML Workshop on Uncertainty and Robustness in Deep Learning*, 2020.

[151] M. G. Müller, M. Durner, A. Gawel, W. Stürzl, R. Triebel, and R. Siegwart, "A Photorealistic Terrain Simulation Pipeline for Unstructured Outdoor Environments," in *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 9765–9772, 2021.

[152] F. J. Romero-Ramirez, R. Muñoz-Salinas, and R. Medina-Carnicer, "Speeded up detection of squared fiducial markers," *Image and vision Computing*, vol. 76, pp. 38–47, 2018.

[153] S. Garrido-Jurado, R. Muñoz-Salinas, F. J. Madrid-Cuevas, and M. J. Marín-Jiménez, "Automatic generation and detection of highly reliable fiducial markers under occlusion," *Pattern Recognition*, vol. 47, no. 6, pp. 2280–2292, 2014.

[154] E. Olson, "AprilTag: A robust and flexible visual fiducial system," in *IEEE international conference on robotics and automation*, pp. 3400–3407, 2011.

[155] M. Fiala, "ARTag, a fiducial marker system using digital techniques," in *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, vol. 2, pp. 590–596, 2005.

[156] T. Kiyokawa, K. Tomochika, J. Takamatsu, and T. Ogasawara, "Fully Automated Annotation With Noise-Masked Visual Markers for Deep-Learning-Based Object Detection," *IEEE Robotics and Automation Letters*, vol. 4, no. 2, pp. 1972–1977, 2019.

[157] T. Kiyokawa, K. Tomochika, J. Takamatsu, and T. Ogasawara, "Efficient collection and automatic annotation of real-world object images by taking advantage of post-diminished multiple visual markers," *Advanced Robotics*, vol. 33, no. 24, pp. 1264–1280, 2019.

[158] O. Abu Daoud, O. Albatayneh, L. Forslof, and K. Ksaibati, "Validating the practicality of utilising an image classifier developed using tensorflow framework in collecting corrugation data from gravel roads," *International Journal of Pavement Engineering*, pp. 1–12, 2021.

[159] B. Pang, E. Nijkamp, and Y. N. Wu, "Deep learning with tensorflow: A review," *Journal of Educational and Behavioral Statistics*, vol. 45, no. 2, pp. 227–248, 2020.

[160] "TensorFlow." [online]. Available: `https://www.tensorflow.org/` [Accessed: 2021-11-29].

[161] A. Corallo, M. Lazoi, and M. Lezzi, "Cybersecurity in the context of industry 4.0: A structured classification of critical assets and business impacts," *Computers in Industry*, vol. 114, p. 103165, 2020.

[162] R. Chillarege, I. S. Bhandari, J. K. Chaar, M. J. Halliday, D. S. Moebus, B. K. Ray, and M.-Y. Wong, "Orthogonal defect classification-a concept for in-process measurements," *IEEE Transactions on software Engineering*, vol. 18, no. 11, pp. 943–956, 1992.

[163] S. Samonas and D. Coss, "The CIA strikes back: Redefining confidentiality, integrity and availability in security," *Journal of Information System Security*, vol. 10, no. 3, 2014.

[164] D. Toloudis, G. Spanos, and L. Angelis, "Associating the Severity of Vulnerabilities with their Description," in *International Conference on Advanced Information Systems Engineering*, pp. 231–242, Springer, 2016.

[165] N. Bridge and C. Miller, "Orthogonal Defect Classification Using Defect Data to Improve Software Development," *Software Quality*, vol. 3, no. 1, pp. 1–8, 1998.

[166] "CppCheck." [online]. `https://cppcheck.sourceforge.io/` [Dostęp: 2023-06-20].

[167] "FlawFinder." [online]. `https://dwheeler.com/flawfinder/` [Dostęp: 2023-06-20].

[168] "Visual Code Grepper." [online]. https://github.com/nccgroup/VCG [Dostęp: 2023-06-20].

[169] "Commit fixing CVE-2020-26267 vulnerability." [online]. https://github.com/tensorflow/tensorflow/commit/ebc70b7a592420d3d2f359e4b1694c236b82c7ae [Dostęp: 2021-11-29].

[170] "Commit fixing CVE-2021-37637 vulnerability." [online]. https://github.com/tensorflow/tensorflow/commit/5dc7f6981fdaf74c8c5be41f393df705841fb7c5 [Dostęp: 2021-11-29].

[171] "Commit fixing CVE-2021-29566 vulnerability." [online]. https://github.com/tensorflow/tensorflow/commit/3f6fe4dfef6f57e768260b48166c27d148f3015f [Dostęp: 2021-11-29].

[172] **Filus, Katarzyna** and J. Domańska, "**Software vulnerabilities in TensorFlow-based deep learning applications**," *Computers & Security*, vol. 124, p. 102948, 2023.

[173] H. Assal and S. Chiasson, "'Think secure from the beginning' A Survey with Software Developers," in *CHI Conference on Human Factors in Computing Systems*, pp. 1–13, 2019.

[174] B. Chess and J. West, *Secure Programming with Static Analysis*. Pearson Education, 2007.

[175] Z. P. Reynolds, A. B. Jayanth, U. Koc, A. A. Porter, R. R. Raje, and J. H. Hill, "Identifying and Documenting False Positive Patterns Generated by Static Code Analysis Tools," in *IEEE/ACM 4th International Workshop on Software Engineering Research and Industrial Practice (SER&IP)*, pp. 55–61, 2017.

[176] "Visual Studio IDE, Code Editor, Azure DevOps, & App Center - Visual Studio." [online]. Available: https://visualstudio.microsoft.com/ [Accessed: 2020-08-05].

[177] "IntelliJ IDEA: The Java IDE for Professional Developers by JetBrains." [online]. Available: https://www.jetbrains.com/idea/ [Accessed: 2020-08-05].

[178] "Veracode." [online]. Available: https://www.veracode.com/ [Accessed: 2020-08-05].

[179] "SonarQube." [online]. Available: https://www.sonarqube.org/ [Accessed: 2020-12-03].

[180] "CCCC - C and C++ Code Counter." [online]. Available: http://sarnold.github.io/cccc/CCCC_User_Guide.html [Accessed: 2020-12-03].

[181] "User Guide for CCCC." [online]. Available: http://cccc.sourceforge.net/ [Accessed: 2020-12-03].

[182] A. J. Fernández-García, L. Iribarne, A. Corral, and J. Criado, "A Comparison of Feature Selection Methods to Optimize Predictive Models Based on Decision Forest Algorithms for Academic Data Analysis," in *World Conference on Information Systems and Technologies*, pp. 338–347, Springer, 2018.

[183] M. Bressan, Y. Rosseel, and L. Lombardi, "The effect of faking on the correlation between two ordinal variables: Some population and Monte Carlo results," *Frontiers in psychology*, vol. 9, p. 1876, 2018.

[184] M.-T. Puth, M. Neuhäuser, and G. D. Ruxton, "Effective use of Pearson's product–moment correlation coefficient," *Animal behaviour*, vol. 93, pp. 183–189, 2014.

[185] M. N. Asim, M. Wasim, M. S. Ali, and A. Rehman, "Comparison of feature selection methods in text classification on highly skewed datasets," in *2017 First International Conference on Latest trends in Electrical Engineering and Computing Technologies (INTELLECT)*, pp. 1–8, 2017.

[186] T. M. Mitchell, *Machine Learning*. USA: McGraw-Hill, Inc., 1 ed., 1997.

[187] G. Langs, B. H. Menze, D. Lashkari, and P. Golland, "Detecting stable distributed patterns of brain activation using gini contrast," *NeuroImage*, vol. 56, no. 2, pp. 497 – 507, 2011. Multivariate Decoding and Brain Reading.

[188] M. Nassar, H. Safa, A. A. Mutawa, A. Helal, and I. Gaba, "Chi Squared Feature Selection over Apache Spark," in *International Database Engineering & Applications Symposium (IDEAS)*, pp. 1–5, 2019.

[189] N. Koroniotis, N. Moustafa, E. Sitnikova, and B. Turnbull, "Towards the development of realistic botnet dataset in the internet of things for network forensic analytics: Bot-IoT dataset," *Future Generation Computer Systems*, vol. 100, pp. 779–796, 2019.

[190] D. G. Altman and J. M. Bland, "Diagnostic tests. 1: Sensitivity and specificity.," *BMJ: British Medical Journal*, vol. 308, no. 6943, p. 1552, 1994.

[191] V. Lenarduzzi, N. Saarimäki, and D. Taibi, "The Technical Debt Dataset," in *International Conference on Predictive Models and Data Analytics in Software Engineering*, pp. 2–11, 2019.

[192] C. Thirumalai, P. A. Reddy, and Y. J. Kishore, "Evaluating software metrics of gaming applications using code counter tool for C and C++ (CCCC)," in *2017 International conference of Electronics, Communication and Aerospace Technology (ICECA)*, vol. 2, pp. 180–184, 2017.

[193] A. Afzal, C. Schmitt, S. Alhaddad, Y. Grynko, J. Teich, J. Forstner, and F. Hannig, "Solving Maxwell's Equations with Modern C++ and SYCL: A Case Study," in *IEEE 29th International Conference on Application-specific Systems, Architectures and Processors (ASAP)*, pp. 1–8, 2018.

[194] "SonarQube C++ plugin (Community)." [online]. Available: `https://github.com/SonarOpenCommunity/sonar-cxx` [Accessed: 2020-12-03].

[195] F. Palomba, G. Bavota, M. Di Penta, F. Fasano, R. Oliveto, and A. De Lucia, "On the diffuseness and the impact on maintainability of code smells: a large scale empirical investigation," *Empirical Software Engineering*, vol. 23, no. 3, pp. 1188–1221, 2018.

[196] **Filus, Katarzyna**, P. Boryszko, J. Domańska, M. Siavvas, and E. Gelenbe, "**Efficient feature selection for static analysis vulnerability prediction**," *Sensors*, vol. 21, no. 4, p. 1133, 2021.

[197] C. Marantos, M. Siavvas, D. Tsoukalas, C. P. Lamprakos, L. Papadopoulos, P. Boryszko, **Filus, Katarzyna**, J. Domańska, A. Ampatzoglou, A. Chatzigeorgiou, *et al.*, "**SDK4ED: One-click platform for Energy-aware, Maintainable and Dependable Applications**," in *Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pp. 981–986, 2022.

[198] V. Kaptan and E. Gelenbe, "Fusing terrain and goals: agent control in urban environments," in *Multisensor, Multisource Information Fusion: Architectures, Algorithms, and Applications 2006*, vol. 6242, p. 624208, International Society for Optics and Photonics, 2006.

[199] S. Evmorfos, G. Vlachodimitropoulos, N. Bakalos, and E. Gelenbe, "Neural network architectures for the detection of SYN flood attacks in IoT systems," in *ACM International Conference on PErvasive Technologies Related to Assistive Environments*, no. 69, pp. 1–4, ACM; https://doi.org/10.1145/3389189.3398000, 2020.

[200] "Cisco 2018 Annual Cybersecurity Report." [online], Cisco 2018. Available: `https://www.cisco.com/c/dam/en_us/about/annual-report/2018-annual-report-full.pdf` [Accessed: 2020-06-26].

[201] "Baseline Security Recommendations for IoT." [online], ENISA report, November 2017. Available: `https://www.enisa.europa.eu/publications/baseline-security-recommendations-for-iot` [Accessed: 2020-06-26].

[202] E. Gelenbe and Y. M. Kadioglu, "Energy Life-Time of Wireless Nodes with Network Attacks and Mitigation," in *2018 IEEE International Conference on Communications Workshops, ICC Workshops 2018, Kansas City, MO, USA, May 20-24, 2018*, pp. 1–6, IEEE, 2018.

[203] H. K. Dam, T. Tran, T. Pham, S. W. Ng, J. Grundy, and A. Ghose, "Automatic feature learning for vulnerability prediction," *arXiv preprint arXiv:1708.02368*, 2017.

[204] Y. Meidan, M. Bohadana, Y. Mathov, Y. Mirsky, A. Shabtai, D. Breitenbacher, and Y. Elovici, "N-baiot—network-based detection of IoT botnet attacks using deep autoencoders," *IEEE Pervasive Computing*, vol. 17, no. 3, pp. 12–22, 2018.

[205] V. H. Bezerra, V. G. T. da Costa, R. A. Martins, S. B. Junior, R. S. Miani, and B. B. Zarpelao, "Providing IoT host-based datasets for intrusion detection research," in *Anais do XVIII Simpósio Brasileiro em Segurança da Informação e de Sistemas Computacionais*, pp. 15–28, 2018.

[206] H. Kang, D. H. Ahn, G. M. Lee, J. D. Yoo, K. H. Park, and H. K. Kim, "IoT network intrusion dataset," 2019.

[207] "1998 DARPA Intrusion Detection Evaluation Data Set. MIT Lincoln Laboratory." [online], 1998. Available: `https://www.ll.mit.edu/r-d/datasets/1998-darpa-intrusion-detection-evaluation-dataset` [Accessed: 2020-06-26].

[208] "1999 DARPA Intrusion Detection Evaluation Data Set. MIT Lincoln Laboratory." [online], 1999. Available: `https://www.ll.mit.edu/r-d/datasets/1999-darpa-intrusion-detection-evaluation-dataset` [Accessed: 2020-06-26].

[209] S. Hettich and S. Bay, "The UCI KDD Archive. Irvine, CA: University of California, Department of Information and Computer Science." [online], 1999. Available: `http://kdd.ics.uci.edu` [Accessed: 2020-06-26].

[210] H. Hindy, D. Brosset, E. Bayne, A. Seeam, C. Tachtatzis, R. Atkinson, and X. Bellekens, "A Taxonomy and Survey of Intrusion Detection System Design Techniques, Network Threats and Datasets," *arXiv preprint arXiv:1806.03517*, 2018.

[211] J. Galeano-Brajones, D. Cortés-Polo, J. F. Valenzuela-Valdés, A. M. Mora, and J. Carmona-Murillo, "Detection and mitigation of DoS attacks in SDN. An experimental approach," in *2019 Sixth International Conference on Internet of Things: Systems, Management and Security (IOTSMS)*, pp. 575–580, 2019.

[212] N. Guizani and A. Ghafoor, "A Network Function Virtualization System for Detecting Malware in Large IoT Based Networks," *IEEE Journal on Selected Areas in Communications*, vol. 38, no. 6, pp. 1218–1228, 2020.

[213] A. Brandon, M. Seekins, B. V. Joshua, C. Samuel, and J. Haller, "Network Data Analysis to Support Risk Management in an IoT Environment," in *2019 IEEE 10th Annual Ubiquitous Computing, Electronics & Mobile Communication Conference (UEMCON)*, pp. 0063–0068, 2019.

[214] D. Kobak and P. Berens, "The art of using t-SNE for single-cell transcriptomics," *Nature communications*, vol. 10, no. 1, pp. 1–14, 2019.

[215] D. Peng, S. Zheng, Y. Li, G. Ke, D. He, and T.-Y. Liu, "How could Neural Networks understand Programs?," in *International Conference on Machine Learning*, pp. 8476–8486, PMLR, 2021.

[216] S. Kim, J. Zhao, Y. Tian, and S. Chandra, "Code Prediction by Feeding Trees to Transformers," in *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*, pp. 150–162, 2021.

[217] **Filus, Katarzyna**, J. Domańska, and E. Gelenbe, "**Random neural network for lightweight attack detection in the iot**," in *Modelling, Analysis, and Simulation of Computer and Telecommunication Systems: 28th International Symposium, MASCOTS 2020, Nice, France, November 17–19, 2020, Revised Selected Papers 28*, pp. 79–91, Springer, 2021.

[218] **Filus, Katarzyna**, M. Siavvas, J. Domańska, and E. Gelenbe, "**The random neural network as a bonding model for software vulnerability prediction**," in *Modelling, Analysis, and Simulation of Computer and Telecommunication Systems: 28th International Symposium, MASCOTS 2020, Nice, France, November 17–19, 2020, Revised Selected Papers 28*, pp. 102–116, Springer, 2021.

[219] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, "MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications," *arXiv preprint arXiv:1704.04861*, 2017.

[220] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, "MobileNetV2: Inverted Residuals and Linear Bottlenecks," in *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 4510–4520, 2018.

[221] G. Huang, Z. Liu, L. Van Der Maaten, and K. Q. Weinberger, "Densely Connected Convolutional Networks," in *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 4700–4708, 2017.

[222] T. Wang, Y. Zhu, C. Zhao, W. Zeng, J. Wang, and M. Tang, "Adaptive Class Suppression Loss for Long-Tail Object Detection," in *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 3103–3112, 2021.

[223] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, "Rethinking the Inception Architecture for Computer Vision," in *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 2818–2826, 2016.

[224] F. Chollet, "Xception: Deep learning with depthwise separable convolutions," in *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 1251–1258, 2017.

[225] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet Classification with Deep Convolutional Neural Networks," *Advances in Neural Information Processing Systems*, vol. 25, 2012.

[226] **Filus, Katarzyna** and J. Domańska, "**Recycling of generic ImageNet-trained models for smart-city applications**," in *The 10th IEEE International Conference on Data Science and Advanced Analytics*, 2023.

[227] G. A. Miller, *WordNet: An electronic lexical database*. MIT press, 1998.

[228] **Filus, Katarzyna**, Ł. Sobczak, J. Domańska, A. Domański, and R. Cupek, "**Real-time testing of vision-based systems for AGVs with ArUco markers**," in *IEEE International Conference on Big Data (Big Data)*, pp. 6290–6298, 2022.

[229] Y. Feng, B. Wu, Y. Fan, L. Liu, Z. Li, and S.-T. Xia, "Boosting Black-Box Attack with Partially Transferred Conditional Adversarial Distribution," in *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 15095–15104, 2022.

[230] C. Xie, Y. Wu, L. v. d. Maaten, A. L. Yuille, and K. He, "Feature Denoising for Improving Adversarial Robustness," in *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 501–509, 2019.

[231] O. Gungor, T. Rosing, and B. Aksanli, "STEWART: STacking Ensemble for White-Box AdversaRial Attacks Towards more resilient data-driven predictive maintenance," *Computers in Industry*, vol. 140, p. 103660, 2022.

[232] S.-M. Moosavi-Dezfooli, A. Fawzi, O. Fawzi, and P. Frossard, "Universal adversarial perturbations," in *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 1765–1773, 2017.

[233] K. R. Mopuri, V. Shaj, and R. V. Babu, "Adversarial Fooling Beyond "Flipping the Label"," in *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*, pp. 778–779, 2020.

[234] "Dataset for the NIPS 2017 adversarial competition." [online], -. Available: `https://github.com/cleverhans-lab/cleverhans/tree/master/cleverhans_v3.1.0/examples/nips17_adversarial_competition/dataset` [Accessed: 2023-02-11].

[235] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database," in *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 248–255, 2009.

[236] M. Li, C. Deng, T. Li, J. Yan, X. Gao, and H. Huang, "Towards Transferable Targeted Attack," in *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 641–649, 2020.

[237] K. He, X. Zhang, S. Ren, and J. Sun, "Identity Mappings in Deep Residual Networks," in *European Conference on Computer Vision*, pp. 630–645, Springer, 2016.

[238] **Filus, Katarzyna** and J. Domańska, "**NetSat: Network Saturation Adversarial Attack**," in *IEEE International Conference on Big Data*, 2023.

[239] B. Zhou, A. Khosla, A. Lapedriza, A. Oliva, and A. Torralba, "Object Detectors Emerge in Deep Scene CNNs," *arXiv preprint arXiv:1412.6856*, 2014.

[240] **Filus, Katarzyna** and J. Domańska, "**Global Entropy Pooling layer for Convolutional Neural Networks**," *Neurocomputing*, p. 126615, 2023.

[241] C. Szegedy, S. Ioffe, V. Vanhoucke, and A. Alemi, "Inception-v4, Inception-ResNet and the Impact of Residual Connections on Learning," in *AAAI conference on artificial intelligence*, vol. 31, 2017.

[242] **Filus, Katarzyna** and J. Domańska, "**NAM: What Does a Neural Network See?**," in *IEEE International Joint Conference on Neural Networks (IJCNN)*, pp. 1–8, 2022.

[243] X. Meng, H. Wang, and B. Liu, "A Robust Vehicle Localization Approach Based on GNSS/IMU/DMI/LiDAR Sensor Fusion for Autonomous Vehicles," *Sensors*, vol. 17, no. 9, p. 2140, 2017.

[244] H. K. Fard, Y. Chen, and K. K. Son, "Indoor positioning of mobile devices with agile iBeacon deployment," in *IEEE 28th Canadian conference on electrical and computer engineering (CCECE)*, pp. 275–279, 2015.

[245] C. Xiao, D. Yang, Z. Chen, and G. Tan, "3-D BLE indoor localization based on denoising autoencoder," *IEEE Access*, vol. 5, pp. 12751–12760, 2017.

[246] A. Alarifi, A. Al-Salman, M. Alsaleh, A. Alnafessah, S. Al-Hadhrami, M. A. Al-Ammar, and H. S. Al-Khalifa, "Ultra wideband indoor positioning technologies: Analysis and recent advances," *Sensors*, vol. 16, no. 5, p. 707, 2016.

[247] A. Poulose and D. S. Han, "UWB indoor localization using deep learning LSTM networks," *Applied Sciences*, vol. 10, no. 18, p. 6290, 2020.

[248] M. Uradzinski, H. Guo, X. Liu, and M. Yu, "Advanced Indoor Positioning Using Zigbee Wireless Technology," *Wireless Personal Communications*, vol. 97, no. 4, pp. 6509–6518, 2017.

[249] V. Bianchi, P. Ciampolini, and I. De Munari, "RSSI-based indoor localization and identification for ZigBee wireless sensor networks in smart homes," *IEEE Transactions on Instrumentation and Measurement*, vol. 68, no. 2, pp. 566–575, 2018.

[250] R. C. Luo and T.-J. Hsiao, "Indoor localization system based on hybrid Wi-Fi/BLE and hierarchical topological fingerprinting approach," *IEEE Transactions on Vehicular Technology*, vol. 68, no. 11, pp. 10791–10806, 2019.

[251] S. Sadowski and P. Spachos, "RSSI-Based Indoor Localization With the Internet of Things," *IEEE Access*, vol. 6, pp. 30149–30161, 2018.

[252] X. Hou, T. Arslan, and J. Gu, "Indoor localization for Bluetooth low energy using wavelet and smoothing filter," in *2017 International Conference on Localization and GNSS (ICL-GNSS)*, pp. 1–6, 2017.

[253] A. Thaljaoui, T. Val, N. Nasri, and D. Brulin, "BLE localization using RSSI measurements and iRingLA," in *IEEE international conference on industrial technology (ICIT)*, pp. 2178–2183, 2015.

[254] K. Coussement and D. F. Benoit, "Interpretable data science for decision making," 2021.

[255] S. Barai, D. Biswas, and B. Sau, "Estimate distance measurement using NodeMCU ESP8266 based on RSSI technique," in *IEEE Conference on Antenna Measurements & Applications*, pp. 170–173, 2017.

[256] T. Chowdhury, M. M. Rahman, S.-A. Parvez, A. Alam, A. Basher, A. Alam, and S. Rizwan, "A multi-step approach for RSSI-based distance estimation using smartphones," in *International Conference on Networking Systems and Security*, pp. 1–5, 2015.

[257] V. Cantón Paterna, A. Calveras Auge, J. Paradells Aspas, and M. A. Perez Bullones, "A Bluetooth Low Energy Indoor Positioning System with Channel Diversity, Weighted Trilateration and Kalman Filtering," *Sensors*, vol. 17, no. 12, p. 2927, 2017.

[258] A. Noertjahyana, I. A. Wijayanto, and J. Andjarwirawan, "Development of Mobile Indoor Positioning System Application Using Android and Bluetooth Low Energy with Trilateration Method," in *International Conference on Soft Computing, Intelligent System and Information Technology*, pp. 185–189, 2017.

[259] H. Jamil, F. Qayyum, F. Jamil, and D.-H. Kim, "Enhanced PDR-BLE Compensation Mechanism Based on HMM and AWCLA for Improving Indoor Localization," *Sensors*, vol. 21, no. 21, p. 6972, 2021.

[260] K. Lee, Y. Nam, and S. D. Min, "An indoor localization solution using Bluetooth RSSI and multiple sensors on a smartphone," *Multimedia Tools and Applications*, vol. 77, no. 10, pp. 12635–12654, 2018.

[261] J.-s. Jeon, Y. Kong, Y. Nam, and K. Yim, "An indoor positioning system using bluetooth RSSI with an accelerometer and a barometer on a smartphone," in *2015 10th International Conference on Broadband and Wireless Computing, Communication and Applications (BWCCA)*, pp. 528–531, 2015.

[262] S. R. Jondhale and R. S. Deshpande, "GRNN and KF framework based real time target tracking using PSOC BLE and smartphone," *Ad Hoc Networks*, vol. 84, pp. 19–28, 2019.

[263] C. Martella, A. Miraglia, J. Frost, M. Cattani, and M. van Steen, "Visualizing, clustering, and predicting the behavior of museum visitors," *Pervasive and Mobile Computing*, vol. 38, pp. 430–443, 2017.

[264] P. Barsocchi, M. Girolami, and D. La Rosa, "Detecting Proximity with Bluetooth Low Energy Beacons for Cultural Heritage," *Sensors*, vol. 21, no. 21, p. 7089, 2021.

[265] P. Centorrino, A. Corbetta, E. Cristiani, and E. Onofri, "Managing crowded museums: Visitors flow measurement, analysis, modeling, and optimization," *Journal of Computational Science*, vol. 53, p. 101357, 2021.

[266] M. Kose, O. D. Incel, and C. Ersoy, "Online Human Activity Recognition on Smart Phones," in *Workshop on mobile sensing: from smartphones and wearables to big data*, vol. 16, pp. 11–15, 2012.

[267] M. M. Hassan, M. Z. Uddin, A. Mohamed, and A. Almogren, "A robust human activity recognition system using smartphone sensors and deep learning," *Future Generation Computer Systems*, vol. 81, pp. 307–313, 2018.

[268] A. Murad and J.-Y. Pyun, "Deep Recurrent Neural Networks for Human Activity Recognition," *Sensors*, vol. 17, no. 11, p. 2556, 2017.

[269] A. Ignatov, "Real-time human activity recognition from accelerometer data using Convolutional Neural Networks," *Applied Soft Computing*, vol. 62, pp. 915–922, 2018.

[270] J. Bai, B. He, H. Shou, V. Zipunnikov, T. A. Glass, and C. M. Crainiceanu, "Normalization and extraction of interpretable metrics from raw accelerometry data," *Biostatistics*, vol. 15, no. 1, pp. 102–116, 2014.

[271] T. Steinecker, A. Kurdas, N. Mansfeld, M. Hamad, R. J. Kirschner, S. Abdolshah, and S. Haddadin, "Mean Reflected Mass: A Physically Interpretable Metric for Safety Assessment and Posture Optimization in Human-Robot Interaction," in *International Conference on Robotics and Automation (ICRA)*, pp. 11209–11215, 2022.

[272] I. Guyon and A. Elisseeff, "An Introduction to Feature Extraction," in *Feature extraction*, pp. 1–25, Springer, 2006.

[273] M.-T. Puth, M. Neuhäuser, and G. D. Ruxton, "Effective use of Spearman's and Kendall's correlation coefficients for association between two measured traits," *Animal Behaviour*, vol. 102, pp. 77–84, 2015.

[274] L. S. Ambati and O. El-Gayar, "Human Activity Recognition: A Comparison of Machine Learning Approaches," *Journal of the Midwest Association for Information Systems*, vol. 2021, no. 1, p. 49, 2019.

[275] S. Ketu and P. K. Mishra, "Performance Analysis of Machine Learning Algorithms for IoT-Based Human Activity Recognition," in *Advances in Electrical and Computer Technologies*, pp. 579–591, Springer, 2020.

[276] C. Krauss, X. A. Do, and N. Huck, "Deep neural networks, gradient-boosted trees, random forests: Statistical arbitrage on the S&P 500," *European Journal of Operational Research*, vol. 259, no. 2, pp. 689–702, 2017.

[277] A. W. Wong, W. Sun, S. V. Kalmady, P. Kaul, and A. Hindle, "Multilabel 12-Lead Electrocardiogram Classification Using Gradient Boosting Tree Ensemble," in *Computing in Cardiology*, pp. 1–4, 2020.

[278] C. Neuwirth, C. Snyder, W. Kremser, R. Brunauer, H. Holzer, and T. Stöggl, "Classification of alpine skiing styles using GNSS and inertial measurement units," *Sensors*, vol. 20, no. 15, p. 4232, 2020.

[279] **Filus, Katarzyna**, S. Nowak, J. Domańska, and J. Duda, "**Cost-effective filtering of unreliable proximity detection results based on BLE RSSI and IMU readings using smartphones**," *Scientific Reports*, vol. 12, no. 1, p. 2440, 2022.

# Spis rysunków

3.1 Średnia, odchylenie standardowe oraz wartości minimalne i maksymalne ocen CVSS dla różnych typów podatności określonych za pośrednictwem standardu CWE. . . . . 59
3.2 Wartości korelacji Pearsona przy użyciu różnych podzbiorów zbioru danych (CWE-119, CWE-399, razem) dla badanych cech. Na osi x umieszczono oryginalne nazwy cech. . . . . 79
3.3 Wartości korelacji Spearmana przy użyciu różnych podzbiorów zbioru danych (CWE-119, CWE-399, razem) dla badanych cech. Na osi x umieszczono oryginalne nazwy cech. . . . 80
3.4 Wartości korelacji Kendalla przy użyciu różnych podzbiorów zbioru danych (CWE-119, CWE-399, razem) dla badanych cech. Na osi x umieszczono oryginalne nazwy cech. . . . . 80
3.5 Rangi cech uzyskane przy użyciu różnych typów metryk dla całego zbioru danych . . . . . 82
3.6 Rangi cech uzyskane przy użyciu różnych typów metryk dla podzbioru danych CWE-399 . . . . . 84
3.7 Rangi cech uzyskane przy użyciu różnych typów metryk dla podzbioru danych CWE-119 . . . . . 85
3.8 Zagregowane wyniki dokładności pogrupowane według podzbioru cech . . . . . 86
3.9 Zagregowane wyniki czułości pogrupowane według podzbioru cech . . . . . 87
3.10 Zagregowane wyniki specyficzności pogrupowane według podzbioru cech . . . . . 88
3.11 Statystyki dokładności (minimum, maksimum, średnia i odchylenie standardowe) uzyskane dla różnych klasyfikatorów. . . 89
3.12 Statystyki czułości (minimum, maksimum, średnia i odchylenie standardowe) uzyskane dla różnych klasyfikatorów. . . . . 89
3.13 Statystyki specyficzności (minimum, maksimum, średnia i odchylenie standardowe) uzyskane dla różnych klasyfikatorów.s . 90

3.14 Średnie drzewo decyzyjne z uwzględnieniem wszystkich 33 cech jako danych wejściowych dla podzbioru z uwzględnionymi podatnościami CWE-399. . . . . 91
3.15 Średnie drzewo decyzyjne z uwzględnieniem wszystkich 33 cech jako danych wejściowych dla podzbioru z uwzględnionymi podatnościami CWE-119. . . . . 92
3.16 Zarys systemu przewidywania luk w oprogramowaniu stworzony w oparciu zarówno o dane tekstowe, jak i metryki generowane z narzędzia do analizy statycznej kodu. Sieci neuronowe wykorzystane do stworzenia systemu to sieci konwolucyjne i losowe sieci neuronowe. . . . . 101
3.17 Macierze pomyłek uzyskane dla podstawowego wariantu Losowych Sieci Neuronowych oraz dla proponowanego wariantu z neutralną inicjalizacją wag oraz restrykcją wag nałożoną w procesie trenowania. . . . . 104
3.18 Wizualizacja danych za pośrednictwem algorytmu t-SNE dla elementów podatnych i neutralnych na podstawie różnych typów cech. . . . . 105
3.19 Znormalizowane macierze pomyłek prezentujące najlepsze wyniki osiągnięte przez model czysto tekstowy i model hybrydowy. 106
3.20 Fotografie przedstawiające środowisko i platformę testową. (a), (b) Środowisko eksperymentalne z robotem, (c) Wykorzystanie znacznika ArUco do fizycznego oznaczenia dużego obiektu. 114
3.21 Zdjęcia różnych obiektów zebrane za pośrednictwem stworzonego systemu . . . . . 117
3.22 Macierze pomyłek uzyskane na podstawie wyników uzyskanych dla MobileNet w czasie rzeczywistym. . . . . 118
3.23 Macierze pomyłek uzyskane dla sieci testowanych offline na podstawie danych zebranych podczas testów online (dane z Eksperymentów 1 i 2 łącznie). W tym przypadku brane są pod uwagę sieci oryginalne (1000 klas ImageNet w klasyfikatorze). 120
3.24 Macierze pomyłek uzyskane dla sieci testowanych offline na podstawie danych zebranych podczas testów online (dane z Eksperymentów 1 i 2 łącznie). W tym przypadku brane są pod uwagę sieci zmodyfikowane w taki sposób, aby klasyfikowały obrazy jedynie do klas docelowych (z odrzuceniem reszty kategorii). . . . . 121

3.25 Wygląd próbek wygenerowanych przy użyciu różnych typów ataków i MobileNetV2: Znormalizowany gradient, NetSat, FGSM i Signed NetSat. Przetestowano różne wartości współczynnika epsilon i liczby iteracji - celem było sprawdzenie wpływu szerokiego spektrum perturbacji - od bardzo słabych do tych, które są widoczne (z założeniem, że główny obiekt jest nadal rozpoznawalny dla człowieka). . . . . 129
3.26 Porównanie wyników NetSat i metody referencyjnej (znormalizowany gradient) dla 15 iteracji/ $\epsilon = 0.05$/MobileNetV2. We wszystkich przypadkach model zwrócił prawidłową prognozę dla czystego obrazu: Niemiecki Owczarek, Golden Retriever i Sznaucer miniaturowy odpowiednio. Zakłócenia dla obu ataków są praktycznie niewidoczne. Próbki generowane za pomocą NetSat pozwalają na uzyskanie etykiet, które są bardzo różne od prawdziwej etykiety. Ten brak podobieństwa jest odzwierciedlony przez metrykę DM. () - zaobserwowana wartość pewności sieci dla predykcji. . . . . 130
3.27 Wartości metryk Fooling rate, Dissimilarity Metric oraz Dissimilarity Metric for Success uzyskane dla analizowanych ataków oraz sieci. . . . . 132

4.1 Wykorzystanie NAM-max, NAM-mean i NAM-var do wizualizacji aktywacji wytrenowanych modeli CNN. Wyniki GRAD-CAM dla klasy TOP1 zostały podane jako odniesienie. Ilustracja przedstawiająca kościół. . . . . 143
4.2 Wykorzystanie NAM-max, NAM-mean i NAM-var do wizualizacji aktywacji wytrenowanych modeli CNN. Wyniki GRAD-CAM dla klasy TOP1 zostały podane jako odniesienie. Ilustracja przedstawiająca sznaucera miniaturowego. . . . . 144
4.3 Wykorzystanie NAM-max, NAM-mean i NAM-var do wizualizacji aktywacji wytrenowanych modeli CNN. Wyniki GRAD-CAM dla klasy TOP1 zostały podane jako odniesienie. Ilustracja przedstawiająca stado koni. . . . . 145
4.4 Wykorzystanie NAM-max, NAM-mean i NAM-var do wizualizacji aktywacji wytrenowanych modeli CNN. Wyniki GRAD-CAM dla klasy TOP1 zostały podane jako odniesienie. Ilustracja przedstawiająca narciarza biegowego. . . . . 146
4.5 Wykorzystanie NAM-max do ręcznej analizy przydatności Xception do wyodrębniania cech opisujących różne obiekty na obrazach. . . . . 148

4.6 Lepsze pokrycie obiektów cechami dla przykładowych obrazów za pomocą VGG16, ResNet50, InceptionV3 i Inception-ResNetV2. . . . . 149
4.7 Wpływ ataku NetSat na mapy cech z ostatniej warstwy konwolucyjnej modelu MobileNetV2. W celu generacji próbek wykorzystano 50 iteracji i $\epsilon = 0,1$. Wyniki pokazują, że NetSat nasyca konwolucyjne mapy cech i ukrywa znaczące wzorce - uniemożliwiając w ten sposób prawidłową predykcję. Pierwszy wiersz przedstawia oryginalne i zakłócone obrazy wraz z przewidywaniami TOP1, podczas gdy drugi wiersz przedstawia wizualizację NAM-mean dla odpowiednich obrazów z pierwszego wiersza. Pod obrazami (a) - (f) znajduje się predykcja (Pred:) oraz jej pewność (w nawiasie). . . . . 150
4.8 Plan eksperymentu. Eksperyment obejmuje przechodzenie między pięcioma stoiskami, na których znajdują się telefony komórkowe wysyłające ramki BLE. Celem jest określenie w danym oknie czasu (5 sekund) najbliższego stoiska oraz ewentualne odfiltrowanie estymacji w przypadku zaobserwowania ruchu. . . . . 160
4.9 Selekcja cech z wykorzystaniem wartości absolutnej współczynnika korelacji Pearsona, ustalonej dla cech i etykiety w formie numerycznej. . . . . 161
4.10 Macierze pomyłek uzyskane dla testowanych binarnych klasyfikatorów aktywności. . . . . 162
4.11 Filtrowanie niewiarygodnych wyników estymacji odległości opartych na proponowanych metrykach podstawowych: (a) metryka aktywności; (b) metryka RSSI. . . . . 164
4.12 Filtrowanie niewiarygodnych wyników estymacji odległości oparte na metryce wiarygodności w czasie. . . . . 165

# Spis tabel


3.1 Wykorzystane w pracy typy CWE posortowane względem malejącej liczby instancji. Rank oznacza pozycję w rankingu CWE TOP25. . . . . 57
3.2 Rozkład procentowy możliwych rezultatów wykorzystania różnych typów podatności określonych za pośrednictwem standardu CWE. . . . . 59
3.3 Rozkład procentowy wpływu na pojedyncze składniki triady CIA dla różnych typów podatności określonych za pośrednictwem standardu CWE. . . . . 59
3.4 Wyniki uzyskane za pośrednictwem Orthogonal Defect Classification dla różnych typów podatności określonych standardem CWE. . . . . 61
3.5 Liczba ostrzeżeń znalezionych przez FlawFinder, CppCheck i Visual Code. Każdy wiersz zawiera liczbę ostrzeżeń znalezionych dla plików zawierających instancje CVE należące do danego typu CWE, a nie liczbę podatności danego typu CWE. Konkretne typy CWE zwrócone dla analizatorów (jeśli taka informacja była dostępna) podano w szczegółowym opisie spostrzeżeń dotyczących danego narzędzia. . . . . 62
3.6 Liczność poszczególnych zbiorów elementów kodu wraz z informacją o podziale na klasy wykorzystanym w eksperymentach. 78
3.7 P-wartości uzyskane w analizie korelacji dla kolumn, w których co najmniej jedna p-wartość sugeruje, że hipoteza zerowa powinna zostać odrzucona. . . . . 81
3.8 10 najlepszych cech uzyskanych dla różnych technik rankingowych selekcji cech. . . . . 83
3.9 Liczba elementów w zbiorze testowym wraz z podziałem na elementy podatne i neutralne. . . . . 102

3.10 Porównanie dokładności, precyzji i czułości dla wszystkich testowanych wersji Losowej Sieci Neuronowej. Dla losowej inicjalizacji przeprowadzono 20 eksperymentów i podano wyniki średnie oraz odchylenie standardowe. Wyrażenia Losowa oraz Neutralna w nagłówku tabeli odnoszą się do sposobu inicjalizacji. RW oznacza proponowaną restrykcję wag. . . . . 103
3.11 Kodowanie różnych obiektów za pomocą identyfikatorów markerów ArUco i odpowiadających im klas ImageNet. Skrót Id. oznacza identyfikator. W kolumnie „ImageNet" zastosowano oryginalne (angielskie) nazwy klas zbioru ImageNet odpowiadające wykorzystanym w pracy obiektom. . . . . 114
3.12 Wyniki dokładności uzyskane w testach online i offline dla rozpatrywanych modeli rozpoznawania obiektów. W nawiasach podano wyniki uzyskane dla specjalizowanych modeli testowanych offline. . . . . 118

4.1 Predykcje TOP1 uzyskane za pomocą badanych wytrenowanych modeli. Predykcje te są wykorzystywane do generowania wyników GRAD-CAM. IncResNetV2 oznacza Inception-ResNetV2, a min. - miniaturowy. . . . . 140
4.2 Opis kolumn używanych w zbiorze danych wykorzystanym do testów obejmujących wykrywanie bliskości za pośrednictwem prezentowanej metody. W nawiasach umieszczono faktyczne stosowane nazwy kolumn. Te, faktyczne nazwy kolumn i jednocześnie oznaczenia poszczególnych nadawców zostały wykorzystane na finalnych wykresach prezentujących działanie systemu. . . . . 154

# Dodatek A

# Lista publikacji

Poniżej przedstawiono pełną listę publikacji autorki pracy:

1. Filus, K., and J. Domańska, „NetSat: Network Saturation Adversarial Attack", IEEE International Conference on Big Data (BigData), Sorrento, Italy, 2023, In Press.

2. Filus, K., and J. Domańska, „*Recycling of Generic ImageNet-trained Models for Smart-city Applications*", The 10th IEEE International Conference on Data Science and Advanced Analytics (DSAA), Thessaloniki, Greece, 2023.

3. Filus, K., and J. Domańska, „*Global Entropy Pooling Layer for Convolutional Neural Networks*", Neurocomputing, vol. 555, 2023.

4. Filus, K., and J. Domańska, „*Software Vulnerabilities in TensorFlow-Based Deep Learning Applications*", Computers & Security, vol. 124, 10/2022, 2023.

5. Filus, K., Ł. Sobczak, J. Domańska, A. Domański, and R. Cupek, „*Real-time testing of vision-based systems for AGVs with ArUco markers*", IEEE International Conference on Big Data, Osaka, Japan, 2022.

6. Filus, K., and J. Domańska, „*NAM: What Does a Neural Network See?*", International Joint Conference on Neural Networks (IJCNN 2022), IEEE WCCI 2022, Padova, Italy, 2022.

7. Filus, K., S. Nowak, J. Domańska, and J. Duda, „*Cost-Effective Filtering of Unreliable Proximity Detection Results Based on BLE RSSI and IMU Readings Using Smartphones*", Scientific Reports, vol. 12, issue 1, 2022.

8. Filus, K., P. Boryszko, J. Domańska, M. Siavvas, and E. Gelenbe, „*Efficient Feature Selection for Static Analysis Vulnerability Prediction*", Sensors, vol. 21 (4), issue Special Issue: Security and Privacy in Software Based Critical Contexts, 2021.

9. Filus, K., J. Domańska, and E. Gelenbe, „*Random Neural Network for Lightweight Attack Detection in the IoT*", MASCOTS 2020: Modelling, Analysis, and Simulation of Computer and Telecommunication Systems, vol. 12527: Springer International Publishing, pp. 79-91, 2021.

10. Filus, K., M. Siavvas, J. Domańska, and E. Gelenbe, „*The Random Neural Network as a Bonding Model for Software Vulnerability Prediction*", Modelling, Analysis, and Simulation of Computer and Telecommunication Systems, vol. 12527: Springer International Publishing, pp. 102-116, 2021.

11. Filus, K., A. Domański, J. Domańska, D. Marek, and J. Szyguła, „*Long-Range Dependent Traffic Classification with Convolutional Neural Networks Based on Hurst Exponent Analysis*", Entropy, vol. 22, issue 10, 2020.

12. Sobczak, Ł., K. Filus, M. Halama, and J. Domańska, „*Visual examination of relations between known classes for deep neural network classifiers*", IEEE International Conference on Big Data (BigData), Sorrento, Italy, 2023, In Press.

13. Halama, M., K. Filus, and J. Domańska, „*Robust category recognition based on deep templates for educational mobile applications*", IEEE International Conference on Big Data (BigData), Sorrento, Italy, 2023, In Press.

14. Kelesoglu, N., K. Filus, and J. Domańska, „*HierAct: a Hierarchical Model for Human Activity Recognition in Game-Like Educational Applications*", 2023 IEEE International Conference on Big Data (BigData), Sorrento, Italy, 2023, In Press.

15. Sobczak, Ł., K. Filus, J. Domańska, and A. Domański, „*Building a real-time testing platform for unmanned ground vehicles with UDP Bridge*", Sensors, vol. 22, issue 21, 2022.

16. Sobczak, Ł., K. Filus, J. Domańska, and A. Domański, „*Finding the best hardware configuration for 2D SLAM in indoor environments via simulation based on Google Cartographer*", Scientific Reports, vol. 12, 2022.

17. Sobczak, Ł., K. Filus, A. Domański, and J. Domańska, „*LIDAR Point Cloud generation for SLAM algorithm evaluation*", Sensors, vol. 21 (10), issue Special Issue: Advance in Sensors and Sensing Systems for Driving and Transportation: Part B, 2021.

18. Domański, A., J. Domańska, K. Filus, J. Szyguła, and T. Czachórski, „*The self-similar markovian sources*", Applied Sciences, vol. 10, issue 11, 2020.

19. Marek, D., J. Szyguła, A. Domański, J. Domańska, K. Filus, and M. Szczygieł, „*Adaptive Hurst-Sensitive Active Queue Management*", Entropy, vol. 24, issue 3, 2022.

20. Szyguła, J., A. Domański, J. Domańska, D. Marek, K. Filus, and S. Mendla, „*Supervised learning of Neural Networks for Active Queue Management in the Internet*", Sensors, vol. 21(15), issue Special Issue "Mathematical Modelling and Analysis in Sensors Networks", 2021.

21. Marek, D., A. Domański, J. Domańska, J. Szyguła, T. Czachórski, J. Klamka, and K. Filus, „*Approximation Models for the Evaluation of TCP/AQM Networks*", Bulletin of the Polish Academy of Sciences, Technical Sciences (BPASTS), vol. 70, issue 4, 2022.

22. Marantos, C., M. Siavvas, D. Tsoukalas, C. P. Lamprakos, L. Papadopoulos, P. Boryszko, K. Filus, J. Domańska, A. Ampatzoglou, A. Chatzigeorgiou, et al., „*SDK4ED: One-click platform for Energy-aware, Maintainable and Dependable Applications*", 25th Design, Automation and Test in Europe Conference, Belgium, 03/2022.