POLITECHNIKA GDAŃSKA
WYDZIAŁ ELEKTRONIKI, TELEKOMUNIKACJI I INFORMATYKI

Imię i nazwisko studenta: Jakub Kwiatkowski
Nr albumu: 192237
Poziom kształcenia: Studia drugiego stopnia
Forma studiów: stacjonarne
Kierunek studiów: Informatyka
Specjalność: Uczenie maszynowe

PRACA DYPLOMOWA MAGISTERSKA

Tytuł pracy w języku polskim: Badanie algorytmów wykrywania pojazdów w porze nocnej

Tytuł pracy w języku angielskim: A study of nighttime vehicle detection algorithms

Opiekun pracy: dr inż. Karolina Marciniuk

POLITECHNIKA GDAŃSKA | WYDZIAŁ ELEKTRONIKI, TELEKOMUNIKACJI I INFORMATYKI

OŚWIADCZENIE dotyczące pracy dyplomowej zatytułowanej: Badanie algorytmów wykrywania pojazdów w porze nocnej

Imię i nazwisko studenta: Jakub Kwiatkowski
Data i miejsce urodzenia: 13.03.1998, Gdynia
Nr albumu: 192237

Wydział: Wydział Elektroniki, Telekomunikacji i Informatyki
Kierunek: informatyka

Poziom kształcenia: drugi
Forma studiów: stacjonarne

Typ pracy: praca dyplomowa magisterska

Świadomy(a) odpowiedzialności karnej z tytułu naruszenia przepisów ustawy z dnia 4 lutego 1994 r. o prawie autorskim i prawach pokrewnych (t.j. Dz. U. z 2019 r. poz. 1231, z późn. zm.) i konsekwencji dyscyplinarnych określonych w ustawie z dnia 20 lipca 2018 r. Prawo o szkolnictwie wyższym i nauce (t.j. Dz. U. z 2020 r. poz. 85, z późn. zm.),[1] a także odpowiedzialności cywilnoprawnej oświadczam, że przedkładana praca dyplomowa została opracowana przeze mnie samodzielnie.

Niniejsza praca dyplomowa nie była wcześniej podstawą żadnej innej urzędowej procedury związanej z nadaniem tytułu zawodowego.

Wszystkie informacje umieszczone w ww. pracy dyplomowej, uzyskane ze źródeł pisanych i elektronicznych, zostały udokumentowane w wykazie literatury odpowiednimi odnośnikami zgodnie z art. 34 ustawy o prawie autorskim i prawach pokrewnych.

25.09.2023, Jakub Kwiatkowski

Data i podpis lub uwierzytelnienie w portalu uczelnianym Moja PG

**) Dokument został sporządzony w systemie teleinformatycznym, na podstawie §15 ust. 3b Rozporządzenia MNiSW z dnia 12 maja 2020 r. zmieniającego rozporządzenie w sprawie studiów (Dz.U. z 2020 r. poz. 853). Nie wymaga podpisu ani stempla.*

[1] Ustawa z dnia 20 lipca 2018 r. Prawo o szkolnictwie wyższym i nauce:

Art. 312. ust. 3. W przypadku podejrzenia popełnienia przez studenta czynu, o którym mowa w art. 287 ust. 2 pkt 1–5, rektor niezwłocznie poleca przeprowadzenie postępowania wyjaśniającego.

Art. 312. ust. 4. Jeżeli w wyniku postępowania wyjaśniającego zebrany materiał potwierdza popełnienie czynu, o którym mowa w ust. 5, rektor wstrzymuje postępowanie o nadanie tytułu zawodowego do czasu wydania orzeczenia przez komisję dyscyplinarną oraz składa zawiadomienie o podejrzeniu popełnienia przestępstwa.

STRESZCZENIE

Przedmiotem pracy jest sprawdzenie wybranych metod detekcji pojazdów w porze nocnej, które mogą być wykorzystane w celu zwiększenia widoczności pieszych na przejściach. Wykrycie pojazdu przez algorytm skutkowałoby zapaleniem się dodatkowego oświetlenia, w taki sposób, żeby kierowcy mogli odpowiednio wcześnie zareagować na pojawienie się na nim pieszych. Na podstawie przeglądu literatury wyselekcjonowano dwie metody oparte o detekcje reflektorów pojazdów. Pierwsza metoda wykorzystuje algorytmy przetwarzające obraz, a druga dane z czujnika intensywności światła. Algorytmy analizujące obraz zostały podzielone na klasyfikatory i detektory, które dodatkowo wskazują obszar wykrytego obiektu. Wykrywanie pojazdów w nocy charakteryzuje się tym, że obrazy są niższej jakości, często czarno-białe, występują liczne szumy i zakłócenia w postaci błysków świateł. W ciemnych warunkach rzadko widoczna jest cała sylwetka samochodu. Zazwyczaj widoczne są z oddali tylko światła, czasami w wyniku zaniedbania kierowcy tylko jedna żarówka jest sprawna lub źle zamontowana w wyniku czego oślepiają matrycę kamery. Ilość pojazdów na drogach ciągle rośnie, co powoduje większą ilość zagrożeń dla innych użytkowników ruchu, dlatego badania wykrywania pojazdów mogą mieć wpływ na podniesienie bezpieczeństwa pieszych na przejściach. W tym celu utworzony został zbiór danych, wybrano miary skuteczności algorytmów i przeprowadzono badania ich skuteczności* i wydajności na proponowanych platformach docelowych. Udało się przeprowadzić badania jedenastu algorytmów detekcji, dwóch algorytmów śledzenia, dwóch algorytmów klasyfikacji obrazu oraz dwóch autorskich sieci do klasyfikacji danych z czujnika intensywności światła. Na podstawie badań zaproponowano kilka rozwiązań, które mają na celu pomóc w zapobieganiu wypadkom z udziałem niechronionych uczestników ruchu drogowego na przejściach dla pieszych.

Słowa kluczowe: Detekcja pojazdów, wykrywanie pojazdów w nocy, głębokie sieci neuronowe, czujnik intensywności światła, algorytmy śledzenia, klasyfikacja

Dziedzina nauki i techniki zgodna z OECD Nauki inżynieryjne i techniczne, Elektrotechnika, elektronika i inżynieria informatyczna, Robotyka i Automatyka

SPIS TREŚCI

LISTA SKRÓTÓW I TERMINÓW

HOG - histogram ukierunkowanych gradientów (ang. *Histogram of Oriented Gradients*)
SVM - maszyna wektorów nośnych (ang. *support vector machines*)
RGB - czerwony, zielony, niebieski (ang. *red, green, blue*)
EKF - rozszerzony filtr Kalmana (ang. *extended Kalman filter*)
RAM - pamięć o dostępie swobodnym (ang. *random-access memory*)
GPU - procesor graficzny (ang. *graphics processing unit*)
CPU - centralna jednostka obliczeniowa (ang. *central processing unit*)
TPU - jednostka przetwarzania tensorów (ang. *tensor processing unit*)
AI - sztuczna inteligencja (ang. *artificial intelligence*)
IoT - internet rzeczy (ang. *internet of things*)
FPS - klatki na sekundę (ang. *frames per second*)
VOC - klasy wizualnych obiektów (ang. *visual object classes*)
LSTM - długa pamięć krótkotrwała (ang. *ang. long short-term memory*)
CNN - splotowa sieć neuronowa (ang. *convolutional neural network*)
mAP - średnia arytmetyczna precyzji (ang. *mean average precision*)
ASIC - specjalizowany układ scalony (ang. *application-specific integrated circuit*)
TOPS - bilion operacji na sekundę (ang. *trillion operations per second*)
IOU - część wspólna do całości (ang. *intersection over union*)
DPM - modele deformowalnych części (ang. *deformable parts models*)
framework - platforma programistyczna
batch - porcja danych

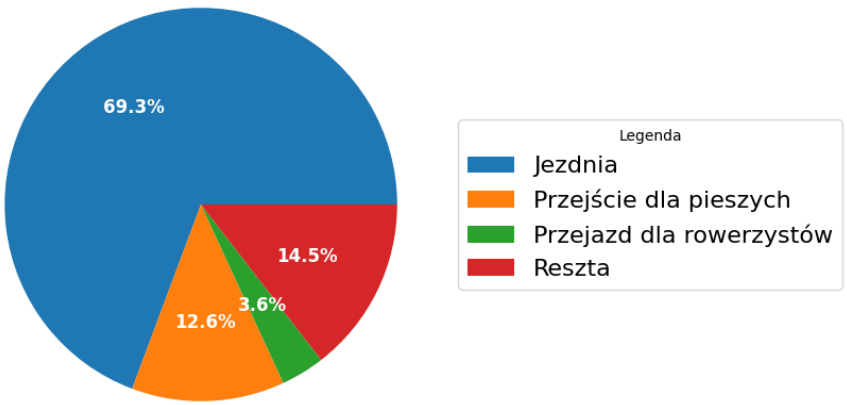
1. WSTĘP

Przedmiotem pracy jest sprawdzenie wybranych metod detekcji pojazdów w porze nocnej, które mogą być wykorzystane w celu zwiększenia widoczności pieszych na przejściach. Wykrycie pojazdu przez algorytm skutkowałoby zapaleniem się dodatkowego oświetlenia na przejściu w taki sposób, żeby kierowca mógł odpowiednio wcześnie zareagować na pojawienie się na nim pieszego. W tym celu została zaproponowana platforma, która wykorzystując dane z kamery i czujnika intensywności światła wykryje zbliżający się pojazd.

Z roku na rok ilość pojazdów na drogach w Polsce, jak i na świecie się zwiększa [1, 2]. Ze statystyk transportu drogowego w Polsce w latach 2018-2019 wynika, że większość wypadków powodują kierujący pojazdami [3]. Badania przeprowadzone przez "International Transport Forum"[4] dowodzą, że w niektórych krajach rozwiniętych (Wielka Brytania i Stany Zjednoczone) na przestrzeni lat 2010 - 2019 wzrosła liczba wypadków z udziałem pieszych.

W Polsce w roku 2022 wypadków, których przyczyną było nieustąpienie pierwszeństwa pieszemu na przejściu dla pieszych, było 11,6%, a 3,5% wypadków było spowodowanych z winy pieszego [5].

W pracy skupiono się na wykrywaniu pojazdów na przejściach dla pieszych, dlatego, że jest drugim najczęstszym miejscem wypadków [5]. Na rysunku 1.1 ukazano statystyki dotyczące miejsc wypadków w Polsce. Na przejściach dla pieszych ma miejsce 12,6% wszystkich wypadków. Informowanie kierujących pojazdy o uczestnikach ruchu przekraczających przejścia i przejazdy jest poważnym problemem. Po zmroku widoczność pieszego na przejściu pogarsza się, zwłaszcza wtedy, kiedy z naprzeciwka nadjeżdża pojazd, który oślepia kierowcę z naprzeciwka. Proponowane rozwiązanie ma na celu pomóc w redukcji wypadków z udziałem pieszych na drogach.



Rysunek 1.1: Statystyki miejsc wypadków w Polsce w roku 2022. Opracowanie według statystyk policji[5]

Występują liczne badania, które zostały poświęcone wykrywaniu pojazdów w dobrych warunkach oświetleniowych. W pracy "Structured learning via convolutional neural networks for vehicle detection"[6] przeprowadzono badania wykorzystując splotową sieć neuronową do segmentacji binarnej do wykrywania pojazdów. W innej pracy [7] badania poświęcono sieci splotowej w celu detekcji pojazdów w różnej skali. Przed dominacją sieci neuronowych wykorzystywane były inne algorytmy, takie jak maszyna wek-

torów nośnych wykorzystujące wyekstrahowane cechy przy pomocy *Histogram Oriented Gradients* [8]. W jednym z artykułów [9] wykorzystany został algorytm *AdaBoost* [10].

Poruszane są również tematy detekcji pieszych i rowerzystów [11]. Tworzone są też liczne prace skupiające się na wykrywaniu pojazdów. Część z nich skupia się na przednich światłach pojazdów [12], a część na wykrywaniu świateł z tyłu [9]. Wiele prac poświęconych jest detekcji pojazdów w nocy [13, 14] jak i śledzeniu ich [15], które jest przydatne w zastosowaniach monitoringu miejskiego.

Przeprowadzono też wiele badań poświęconych wykrywaniu pojazdów w niekorzystnych warunkach. Praca zatytułowana "Robust Vehicle Detection and Distance Estimation Under Challenging Lighting Conditions"[16] proponuje segmentację cech Haar'a (ang. *Haar-like features*) do detekcji pojazdów w trudnych warunkach oświetleniowych. W innej pracy zatytułowanej "Raindrop-Tampered Scene Detection and Traffic". Praca zatytułowana "Nighttime Visibility Analysis and Estimation Method in the Presence of Dense Fog"[17] skupiła się nad wykrywaniem pojazdów podczas gęstej mgły. Głównym problemem wykrywania pojazdów w takich warunkach są powstające aureole wokół źródeł światła (ang. *halos around light sources*). W obu pracach wykorzystywana jest segmentacja i metody kontrastowe, do wykrycia anomalii. Inne badanie zatytułowane "Improving the Performance of Vehicle Detection System in Bad Weathers"[18] wykorzystało okno przesuwne Violi i Jones'a (ang. *Viola and Jones's sliding-window*) do detekcji. Badanie "Flow Estimation for Nighttime Traffic Surveillance"zajęto się problemem wykrywania pojazdów w porze nocnej, gdzie na obiektywie kamery znajduje się woda.

W Polsce w latach 2020-2022 najwięcej wypadków było w ciągu dnia i wynika to z tego, że wtedy jeździ najwięcej samochodów. Jednak w nocy jest czterokrotnie większe prawdopodobieństwo, że jeśli zdarzył się wypadek, to był on śmiertelny [5, 19, 20].

Informacja o nadjeżdżającym pojeździe jest szczególnie istotna w nocy, kiedy widoczność zarówno dla kierowcy, jak i pieszego jest ograniczona. Praca skupia się na badaniu algorytmów wykrywania pojazdów w porze nocnej, przy pomocy kamery *RGB* i czujnika intensywności światła. Po zachodzie słońca rzadko jest widoczna cała sylwetka samochodu. Na obrazie najbardziej widoczne są światła pojazdu. Na ich wykrywaniu skupia się algorytm. Światła mijania są trudne do wykrycia ze względu na to, że mogą być nieprawidłowo ustawione i mogą spowodować oślepianie matrycy kamery. Częstym przypadkiem jest też, że jedna żarówka w pojeździe jest uszkodzona tak jak na 1.2a, w wyniku czego na obrazie widoczne jest tylko jedno światło, co stanowi dodatkowe utrudnienie [21].

(a) Pojazd z niesprawnym światłem **(b)** Odbicia świetlne **(c)** Oślepienie kamery

Rysunek 1.2: Obrazy z opracowanego zbioru danych - zaprezentowano przypadki trudne do detekcji dla algorytmów uczenia maszynowego

Współczesne prace do detekcji pojazdów w nocy wykorzystują głębokie sieci neuronowe. Badanie zatytułowane "GAN-Based Day-to-Night Image Style Transfer for Nighttime Vehicle Detection" [22] wykorzystało transferowanie stylu za pomocą sieci typu *GAN* (ang. *Generative Adversarial Network*) do rozszerzania zbioru danych i Faster-RCNN i YOLO9000 [23] do detekcji obrazów. W innym bada-

niu "An Improved YOLOX Model and Domain Transfer Strategy for Nighttime Pedestrian and Vehicle Detection"[24] wykorzystano sieć detekcji YOLOX [25] do detekcji pojazdów. W badaniu "Design and Evaluation of a Vehicle Detection System in Low Light Conditions" wykorzystane zostały cykliczne generatywne sieci adwersarzy (ang. *Cycle Generative Adversarial Networks*) do rozszerzenia zbioru i sieci detekcji takie jak YOLOv4, Faster-RCNN, ResNet, SSD MobileNetV2. Inne badania poruszające tę tematykę używały także algorytmów YOLOV3 [26, 27] lub MobileNetV3 [28] do wykrywania pojazdów.

W tym badaniu skupiono się na wykrywaniu pojazdów z wykorzystaniem głębokich sieci neuronowych. Zostały zebrane dane i utworzone zostały dwa zbiory danych - do klasyfikacji i detekcji. Za pomocą kamery i czujnika zebrano dane wizualne i informację o natężeniu światła. Następnie dane z czujnika zostały zsynchronizowane i ręcznie oznaczone.

Pierwszym rodzajem badanych algorytmów były sieci detekcji. Opisane zostały metody zastosowanych metod wykrywania obiektów i ekstrakcji cech przy pomocy sieci neuronowych. Sprawdzono sieci CenterNet HourGlass104, CenterNet MobileNetV2 FPN, EfficentDet D0, EfficentDet D1, SSD MobileNetV2 FPNLite 320x320, SSD MobileNetV2 FPNLite 640, SSD MobileNetV2, SSD ResNet50, Faster RCNN ResNet50, YOLOV8n oraz YOLOV8s. Wymienione sieci zostały sprawdzone pod kątem metryk dokładności, precyzji i czułości oraz wydajności na platformach docelowych. Przed treningiem zostały wykonane testy mające na celu jak najlepsze wytrenowanie wybranych sieci. Wszystkie algorytmy detekcji zostały wytrenowane na własnym zbiorze danych. Wyniki tych sieci zostały przeanalizowane pod kątem wybranych wcześniej metryk. Najlepszą siecią pod względem czasu treningu i osiągniętych metryk jest YOLOV8s.

Dokonana została analiza przebiegu treningu i różnice pomiędzy architekturami. Następnie zostały przedstawione przykłady detekcji wytrenowanych sieci oraz przeanalizowane zostały problemy związane z detekcją, takie jak znikanie wykrytego obiektu na kolejnej klatce, detekcja z bliska i nietypowe oświetlenie pojazdów. Dodatkowo została przeprowadzona optymalizacja sieci, stosując redukcję kolorów na obrazach w zbiorze danych.

Przeprowadzono testy wydajności na komputerze PC, dwóch mikrokomputerach z i bez akceleratora. Każda z sieci została sprawdzona w dwóch formatach zapisu i wykazano różnice w szybkości przetwarzania klatek. Przy wykorzystaniu formatów uniwersalnych szybsza okazała się platforma *Nvidia Jetson Nano*, natomiast przy formatach mobilnych *Raspberry Pi 4*. Na urządzeniach został wykorzystany również akcelerator sprzętowy. W przypadku wytrenowanych sieci odnotowano niewielkie przyspieszenie lub spowolnienie w szybkości detekcji, z powodu nieobsługiwanych instrukcji przez TPU. Z tego powodu zostały sprawdzone specjalnie wyeksportowane sieci producenta, które uzyskały znacznie lepsze wyniki niż sieci, których obliczenia nie były wykonywane na akceleratorze.

Zostało sprawdzone również, jak sprawują się sieci detekcji wytrenowanych na innym zbiorze podczas klasyfikacji obrazu. Osiągnęły one gorsze metryki niż klasyfikatory, jednak w badaniu wykazano dużą uniwersalność sieci detekcji. Następnie zbiór treningowy został ulepszony i wyniki sieci detekcji osiągnęły porównywalne metryki z klasyfikatorami obrazów, cechując się bardzo wysoką czułością. Sieci detekcji mają jednak swoje zalety w postaci lepszej wytłumaczalności decyzji, jak i możliwość śledzenia obiektów. Badane również było jak algorytmy śledzenia ruchu takie, jakie *BoT-SORT* i *ByteTrack* pomagają sieciom detekcji.

Drugim rodzajem algorytmów były sieci przeznaczone do klasyfikacji, które można podzielić na te, które korzystały z obrazu lub danych z czujnika intensywności światła. W badaniu obrazów wykorzystano dwie sieci, jedna z nich to MobileNetV2, druga to autorska sieć splotowa. Dla porównania wykorzystano również prosty algorytm zliczający ilość jasnych pikseli. Podczas tworzenia sieci kładziony był szczególny nacisk na małą liczbę parametrów algorytmu. Dodatkowo sprawdzono jak sieci detekcji,

takie jak YOLOV8n i YOLOV8s radzą sobie w zadaniu klasyfikacji binarnej. Wykorzystano również algorytmy śledzenia ruchu, które miały na celu polepszyć wyniki klasyfikacji detektorów. Wykorzystano dwie sieci klasyfikatorów, które wykorzystywały dane z czujnika intensywności światła. Obie były autorskimi rozwiązaniami, pierwsza sieć wykorzystywała sploty do ekstrakcji cech, a druga blok LSTM.

Dzięki badaniom udało się wybrać najlepsze rozwiązania w zależności od platformy sprzętowej i oczekiwanych wyników.

2. PLATFORMA DOCELOWA

W pracy zaproponowano platformę docelową, na której będą wykorzystane algorytmy wykrywania pojazdów w nocy. Urządzenie przy pomocy algorytmu z wykorzystującego obraz z kamery i informacje z czujnika natężenia światła będzie wykrywać nadjeżdżające w nocy pojazdy.

2.1. Kryteria przy wyborze sprzętu

Przy wyborze sprzętu kierowano się wybranymi kryteriami proponowanymi w książce "Sensors Handbook"[29]. Wyselekcjonowano następujące kryteria:

Zakres pomiarowy: Przy wyborze kamery i czujnika intensywności światła priorytetem była dużą dostępność i niska cena. Celem jest, aby testowane algorytmy były w stanie dokonywać wysokiej jakości detekcji na sprzęcie powszechnego użytku, który często ma mniejszy zakres pomiarowy niż specjalistyczne urządzenia.

Zakres temperaturowy: Wybrane urządzenia powinny być odporne na czynniki środowiskowe i nie ulegać szybkiemu zużyciu. Sprawdzone zostały zakresy temperaturowe dla każdego urządzenia.

Koszty: Cena platformy i związane z nim koszty eksploatacji jak np. kalibracja powinny być jak najniższe. Wybór urządzeń, które są w powszechnym użytku, powinien zagwarantować niską cenę. W przypadku wzrostu ceny danego komponentu można go zastąpić innym urządzeniem z podobną specyfikacją. Wykorzystanie zaawansowanych algorytmów i normalizacji powinno umożliwić rzadsze kalibrowanie urządzenia.

Wymiary: Pod uwagę były rozmiary urządzenia. Docelowe urządzenie powinno być kompaktowych rozmiarów i łatwe w montażu. Wybrane urządzenia spełniają te wymagania.

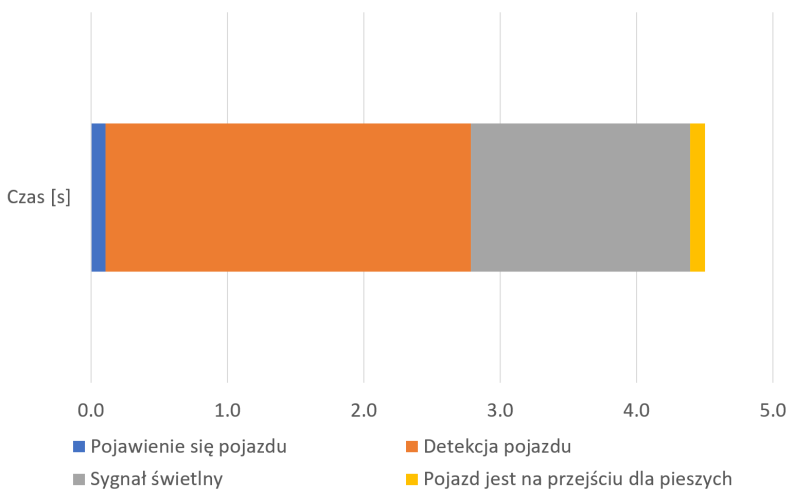
Interfejsy komunikacyjne: Platforma sprzętowa potrzebuje złącz GPIO do podłączenia czujnika intensywności światła oraz opcjonalnej możliwości podłączenia czujnika na przykład APDS, który umożliwia wykrycie zbliżającego się pieszego do przejścia dla pieszych. Wymagane są też złącza do podłączenia kamery RGB. *Raspberry Pi 4* ma wejście *CSI*, a Nvidia Jetson ma wejście *CSI-2*. Wejścia te są uniwersalne i na rynku jest duży wybór kamer, do tych interfejsów. W przypadku *Arduino Portenta H7* wykorzystywany jest dodatkowy "*Vision Shield*"[30], który trzeba dodatkowo dokupić do zestawu.

Dostępność i wsparcie techniczne: Wybrane urządzenia są popularne i mają dostępne wsparcie techniczne od producenta i społeczności.

Obsługiwane biblioteki: Musi być możliwe uruchomienie badanych algorytmów detekcji. Do tego potrzebna jest możliwość obsługiwania bibliotek takich jak *Tensorflow* [31], *Tensorflow Lite* [32] i *PyTorch* [33].

Wydajność: Zastosowane algorytmy wymagają szybkich pomiarów. Platforma powinna dostarczać pomiary w odpowiednio krótkim czasie. W tym celu porównano mikrokomputery pod kątem wydajności procesora, ilości pamięci RAM i wykorzystywanego akceleratora sprzętowego. Algorytm musi w odpowiednim czasie wykryć pojazd i poinformować pieszego o jego zbliżaniu się. W tym celu przyjęto następujące założenia. Zastosowany algorytm będzie wykorzystywany w obszarze zabudowanym, w którym prędkość maksymalna wynosi 50 km/h [34]. Częstym przypadkiem jest przekraczanie prędkości przez kierowcę [3] i ważne jest, aby uwzględnić sytuacje skrajne. W tym celu został przyjęty margines błędu wynoszący dwukrotność maksymalnej dopuszczalnej prędkości - 100 km/h co odpowiada w przybliżeniu 28 m/s. Powołując się na jedno z badań, przeciętna reakcja człowieka w ruchu drogowym wynosi 0,627 sekundy [35]. W innym badaniu [36] średnia otrzymanych wyników jest podobna. Najgorsze wyniki

w pierwszym kwartylu oscylowały w granicy 0,8 sekundy. Uwzględniając osoby z najgorszymi wynikami i marginesem błędu zostało przyjęte, że czas reakcji człowieka wynosi 1 s. Czas reakcji układu hamulcowego w pewnym samochodzie w zależności od przebiegu wynosił od 0,31 s do 0,58 s [37]. Przyjęto, że czas reakcji układu hamulcowego wynosi 0,6 s. Czas na pojawienie się sygnału świetlnego to 1,6 sekundy przed dojechaniem pojazdu do przejścia dla pieszych. Z tego wynika, że ostrzeżenie powinno zadziałać minimalnie 45 metrów przed przejściem dla pieszych. W jednym z miejsc odległość kamery od skrzyżowania, czyli maksymalna odległość, z jakiej możliwe jest wykrycie pojazdu, wynosi 120 m. Istotne jest, aby zbliżający się pojazd na odcinku 45 metrów przed przejściem dla pieszych został wykryty. W badaniu odległość ta wynosi 75 m. Ważnym czynnikiem jest, aby pojazd został wykryty jak najwcześniej. Czas na detekcję pojazdu to 2,7 sekundy. Na rysunku 2.1 został przedstawiony wykres przebieg detekcji i sygnału ostrzegawczego w czasie.



Rysunek 2.1: Przebieg w czasie detekcji pojazdu i sygnału ostrzegawczego

2.2. Porównanie wybranych mikrokomputerów

Wstępnie pod uwagę zostały wzięte jedne z najbardziej popularnych małych komputerów/mikrokontrolerów - *Arduino Portenta H7* [38], *Raspberry Pi 4* [39] i NVIDIA Jetson Nano [40, 41]. A ostatecznie zostały wybrane 2 urządzenia - *Raspberry Pi 4* i *Nvidia Jetson Nano*. W celu sprawdzenia potencjalnej użyteczności urządzenia w projekcie zostały porównane pod względem potencjalnej szybkości przetwarzania obrazu przez sieci neuronowe przy pomocy procesora i karty graficznej. Ważnym parametrem jest też wielkość pamięci RAM, aby możliwe było możliwe uruchomienie algorytmu. Dodatkowo sprawdzona została opcjonalna możliwość wykorzystania akceleratora sprzętowego do przyspieszenia obliczeń na urządzeniu. Z uwagi na różne docelowe umiejscowienie urządzenia zarówno stacjonarne, jak i niestacjonarne, sprawdzany jest pobór mocy. Specyfikacja wybranych urządzeń została umieszczona w tabeli 2.1.

Raspberry Pi 4 [39] i inne urządzenia tego producenta są wykorzystywane w *IoT* [42] jak i w detekcji i śledzenia pojazdów [43, 44]. Ilość pamięci operacyjnej urządzenia pozwala na uruchomienie dużych sieci neuronowych takich jak CenterNet MobileNet [45]. Specyfikacja procesora również sugeruje obiecujące wyniki w kwestii wydajności podczas inferencji.

NVIDIA Jetson Nano zostało stworzone z myślą o zastosowaniach sztucznej inteligencji, producent umieszcza wiele przykładowych projektów, na których można się wzorować [46, 47]. Pod względem wydajności jest zbliżona do Raspberry Pi4. Posiada procesor z 4 rdzeniami oraz tę samą ilość pamięci operacyjnej. Ma nieco wyższy maksymalny pobór mocy - zamiast 15 W jak w *Raspberry Pi 4* ma 20 W. W

badaniu porównującym wydajność NVIDIA Jetson Nano okazał się około 5 razy szybszy od Raspberry Pi4 w teście wykorzystującym splotowe sieci neuronowe na obrazach [48].

Arduino Portenta H7 nie zostało wykorzystane, ale warto o nim wspomnieć. Ma ono zastosowanie w *IoT* i *Industry 4.0* [49], gdzie priorytetem są małe rozmiary urządzenia i niski pobór mocy. Pomimo małej mocy obliczeniowej urządzenia, wykorzystywane są również sieci neuronowe. Rozwiązanie nazywane jest Tiny ML, które jest wykorzystywane do detekcji wycieku gazu [50] lub wykrywania intruzów [51]. Komputer charakteryzuje się możliwością pracy w niskich temperaturach i w przeciwieństwie do pozostałych urządzeń ma bardzo mało pamięci o dostępie swobodnym (8 MB), która utrudnia uruchomienie sieci neuronowych o dużych rozmiarach. Z tego powodu na urządzeniu nie zostały przeprowadzone testy.

Tabela 2.1: Specyfikacja urządzeń. Raspberry Pi4 i NVIDIA Jetson Nano P3450 zostały wybrane do badania

	Raspberry Pi 4	**NVIDIA Jetson Nano P3450**	**Arduino Portenta H7**
Procesor	Broadcom BCM2711, Quad core Cortex-A72 @ 1.5 GHz	Quad-core ARM A57 CPU	STM32H747 dual-core Cortex M7 480 MHz Cortex M4 240MHz
RAM	4GB LPDDR4	4GB LPDDR4	8-64 MB
Akceleracja sprzętowa / GPU	Broadcom VideoCore VI	128- core Maxwell GPU @ 921 MHz	Chrom-ART graphical hardware Accelerator™
Dodatkowa akceleracja	Google Coral	Google Coral	Brak
Temperatura pracy	USB i Ethernet: 0 °C do 70 °C SoC: -40 °C do 85 °C	0 °C do 60 °C	Wi-Fi:-10 °C do +55 °C SoC:-40 °C do +85 °C
Pobór mocy	IDLE: 3,8 – 4,0 W MAX: 15,0 W	MAX: 5 V * 4 A = 20 W	IDLE: 2,95 μA * 5 V = 15 μW MAX: 5,0 W
Języki programowania	C, C++. Micropython, JavaScript, Python	C, C++. Micropython, JavaScript, Python	C, C++, Micropython, JavaScript
Framework'i AI	TensorFlow, TensorFlow Lite Caffe, MXNet, PyTorch	TensorFlow, TensorFlow Lite Caffe, MXNet, PyTorch	TensorFlow Lite
Wymiary [mm]	100 x 70 x 30	100 x 79 x 30	66 x 25 x brak danych*

*Producent nie oficlajnej wysokości urządzenia

2.3. Akcelerator sprzętowy

W badaniu został wykorzystany również akcelerator sprzętowy *Google Edge TPU coprocessor* [52]. Jest to niewielki układ ASIC (ang. *Application-specific integrated circuit*) zaprojektowany przez *Google*, który umożliwia szybkie przetwarzanie modeli *Tensorflow Lite* przy zachowaniu oszczędności energii. Jest zdolny do wykonania 4 bilionów operacji na sekundę (4 TOPS) przy zużyciu 2 watów mocy. Producent deklaruje, że inferencja na MobileNetV2 może być wykonywana z szybkością prawie 400 klatek na sekundę. Procesor TPU wykonuje obliczenia na wartościach 8-bitowych, dlatego większość sieci neuronowych przed wykorzystaniem musi zostać poddana kwantyzacji. Procesor również zawiera ograniczoną ilość instrukcji - niektóre z nich nie są wspierane przez procesor. W przypadku, kiedy instrukcja nie jest wspierana, obliczenia są wykonywane na procesorze CPU urządzenia, do którego został podłączony akcelerator [53]. Urządzenie wymaga specjalnego wyeksportowania sieci neuronowej w formacie tflite w celu optymalizacji zestawu instrukcji wykonywanych na sieci neuronowej. Producent udostępnia

potrzebne oprogramowanie do optymalizacji sieci.

Tabela 2.2: Specyfikacja urządzenia *Google Edge TPU coprocessor* [54]

TOPS	4
TOPS/W	2
Wymiary	65 mm x 30 mm
Połączenie	USB 3.0, USB 3.1 Gen 1
Pobór mocy	2,5 W i 4,5 W w trybie maksymalnej wydajności
Temperatura pracy	25 °C i 35 °C w trybie maksymalnej wydajności

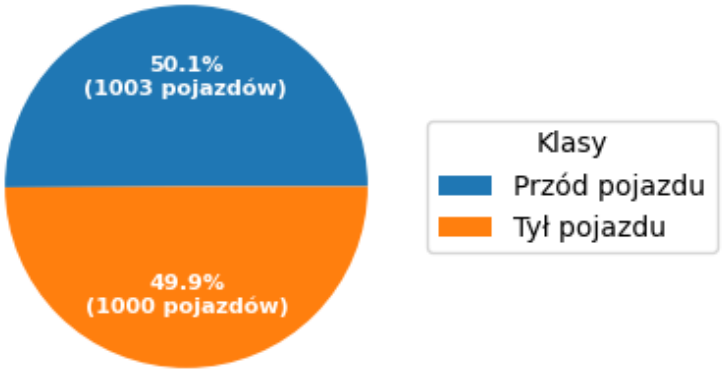
3. ZBIÓR DANYCH

W badaniu przedstawiono zbiór danych przeznaczonych do detekcji pojazdów w warunkach nocnych, zawierający obrazy i nagrania z czujnika intensywności światła. Zadanie detekcji pojazdów w nocy stanowi wyzwanie z uwagi na ograniczoną widoczność i obniżoną jakość obrazów. W celu stworzenia tego zbioru danych skorzystano z kamery Go Pro 5 oraz czujnika intensywności światła APDS-9966 [55], które są dostępne na rynku i popularne w zastosowaniach rejestrowania światła. Nagrania intensywności światła zostały przechwycone za pomocą czujnika intensywności światła podłączonego do mikrokomputera ESP 32. Ten rodzaj czujnika pozwala na pomiar poziomu natężenia światła w określonych punktach przestrzeni, co umożliwia uzyskanie informacji o intensywności światła w badanym obszarze. Adnotacja danych została wykonana w dwóch popularnych formatach: YOLO oraz Pascal VOC [56]. Format YOLO charakteryzuje się prostotą i mniejszymi rozmiarami plików, co jest korzystne w przypadku analizy dużych zbiorów danych. Z kolei format Pascal VOC jest szeroko używany w społeczności uczenia maszynowego i głębokiego uczenia, co ułatwia integrację zbioru danych z różnymi *framework'ami* i narzędziami. Wybór kamery Go Pro 5 oraz czujnika intensywności światła jako źródeł danych był związany z ich popularnością i dostępnością na rynku, co pozwala na łatwe replikowanie eksperymentów i rozszerzanie tego zbioru danych w przyszłości.

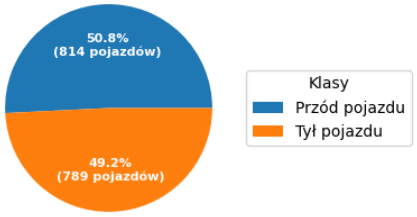
3.1. Podział danych do detekcji

Opracowany zbiór danych dedykowany detekcji pojazdów, obejmujący około 12 godzin nagrań pojazdów. Cały zbiór danych został poddany procesowi adnotacji pojazdów, w wyniku którego zaklasyfikowano 2003 pojazdy przeznaczone do detekcji. W celu identyfikacji pojazdów wykorzystano dwa rodzaje klasyfikacji: "przód"(1003 obiekty) oraz "tył"(1000 obiektów).

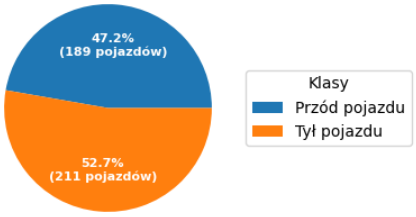
Jednym z kluczowych aspektów eksperymentu była optymalizacja podziału danych na zbiory treningowe i testowe w celu zapewnienia odpowiedniego uczenia i walidacji modelu detekcji. Zgodnie z dobrymi praktykami, zbiór danych został podzielony w proporcjach 80/20, co oznacza, że 80% danych zostało wykorzystane jako zbiór treningowy, a pozostałe 20% stanowiło zbiór testowy. Taki podział pozwala na skuteczną naukę modelu na zbiorze treningowym i jednocześnie pozwala na rzetelną ocenę wydajności modelu na nieznanych wcześniej danych testowych. Obrazy zawarte w zbiorze danych zostały uprzednio przetworzone do postaci czarno-białej i sprowadzone do proporcji 1:1 poprzez przycięcie obrazu. Obrazy zostały zmniejszone do rozdzielczości 640 pikseli. Rozdzielczość w zbiorze została ustalona na podstawie wybranych sieci do testów. Największy obraz wejściowy sieci ma 640 pikseli. Na rysunku 3.1 został przedstawiony rozkład klas w zbiorze przeznaczonym do detekcji. Obiekty zostały podzielone na dwie klasy "przód pojazdu"i "tył pojazdu".



(a) Rozkład klas w całym zbiorze danych przeznaczonym do detekcji



(b) Rozkład klas w zbiorze treningowym



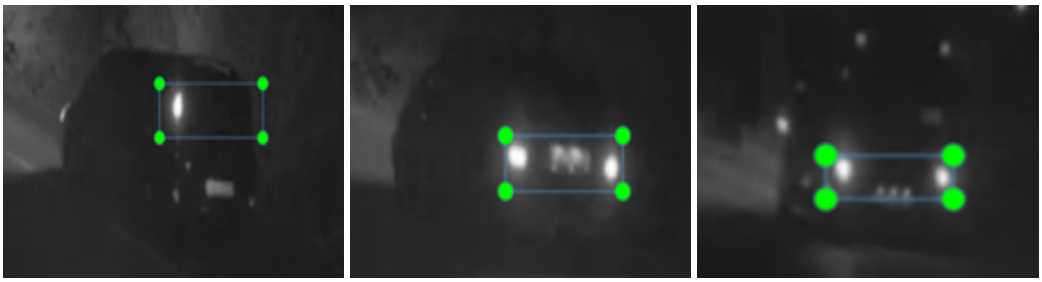
(c) Rozkład klas w zbiorze testowym

Rysunek 3.1: Rozkład klas w zbiorze

3.2. Metoda adnotacji obiektów

Wybrane obrazy zostały przeanalizowane w celu detekcji dwóch rozróżnialnych źródeł światła. W badaniu pominięto pojazdy znajdujące się daleko od kamery, w celu wyeliminowania wszelkich potencjalnych czynników zakłócających. W zbiorze znajdują się przypadki trudne w klasyfikacji. Autobusy posiadają wiele źródeł światła, co utrudnia dokładne zlokalizowanie źródła światła, które ma być oznaczone. Niektóre z pojazdów mają niesprawne oświetlenie - jedno ze świateł nie świeci lub światła mijania są ustawione pod złym kątem, oślepiając kamerę. Na uwagę zasługuje zróżnicowanie w rozmieszczeniu i kształcie geometrycznym świateł przednich w poszczególnych pojazdach. Stwierdzono również rozbieżności w poziomach luminancji oświetlenia przedniego pomiędzy różnymi pojazdami.

3.2.1. Przykłady klasy: tył pojazdu

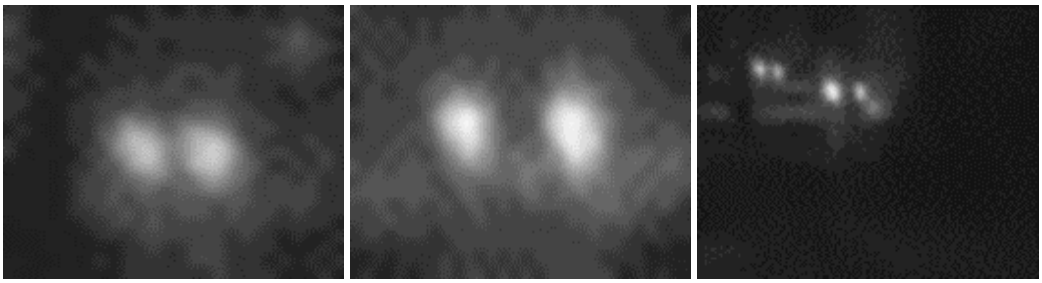


(a) Pojazd z niesprawnym światłem **(b)** Pojazd osobowy **(c)** Pojazd osobowy

(d) Ciężarówka z wieloma światłami **(e)** Ciężarówka z wieloma światłami

Rysunek 3.2: Przykładowe obrazy zaklasyfikowane jako tył pojazdu

Pojazdy z jednym niesprawnym światłem były adnotowane jako pojazd. Algorytm musi być w stanie wykryć pojazd z niesprawnym oświetleniem. Na obrazach 3.2d i 3.2e są ciężarówki z dodatkowym oświetleniem bocznym. Stanowi to dodatkowe utrudnienie dla algorytmu detekcji. Ponadto należy wspomnieć o tym, że światła są umieszczane na równych wysokościach pojazdów. Na zdjęciu 3.2e światła znajdują się bardzo nisko. Dodatkowo tylne światła pojazdu mogą się znajdować na innej wysokości niż oświetlenie tablicy rejestracyjnej. Ukazano taki przypadek na obrazie 3.2.

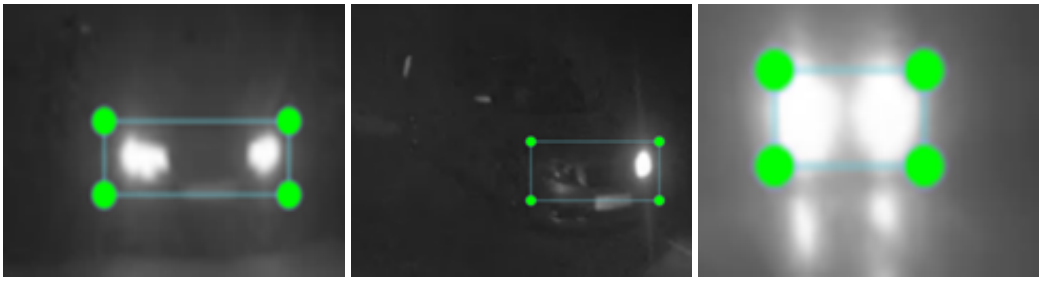


(a) Pojazd w zbyt dużej odległości **(b)** Pojazd w zbyt dużej odległości **(c)** Pojazd zbyt trudno zaklasyfikować

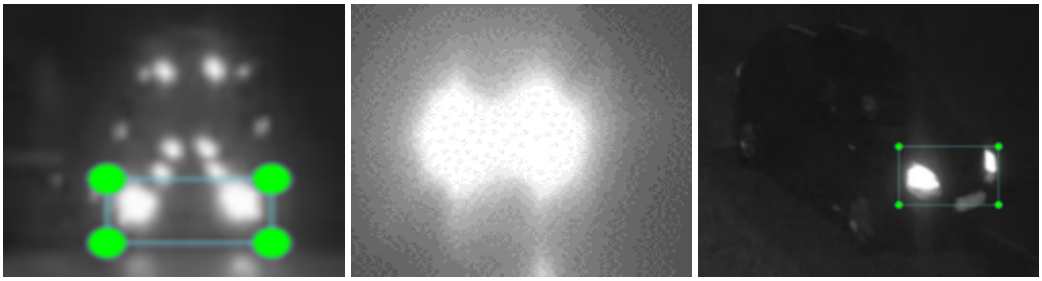
Rysunek 3.3: Przykładowe obrazy niezaklasyfikowane jako tył pojazdu

Dodatkowym utrudnieniem jest klasyfikacja tyłu pojazdów w zbyt dużej odległości. W przypadku oglądania nagrań w bardzo łatwy sposób można określić kierunek jazdy samochodu, a co za tym idzie klasy adnotacji ("przód" lub "tył"). W przypadku pojedynczego obrazu określenie czy jest to tył, czy przód pojazdu stanowi wyzwanie tak jak na rysunkach 3.3a, 3.3b i 3.3c. W związku z trudnościami w określeniu klasy pojazdu z pojedynczego zdjęcia postanowiono nie adnotować takich obiektów.

3.2.2. Przykłady klasy: przód pojazdu



(a) Przód pojazdu **(b)** Pojazd z niesprawnym światłem **(c)** Przód pojazdu - mocne światło

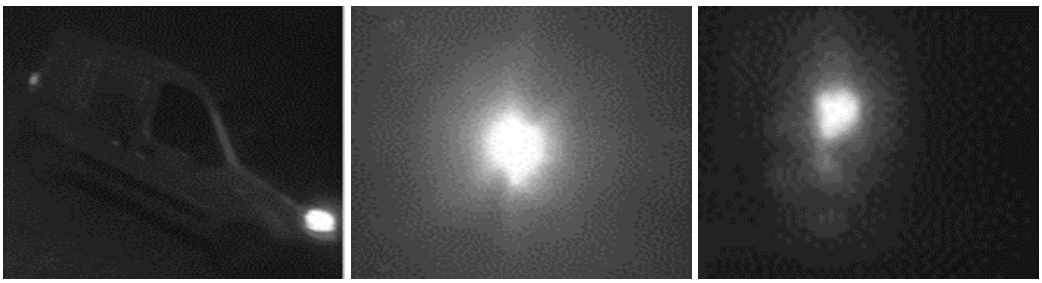


(d) Ciężarówka z wieloma światłami **(e)** Przód pojazdu - mocne światło **(f)** Przód pojazdu z bliska

Rysunek 3.4: Przykładowe obrazy zaklasyfikowane jako przód pojazdu

Światła z przodu pojazdu charakteryzują się tym, że niektóre z nich 3.4c 3.4e potrafią oślepiać kamerę i czujnik intensywności światła. Powody takiego zjawiska mogą być dwa. Po pierwsze kierowca mógł wykorzystywać światła drogowe, kiedy droga była pusta. Najczęściej zdarza się to na drogach pozamiejskich ze słabym oświetleniem. Algorytm musi być w stanie poradzić sobie z detekcją takich obiektów. Po drugie światła mijania mogą być nieprawidłowo ustawione. W zbiorze danych jest duże zróżnicowanie światła, jakie emitują pojazdy. Ponadto podobnie jak w przypadku świateł z tyłu pojazdu niektóre pojazdy mogą mieć uszkodzone jedną z żarówek 3.4b. Przypadki takie również są uznawane jako prawidłowe i adnotowane.

Niektóre pojazdy ciężarowe wyróżniają się spośród pozostałych dodatkowym oświetleniem 3.4d. W przypadku takich pojazdów obszar adnotacji obejmuje tylko światła mijania bez dodatkowego oświetlenia. Pojazdy, które są blisko urządzenia rejestrującego, wyglądają inaczej niż, te z daleka. Sylwetka pojazdu jest bardziej widoczna, a linia świateł mijania znajduje się pod większym kątem 3.4f.



(a) Pojazd zbyt blisko **(b)** Pojazd zbyt trudno zaklasyfikować **(c)** Pojazd zbyt trudno zaklasyfikować

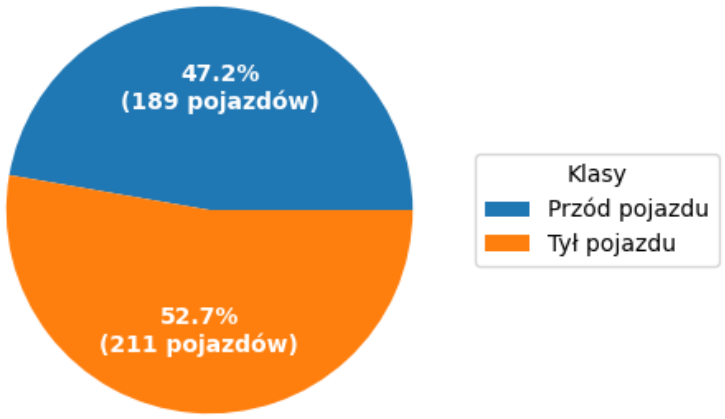
Rysunek 3.5: Przykładowe obrazy niezaklasyfikowane jako przód pojazdu

Pojazdy, które znajdują się zbyt blisko urządzenia rejestrującego jak na rysunku 3.5a nie są adnotowane. Jest to przypadek skrajny. Założeniem jest, że algorytm powinien wykryć pojazd wcześniej

niż tuż przed przejściem dla pieszych. Drugim skrajnym przypadkiem są pojazdy znajdujące się w oddali z bardzo intensywnym oświetleniem. Obiekt taki bardzo trudno zaklasyfikować. Podczas adnotacji założono, że pojazd jest klasyfikowany, dopiero kiedy widoczne są dwa punkty świetlne.

3.3. Zbiór danych do klasyfikacji obrazu

Oprócz zbioru danych do detekcji został utworzony zbiór do klasyfikacji obrazu. Zawiera on ponad 100 000 klatek z nagrań. Warto zwrócić uwagę na dwa kluczowe punkty: liczba i rodzaj adnotacji. Przy tworzeniu zbioru do detekcji obiektów, wymagane jest precyzyjne oznaczenie obiektów w obrazach, co jest zadaniem czasochłonnym i wymagającym specjalistycznego oprogramowania. Dlatego takie zbiory zazwyczaj mają mniejszą liczbę próbek w porównaniu do zbiorów klasyfikacyjnych. Natomiast w przypadku klasyfikacji, można wykorzystać prostsze adnotacje, polegające na przypisaniu kategorii lub etykiety do całego obrazu, co może być bardziej efektywne w procesie tworzenia zbioru danych. Na rysunku 3.6 przedstawiono rozkład klas w całym zbiorze danych przeznaczonym do klasyfikacji. Rozróżniane są dwie klasy "Jest pojazd" i "Nie ma pojazdu".



Rysunek 3.6: Rozkład klas w całym zbiorze danych przeznaczonym do klasyfikacji

Zbiór danych do klasyfikacji jest znacznie większy niż zbiór do detekcji. Jest tak, ponieważ adnotacja obrazów w przypadku klasyfikacji jest mniej złożona, a co za tym idzie, możliwe jest przetworzenie większej liczby obrazów w tym samym czasie. To z kolei przekłada się na większą liczbę próbek w zbiorze danych, co stanowi dużą zaletę dla algorytmów uczenia maszynowego. Wyniki uzyskane z tak dużego i różnorodnego zbioru danych mogą przynieść istotne korzyści w dziedzinie analizy obrazów. Dzięki temu, modele klasyfikacyjne mogą mieć większą zdolność generalizacji na nowe, nieznane dane.

3.3.1. Zanieczyszczenia świetlne

W niniejszej analizie przedstawiono różnorodne typy scenerii wykorzystane w zestawie danych do klasyfikacji, które mają potencjalne znaczenie dla algorytmu detekcji obiektów. Przedstawione na rysunkach obrazy 3.7a oraz 3.7b reprezentują odmienne środowiska: miejskie i wiejskie. W mieście głównie w nocy, drogi i ulice są intensywnie oświetlone, co ma na celu poprawę widoczności dla kierowców, pieszych i zapewnienie większego poczucia bezpieczeństwa. Na pierwszym rysunku 3.7a widoczna jest sceneria miejska, gdzie dominująca infrastruktura miejska oraz obecność licznych obiektów i pojazdów

może stanowić wyzwanie dla algorytmów detekcji. Kolorem czerwonym zaznaczono obiekty, które potencjalnie mogą być wykryte przez algorytm detekcji jako pojazd. Oświetlenie ma wygląd zbliżony do świateł nadjeżdżającego pojazdu, składają się z dwóch punktów światła umieszczonych poziomo obok siebie. Liczne zanieczyszczenia mogą doprowadzić do błędnej detekcji obiektu lub do złej klasyfikacji obrazu. Dodatkowo kolorem żółtym został oznaczony pojazd przejeżdżający przez skrzyżowanie. Jest to dodatkowe utrudnienie dla algorytmu, który powinien wykrywać przód lub tył pojazdu. Pojazd taki nie przejeżdża przez przejście dla pieszych i nie jest potrzebna detekcja tego obiektu.

Sceneria poza miastem jest mniej zróżnicowana. Na rysunku 3.7b widnieje droga bez oświetlenia. W pobliżu drogi był oświetlony chodnik, dzięki temu widoczna jest droga. Na pustej, pozamiejskiej drodze nie znajdują się zanieczyszczenia świetlne i trudno znaleźć elementy, które mogłyby zostać pomylone z pojazdem.

(a) Sceneria miejska **(b)** Sceneria poza miastem

Rysunek 3.7: Przykładowe obrazy różnych scenerii

4. MIARY SKUTECZNOŚCI ALGORYTMÓW

W celu zbadania sieci wyselekcjonowano metryki dla sieci detekcji i klasyfikacji. We wszystkich badanych algorytmach bardzo ważna jest równowaga pomiędzy dwiema metrykami. Pierwszą jest precyzja, która ocenia, jaka liczba pozytywnych przewidywań jest faktycznie prawdziwych. W kontekście proponowanego urządzenia precyzja jest istotna, dlatego, że jeśli na przejściach dla pieszych dodatkowe oświetlenie będzie się często zapalać, kiedy nie ma pieszego, to może to spowodować niepożądane skutki w postaci ignorowania kierowców pojawiającego się oświetlenia. Drugą jest czułość, która ocenia zdolność do prawidłowego identyfikowania pozytywnych przypadków. W kontekście proponowanego systemu, jeśli pojazd nie zostanie wykryty i nie zostanie uruchomione dodatkowe oświetlenie, to pieszy może zostać niezauważony.

4.1. Metryki dla algorytmów detekcji

Wytrenowane sieci neuronowe do detekcji zostały sprawdzone na zbiorze testowym, wykorzystując zestaw metryk COCO [57]. Wszystkie metryki oprócz sieci z rodziny YOLO zostały policzone przy pomocy biblioteki *Tensorflow*.

Warto zwrócić uwagę, że w metrykach *COCO* nie jest rozróżniane *mAP* (ang.mean average precision) i *AP* (average precision). *AP* jest średnią po wszystkich kategoriach. Tradycyjnie nazywane jest to *mAP*.

4.1.1. Intersection Over Union

Algorytmy do detekcji starają się przewidzieć z wysoką dokładnością lokalizację obiektów danej klasy na obrazie. Dokonują tego poprzez tworzenie prostokątów otaczający obiekty. Ocena predykcji algorytmu odbywa się za pomocą:

- $\mathbf{B_{g_t}}$ - prostokątów *ground-truth* - są to prostokąty tworzone przy adnotacji zbioru. Zawierają informację o położeniu obiektu i jego klasę.
- $\mathbf{B_p}$ - prostokątów, których predykcji dokonał algorytm. Składa się z 3 atrybutów, są nimi pozycja obiektu, klasa i ocena pewności predykcji, która jest zazwyczaj wartością od 0 do 1.

IOU 4.1 (ang. *Intersection Over Union*) obliczane przy pomocy wzoru 4.1.

$$J(B_p, B_{g_t}) = IOU = \frac{area(B_p B_{g_t})}{area(B_p B_{g_t})} \tag{4.1}$$

Idealna predykcja jest wtedy, kiedy oba prostokąty nachodzą na siebie idealnie. W takim wypadku wartość *IOU* jest równa 1. Jeśli prostokąty na siebie w ogóle nie nachodzą, to wartość ta wynosi 0. Oceny można dokonywać przy różnych progach *IOU*. W przypadku kiedy wartość *IOU* jest wysoka, metryka jest bardzo restrykcyjna i wymaga bardzo dokładnej predykcji lokalizacji przez algorytm. Niska wartość *IOU* oznacza, że ocena jest łagodniejsza i nie jest wymagana przez algorytm dokładna predykcja lokalizacji obiektu.

4.1.2. Precyzja i czułość

Precyzja jest zdolnością algorytmu do dokonywania tylko istotnych predykcji. Czułość z kolei jest zdolnością algorytmu do wykrycia wszystkich istotnych predykcji. W celu obliczenia jakości predykcji każdy prostokąt musi zostać zaklasyfikowany jako:

- True positive (TP) - poprawna detekcja,
- False positive (FP) - niepoprawna detekcja nieistniejącego obiektu lub niepoprawna detekcja istniejącego obiektu,
- False negative (FN) - niewykryty obiekt.

Wykorzystano wzór na precyzję 4.2 i czułość 4.3.

$$Precision = Pr = \frac{\sum_{n=1}^{S} TP_n}{\sum_{n=1}^{S} TP_n + \sum_{n=1}^{N-S} FP_n} = \frac{\sum_{n=1}^{S} TP_n}{Wszystkie\ detekcje} \tag{4.2}$$

$$Recall = Rc = \frac{\sum_{n=1}^{S} TP_n}{\sum_{n=1}^{S} TP_n + \sum_{n=1}^{G-S} FN_n} = \frac{\sum_{n=1}^{S} TP_n}{Wszystkie\ prawdziwe\ detekcje} \tag{4.3}$$

Gdzie:

- G - liczba wszystkich prostokątów ze zbioru referencyjnego,
- N - liczba wszystkich predykcji algorytmu,
- S - liczba wszystkich poprawnych predykcji algorytmu.

4.1.3. Średnia arytmetyczna precyzji i czułości

Algorytmy detekcji zwracają stopień pewności detekcji, który może być brany pod uwagę dla pozytywnych detekcji, gdzie pewność jest większa niż pewien próg τ. Detekcje, które mają niższą wartość pewności od τ są traktowane jako negatywne. Po uwzględnieniu stopnia pewności detekcji otrzymane zostały równania na precyzję 4.4 i czułość 4.5.

$$Pr(\tau) = \frac{\sum_{n=1}^{S} TP_n(\tau)}{\sum_{n=1}^{S} TP_n(\tau) + \sum_{n=1}^{N-S} FP_n(\tau)} = \frac{\sum_{n=1}^{S} TP_n(\tau)}{Wszystkie\ detekcje(\tau)} \tag{4.4}$$

$$Rc(\tau) = \frac{\sum_{n=1}^{S} TP_n(\tau)}{\sum_{n=1}^{S} TP_n(\tau) + \sum_{n=1}^{G-S} FN_n(\tau)} = \frac{\sum_{n=1}^{S} TP_n(\tau)}{Wszystkie\ prawdziwe\ detekcje} \tag{4.5}$$

W celu obliczenia średniej precyzji (ang. *average precision*) na początku trzeba obliczyć K wartości dla różnych pewności τ(k) 4.6:

$$\tau(k), k = 1, 2, 3..., K\ gdzie\ \tau(i) > \tau(j)\ dla\ i\ > j \tag{4.6}$$

Posortowane wartości metryki dokładności obliczone zostały wzorem 4.7.

$$R_r(n), n = 1, 2, 3...K\ gdzie\ R_r(m) < R_r(n)\ dla\ m > n \tag{4.7}$$

Następnie na parze precyzji i czułości dokonywana jest interpolacja w taki sposób, że krzywa jest monotoniczna. Równanie zostało przedstawione we wzorze 4.8.

$$Pr_i(R) = max_{k|Rc(\tau(k))\geq R} Pr(\tau(k)) \tag{4.8}$$

Średnia precyzja (ang. *average precision*) 4.9 obliczana jest przy pomocy wzoru 4.9.

$$AP = \sum_{k=0}^{K} [R_r(k) - R_r(k+1)] Pr_i(R_r(k)) \tag{4.9}$$

Średnia precyzja jest otrzymywana indywidualne dla każdej klasy. W dużych zbiorach z wieloma klasami przydatnym jest mieć jedną uniwersalną metrykę uwzględniającą czułość dla wszystkich klas.

W tym celu wykorzystywane jest średnia arytmetyczna precyzji (ang. *mean average precision*) *mAP* wyrażona wzorem 4.10.

$$mAP = \frac{1}{C} \sum_{i=0}^{C} AP_i \tag{4.10}$$

Następną metryką jest średnia czułość (ang. *Average Recall*). Miara nie uwzględnia pewności predykcji τ. Metryka uwzględnia wszystkie wartości w przedziale *IOU* [0,5, 1]. W ogólności metryka wyrażona jest wzorem 4.11.

$$AR = \int_{0,5}^{1} Rc_{IOU}(IOU)\, do \tag{4.11}$$

W metrykach COCO wykorzystana została inna wersja tej metryki. AR (ang. *Average Recall*) jest średnią czułości na różnych progach *IOU*. Progi 4.12 zostały obliczone wzorem 4.12.

$$t(o), o = 1, 2, 3..., O \tag{4.12}$$

Średnia czułość w metrykach COCO wyrażona jest wzorem 4.13.

$$AR = \frac{1}{O} \sum_{o=1}^{O} max_{k|Pr_{t(o)}(\tau(k))>0}(Rc_{t(o)}(\tau(k))) \tag{4.13}$$

4.1.4. Wybrane metryki

W badaniu zostały metryki, które zdaniem autora są bardziej istotne od pozostałych.

- *mAP* (Mean Average Precision),
- *mAP@.50IOU* - Mean Average Precision at 50% Intersection over Union
- *mAP (small)*,
- AR@1 - *Average Recall* z jedną detekcją,

Miara *AP* została wykorzystana, ze względu na to, że uchodzi za uniwersalną miarę. Wybrane sieci we *framework'u Tensorflow* [58] zostały we wstępnie przetrenowane na zbiorze "COCO 2017"i porównywane są przy pomocy tej metryki. Jest to najważniejsza metryka w "COCO Challenge"[59]. Metryka ta jest średnią wszystkich 10 progów - .50:.05:.95 i 80 kategorii. W badaniu występują 2 klasy, dlatego będzie to średnia po 2.

Miara *mAP@.50IOU* została wybrana dlatego, że jest najważniejszą metryką w "PASCAL VOC Challenge"2007, 2010, 2012 [56]. Metryka ta liczy średnią precyzję dla obiektów z 50% *IOU* (Intersection Over Union). Detekcja uznawana jest za pozytywną, jeśli ramka predykcji pokrywa się przynajmniej w 50% z ramką z adnotacji.

mAP (small) zostało wykorzystane dlatego, że według założeń badania najlepiej jak by samochody były wykrywane jak najwcześniej. Metryka ta określa, z jaką precyzją wykrywane są małe obiekty, a na obrazach przeważnie są to samochody znajdujące się daleko od kamery i przejścia dla pieszych.

4.2. Metryki dla klasyfikatorów

Metryki dla klasyfikatorów różnią się od metryk dla algorytmów detekcji. Każda predykcja algorytmu jest porównywana z wartością prawdziwą (ang. *ground truth*) i oceniana jest jako:

- TP (ang. *True Positive*) - Wartość predykcji algorytmu jest pozytywna i jest taka sama jak wartość prawdziwa.

- TN (ang. True Negative) - Wartość predykcji algorytmu jest negatywna i jest zgodna z prawdą.
- FP (ang. *False Positive*) - Wartość predykcji algorytmu jest pozytywna, ale jest błędna, co oznacza, że algorytm błędnie wykrył obiekt lub zjawisko, które w rzeczywistości nie istnieje. Jest to błąd typu I.
- FN (ang. *False Negative*) - Wartość predykcji algorytmu jest negatywna, ale jest błędna, co oznacza, że algorytm nie wykrył obiektu lub zjawiska, które istnieje w rzeczywistości. Jest to błąd typu II.

Miara dokładności ocenia ogólną poprawności modelu klasyfikacyjnego. Określa, jak wiele z całkowitej liczby przewidywań modelu jest poprawnych. Wyższa dokładność oznacza, że model częściej dokonuje poprawnych klasyfikacji, ale może być myląca, gdy dane są nierównomiernie rozkładane między różnymi klasami. Dokładność (ang. *Accuracy*) 4.14 obliczana jest wzorem 4.14.

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{4.14}$$

Miara precyzji (ang. *Precision*) ocenia dokładność pozytywnych przewidywań dokonywanych przez model. Odpowiada na pytanie, ile z pozytywnych przewidywań jest faktycznie prawdziwych. Wysoka precyzja oznacza, że model rzadko daje fałszywe pozytywne wyniki, co jest ważne w przypadkach, gdy błędy typu I (fałszywie pozytywny) są kosztowne lub nieakceptowalne. Precyzja (ang. *Precision*) 4.15 obliczana jest wzorem 4.15.

$$Precision = \frac{TP}{TP + FP} \tag{4.15}$$

Miara czułości (ang. *Recall*), znana również jako współczynnik prawdziwie pozytywnych (ang. *True Positive Rate*), mierzy zdolność modelu do prawidłowego zidentyfikowania pozytywnych przypadków. Odpowiada na pytanie, ile z faktycznych pozytywnych przypadków model jest w stanie wykryć. Wysoka czułość jest ważna w sytuacjach, gdzie błędy typu II (fałszywie negatywny) są kosztowne lub niebezpieczne, ponieważ oznacza to, że model jest skuteczny w wykrywaniu pozytywnych przypadków. Czułość (ang. *Recall*) obliczana jest wzorem 4.16.

$$Recall = \frac{TP}{TP + FN} \tag{4.16}$$

W niektórych przypadkach została zastosowana miara F1, która jest harmoniczną średnią precyzji i czułości. Jest szczególnie przydatna w sytuacjach, gdzie zależy nam na równoważeniu zdolności modelu do identyfikowania zarówno pozytywnych, jak i negatywnych przypadków. Działa dobrze w przypadkach, gdzie rozkład klas w danych jest niezbalansowany, co oznacza, że jedna z klas występuje znacznie częściej niż druga. Miara F1 obliczana jest wzorem 4.17.

$$F1 = \frac{2 \cdot Precision \cdot Recall}{Precision + Recall} \tag{4.17}$$

5. OPIS ZASTOSOWANYCH METOD WYKRYWANIA OBIEKTÓW

W literaturze naukowej rozważane były różnorodne metody wykrywania obiektów, a przed erą dominacji sieci neuronowych zastosowano wiele innych podejść, które niosły ze sobą pewne istotne ograniczenia. Jednym z przykładów jest podejście oparte na modelach deformowalnych części (ang. *DPM - deformable parts models*) [60]. W metodzie tej, dla detekcji obiektów wykorzystywano technikę okna przesuwnego, która polega na analizie różnych fragmentów obrazu w poszukiwaniu obiektu. Niestety, choć *DPM* wykazywało pewne sukcesy w wykrywaniu obiektów, charakteryzowało się ono znacznym ograniczeniem, które stanowiło poważną przeszkodę w zastosowaniach wymagających detekcji w czasie rzeczywistym - była to jego niewystarczająca szybkość działania.

W celu osiągnięcia wydajniejszych detekcji w czasie rzeczywistym zaczęto poszukiwać alternatywnych technik, które umożliwiłyby efektywne wykrywanie obszarów, na których znajdują się obiekty. W wyniku tych poszukiwań obecnie przyjęło się uznawać głębokie sieci neuronowe za najskuteczniejszą metodę wykrywania obiektów na obrazach. Architektury tych sieci najczęściej składają się z jednej lub dwóch głównych części, które następnie są odpowiednio ze sobą łączone, tworząc zaawansowany algorytm detekcji obiektów.

Warto zwrócić uwagę, że skuteczność głębokich sieci neuronowych wynika z ich zdolności do automatycznego wyodrębniania cech charakterystycznych dla obiektów. W trakcie uczenia się sieci te są w stanie wykształcić abstrakcyjne reprezentacje, które odpowiadają za detekcję obiektów o różnych kształtach, wielkościach czy kontekstach. To sprawia, że głębokie sieci neuronowe są znacznie bardziej elastyczne i wszechstronne w porównaniu do wcześniejszych metod, takich jak *DPM*.

5.1. Architektury jednoelementowe

Przykładem architektury jednoelementowej jest rodzina sieci YOLO [61, 62]. Metoda ta nie wykorzystuje propozycji regionów. Sieć analizuje cały obraz i oblicza prawdopodobieństwa, że obiekt znajduje się w danej części obrazu. Ten algorytm różni się od SSD (ang. *Single Shot Detector*) tym, że wykorzystuje dwie warstwy w pełni połączone zamiast sieci splotowych o różnych rozmiarach. Sieć została stworzona z myślą o detekcji obiektów w czasie rzeczywistym. Kosztem wydajności sieć popełnia więcej błędów lokalizacji obiektów, natomiast mało prawdopodobne są wyniki fałszywie pozytywne. W badaniu zostały sprawdzone sieci YOLOV8 [63].

Podobną ideą kierowali się twórcy EfficientDet [64]. Sieć została stworzona z myślą o zastosowaniach w robotyce i autonomicznych pojazdach. Sieć charakteryzuje się dużą wydajnością pod względem operacji zmiennoprzecinkowych w stosunku do osiąganej dokładności. Dla porównania najmniejszą siecią w zestawieniu, pomijając sieci EfficientDet jest YOLOV8n. Podczas detekcji musi zostać wykonane 8.7 MFLOPS, co jest wartością 4-krotnie większą od EfficientDet D0, która musi wykonać 2.5 MFLOPS. Jednocześnie ta sieć osiąga lepszy wynik na zbiorze COCO test-dev [59].

Po zapoznaniu się z literaturą do badania zostały wybrane dwie różne architektury jednoelementowe:

- YOLOV8,
- EfficientDet.

5.2. Architektury dwuelementowe

Częstym przypadkiem jest, że sieci składają się z dwóch elementów. Architektury takie składają się z części przeznaczonej do ekstrakcji cech, która zazwyczaj wykorzystuje warstwy splotowe. Wynikiem jest zbiór cech z obrazu, który następnie przekazywany jest do drugiej części architektury przeznaczonej do detekcji obiektu. Ta sieć zwraca wynik końcowy algorytmu detekcji, którym są współrzędne obiektu, klasę, oraz stopień pewności detekcji. Takie rozwiązanie pomaga na wybranie konkretnych technologii w zależności od potrzeb.

5.2.1. Metody ekstrakcji cech

MobileNetV2 [65] został stworzony specjalnie z myślą o urządzeniach mobilnych i o ograniczonej wydajności. Znacznie obniża wykonywaną ilość operacji i pamięci przy jednoczesnym zachowaniu dokładności. Metoda bazuje na odwróconym połączeniu resztkowym (ang. *inverted residual structure*), gdzie połączenia resztkowe są wykorzystywane w połączeniach pomiędzy wąskimi gardłami. Pośrednia warstwa rozszerzająca wykorzystuje splot w kierunku głębokości (ang. *depthwise convolution*) do filtrowania cech jako źródła nieliniowości. Jako całość, architektura MobileNetV2 zawiera 32 warstwy splotowe i 19 splotów w kierunku głębokości. Ma około 5 razy mniej parametrów względem architektury MobileNetV1 [66] jednocześnie osiągając bardzo zbliżone wyniki. Sieć została wybrana do dalszych badań.

HourGlass [67] jest siecią o innym przeznaczeniu. Jest głównie wykorzystywana do estymacji pozy. Została wykorzystania w badaniu w celu sprawdzenia, jak algorytm o innym przeznaczeniu sprawuje się na tle innych bardziej wyspecjalizowanych.

W badaniu zostały wykorzystane następujące algorytmy ekstrakcji cech:

- MobileNetV2,
- ResNet50V1,
- EfficientDet,
- HourGlass104.

5.2.2. Metody detekcji

Metoda R-CNN [68] wykorzystuje propozycje regionów. Metoda wyszukiwania selektywnego wyszukuje 2000 propozycji regionów, na których mogą znajdować się obiekty. Proponowane regiony są następnie przekazywane do sieci splotowej w celu ekstrakcji cech, które następnie są klasyfikowane przy pomocy SVM. Cały algorytm był innowacyjny w swoim czasie, jednak cały proces wyszukiwania obiektów jest zbyt czasochłonny, żeby można było go wykorzystać w czasie rzeczywistym na urządzeniu w budowanym. Z tego powodu ta metoda została wykluczona z badań. Fast-RCNN [69] jest ulepszoną metodą względem poprzednika (R-CNN). Według autorów sieć przetwarza obrazy 146 razy szybciej niż *R-CNN* bez redukcji wielowymiarowości przy pomocy *SVD* [70] i 213 razy szybszy z wykorzystaniem redukcji wielowymiarowości. *R-CNN* jest wolne, dlatego, że sieć splotowa wykonuje *forward pass* dla każdej propozycji obiektu bez współdzielenia obliczeń. Fast-R-CNN wykonuje stosuje metodę *SPPnet* (ang. *Spatial Pyramid Pooling Networks*) [71]. Metoda ta dokonuje ekstrakcji cech przy pomocy sieci splotowej dla każdego obrazu, a następnie klasyfikuje każdą propozycję obiektu, wykorzystując wektor cech ze wspólnej mapy cech. Pomimo dużej poprawy w szybkości działania algorytm nadal jest niewystarczająco szybki i również nie jest wzięty pod uwagę w badaniu.

W obu poprzednich algorytmach jest wykorzystywana metoda selektywnego przeszukiwania (ang. *selective search*), która jest czasochłonna i wpływa negatywnie na szybkość detekcji W Faster-RCNN [72] Podobnie jak Fast-RCNN obraz przetwarzany jest przez sieć splotową w celu ekstrakcji cech. Następnie zamiast wykorzystania metody selektywnego przeszukiwania zastosowana jest osobna sieć neuronowa, która uczy jak dokonywać predykcji regionów obrazu, które ma proponować. Zastosowanie tej metody pozwala uzyskać 10-krotnie szybszą detekcję. Algorytm ten został wykorzystany w badaniu.

SSD – Single Shot Multidetector [73] został stworzony z myślą o urządzeniach wbudowanych. Twórcy twierdzą, że może mieć też zastosowanie w komputerach o dużej wydajności, dla których inne metody detekcji nie działają wystarczająco szybko, żeby działać w czasie rzeczywistym (R-CNN i Fast-R-CNN). Algorytm dokonuje dyskretyzacji regionów wyjściowych w zbiór regionów domyślnych mających różne proporcje i wymiary dla każdej mapy cech. W czasie predykcji sieć generuje wyniki dla każdej klasy w domyślnym regionie i stosuje ulepszenia w celu lepszego dopasowania do kształtu obiektu. Dodatkowo sieć wykorzystuje cechy z wielu map cech z różnymi rozdzielczościami w celu dobrego wykrywania obiektów o różnych rozmiarach. SSD jest metodą prostszą względem metod wykorzystujących propozycje regionów ze względu na to, że nie wykonuje propozycji regionów. Zachowując stosunkowo wysoką dokładność sieci, uzyskano znacząco lepsze wyniki w wydajności. W badaniu proponującym dane rozwiązanie sieć osiągnęła 59 klatek na sekundę w porównaniu do sieci referencyjnej Faster-R-CNN, która osiągnęła 7 klatek na sekundę podczas testu na zbiorze VOC2007 [56].

Wraz z *SSD* mogą być wykorzystane równolegle *FPN* (ang. *Feature Pyramid Networks*) [74]. Sieć ta pomaga zwiększyć precyzję i dokładność przy detekcji obiektów. Tworzą one piramidę cech na wielu warstwach, wykorzystując *forward-pass*. Warstwy wysokiej rozdzielczości są szczególnie istotne przy wykrywaniu obiektów o małych rozmiarach.

Kolejną metodą wykorzystaną w badaniu jest CenterNet [75]. Jest to metoda, która przewiduje punkty kluczowe przy pomocy sieci splotowej. Punkty te są wykorzystywane do tworzenia regionów wokół punktu i ich klasyfikacji. Sieciami wykorzystującymi podobne podejście jest CornerNet [76] i Grid-RCNN [77]. Metoda osiąga bardzo wysokie wyniki w dokładności i precyzji detekcji. Jednocześnie algorytm nie jest czasochłonny.

Po zapoznaniu się z literaturą do badania zostały wybrane 4 różne metody detekcji:

- SSD,
- CenterNet,
- Faster RCNN,
- FPN.

6. WYNIKI TRENINGU I TESTÓW

6.1. Platforma testowa

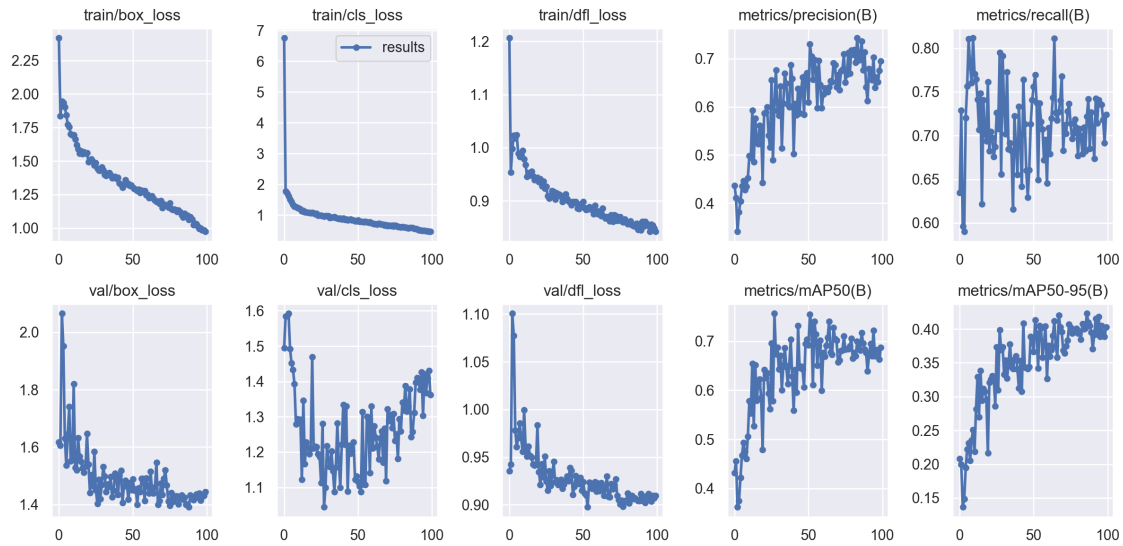
W ramach przeprowadzonych testów eksperymentalnych konieczne było użycie specyficznej platformy testowej w celu treningu oraz oceny wydajności algorytmów. Wszystkie treningi oraz część testów zostały przeprowadzone na komputerze mobilnym z systemem operacyjnym Windows 11 w wersji 22H2. W ramach tego środowiska, wykorzystany został procesor produkowany przez firmę AMD Ryzen 5800H. Ważnym elementem tej konfiguracji testowej była również karta graficzna. W celu przeprowadzenia testów wykorzystano kartę graficzną Nvidia modelu 3070M. Aby umożliwić przeprowadzenie eksperymentów wykorzystujących obliczenia na kartach graficznych, zainstalowana została biblioteka Nvidia CUDA w wersji 11.2. CUDA to popularna platforma obliczeń równoległych, która umożliwia wydajne wykorzystanie zasobów kart graficznych w celu przyspieszenia różnorodnych zadań numerycznych. W ramach tych testów głównym narzędziem programistycznym wykorzystanym do implementacji i analizy algorytmów była Python w wersji 3.9. Aby móc korzystać z zaawansowanych funkcjonalności oraz algorytmów, wykorzystano kilka kluczowych bibliotek. Jednymi z głównych były *PyTorch* w wersji 2.0.0+cu177 oraz *Tensorflow* w wersji 2.10. Dodatkowo wykorzystano bibliotekę *Tensorflow* models w wersji 2.11.1, co umożliwiło dostęp do zaawansowanych modeli i architektur, które mogły być użyte do porównania i analizy w ramach przeprowadzonych eksperymentów. W tabeli 6.1 zaprezentowano specyfikację platformy wykorzystanej do treningu i części testów sieci neuronowych.

Tabela 6.1: Platforma sprzętowa i wersje bibliotek

Parametr	Opis
System operacyjny	Windows 11 22H2
CUDA	11.2
Karta graficzna	NVIDIA 3070M
Procesor	AMD Ryzen 5800H
Python	3.9
PyTorch	2.0.0+cu177
TensorFlow	2.10
TensorFlow models	2.11.1

6.2. Przebieg treningu i ewaluacji

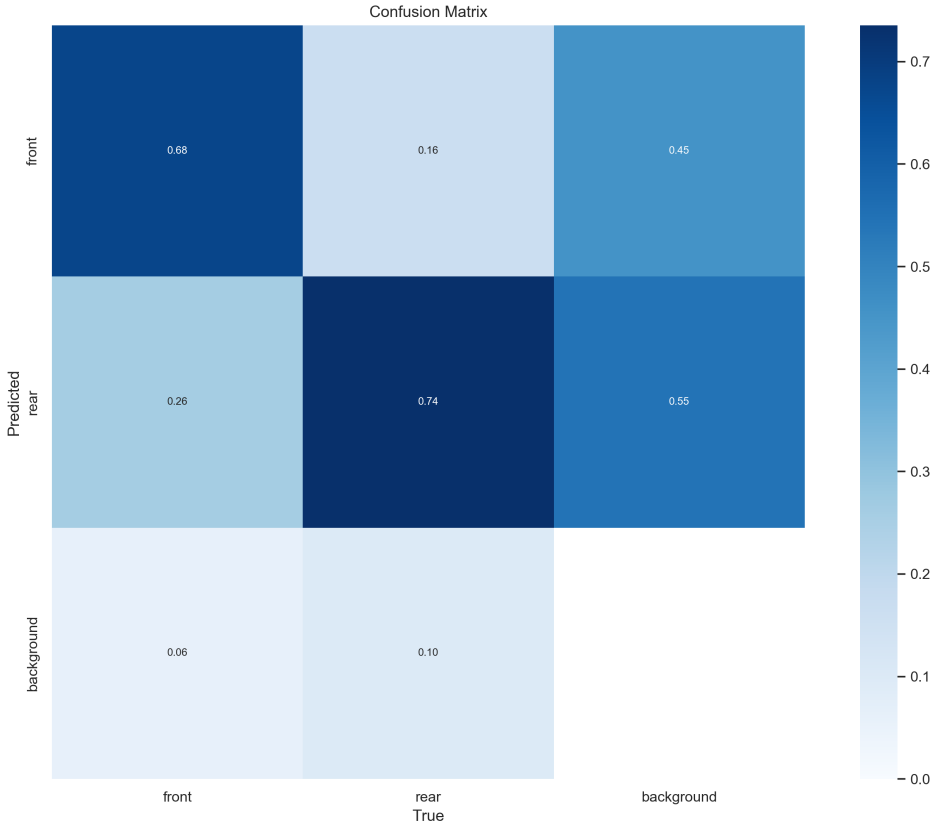
Przeprowadzono badanie na 11 sieciach. Najlepsze wyniki dla najważniejszej metryki według COCO - *mAP* osiągnęła sieć YOLOV8s. Najlepsze metryki według PASCAL VOC (*mAP@.50IOU*) osiągnęła sieć YOLOV8n. Wyniki obu sieci YOLO są do siebie bardzo zbliżone. Sieć YOLOV8n bardzo szybko osiąga wysokie *mAP* podczas treningu. Już w 38 epoce, czyli 3140 kroków model osiągnął wynik testowy bardzo bliski maksymalnemu - *mAP@.50IOU* 0,727 i *mAP* 0,401. Pomimo bardzo krótkiego treningu wynik ten jest zdecydowanie wyższy od większości sieci, które badano. Pozostałe metryki nie zostały zmierzone dla sieci YOLO, ze względu na brak dostępności ich w bibliotece *Ultralytics*. Na rysunku 6.1 przedstawiony został przebieg treningu i ewaluacji dla najlepszej sieci.



Rysunek 6.1: Przebieg treningu i ewaluacji dla YOLOV8s

Trening sieci YOLOV8n na rysunku 6.1 przebiega bardzo szybko. Przed 50 epoką sieć osiąga bardzo wysokie wyniki ewaluacji. Dalszy trening skutkuje większym dopasowaniem się do danych i nie przynosi lepszych rezultatów.

Na rysunku 6.2 przedstawiona została macierz pomyłek. Można zauważyć, że najczęstszym błędem w przypadku sieci YOLOV8n jest pomyłka z tłem obrazu, czyli nieprawidłowo oznaczony obiekt.



Rysunek 6.2: Macierz pomyłek dla YOLOV8s

Na rysunku 6.3 zaprezentowano przebiegi testów walidacyjnych podczas treningu sieci z *framework'a Tensorflow*. Najlepsze wyniki osiągnęła sieć CenterNet MobileNetV2. Można zauważyć, że najlepsze wyniki były osiągane w okolicy 60 tysięcy kroków.

Sieć MobileNetV2 FPNLite 640 osiąga najlepsze wyniki przy najmniejszej ilości kroków. Przed 20 tysiącami kroków sieć została wytrenowana i dalszy trening nie dał lepszych rezultatów w metryce *mAP*.

W okolicy 54 tysięcy kroków w sieci Faster RCNN ResNet50 nastąpiło pogorszenie wartości metryki *mAP* w testach walidacyjnych. Najwyższa wartość została osiągnięta po 66-ciu tysiącach kroków. Te sieci nie 100 tysięcy kroków, aby osiągnąć najwyższą wartość. Po osiągnięciu najwyższego wyniku walidacyjnego sieć uzyskiwała dobre rezultaty w metryce, ale nie osiągano wyższej wartości.

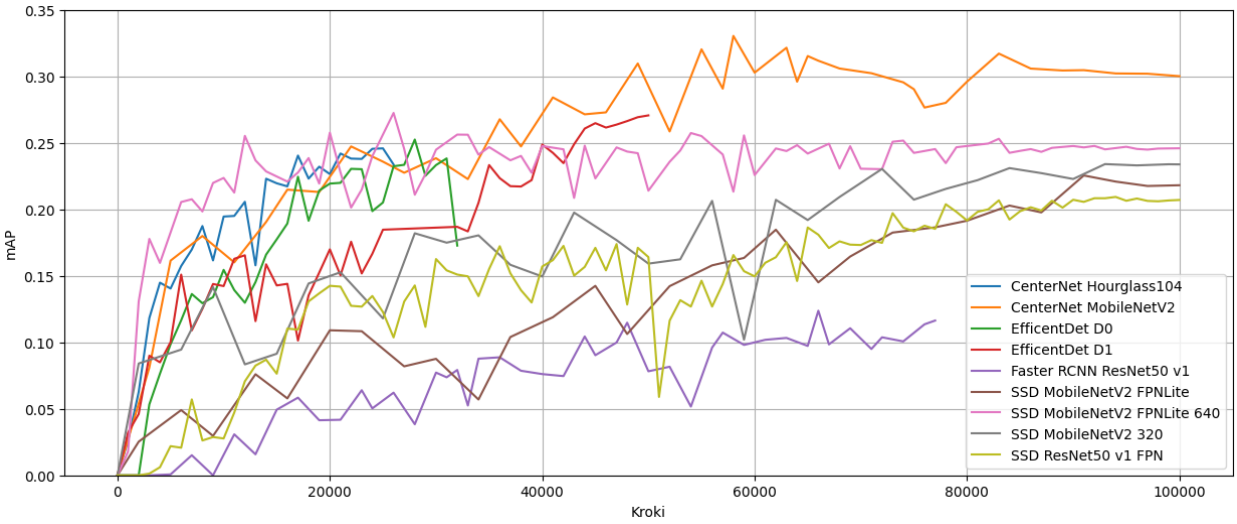
Podczas treningu sieci SSD ResNet50 v1 FPN można zauważyć, że w okolicy 50 tysięcy kroków nastąpiło znaczne pogorszenie sieci w metryce *mAP*. Dalszy trening poskutkował polepszeniem wyników. Do około 70 tysięcy kroków występowała duża wariancja podczas ewaluacji metryką *mAP*. Pod koniec treningu można zauważyć bardzo małe zmiany w metryce *mAP*.

Sieć FasterRCNN ResNet 50 v1 osiągnęła najgorsze wyniki podczas treningu. Przyrost dokładności względem kroków był najmniejszy.

Sieć EfficientDet D0 była trenowana na 50 tysiącach kroków. Czas trwania treningu sieci wynosił prawie 15 godzin (liczony czas nie uwzględnia walidacji i ładowania się zbioru danych).

Na wykresie można zauważyć, że wysokie wyniki w metryce *mAP* na początku treningu mogą oznaczać wysokie wyniki w metryce *mAP* pod koniec treningu.

Pod koniec treningu współczynnik uczenia się zmniejsza i można zauważyć, że przyrost *mAP* jest wtedy najmniejszy.



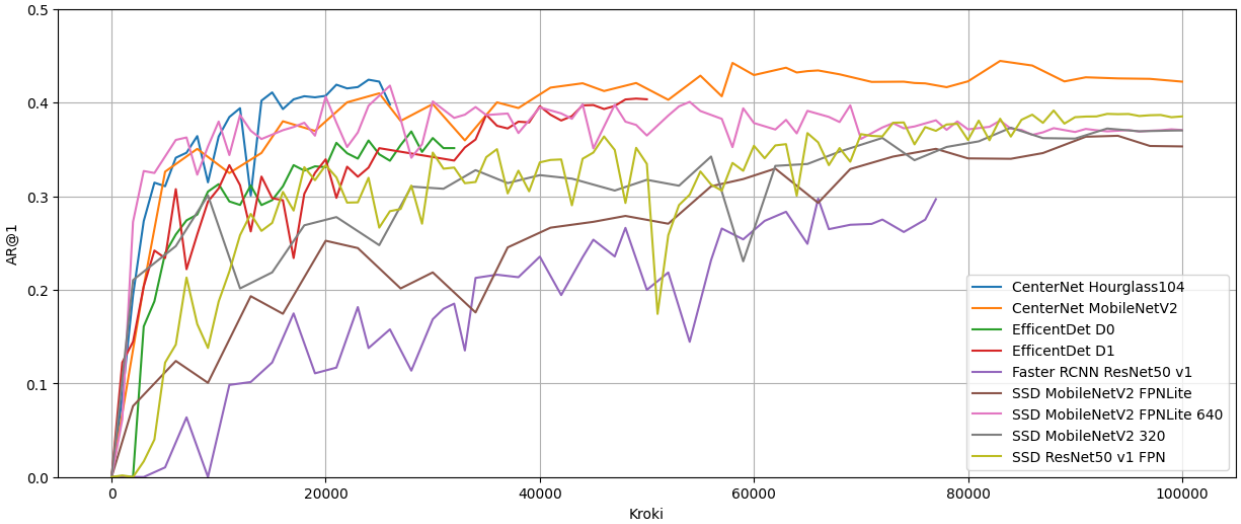
Rysunek 6.3: Przebiegi testów walidacyjnych podczas treningu sieci z *framework'a Tensorflow* - na osi pionowej została ukazana metryka *mAP*, a na osi poziomej liczbę kroków

Na rysunku 6.4 zaprezentowano przebiegi testów walidacyjnych podczas treningu sieci z *framework'a Tensorflow*. Najlepsze wyniki w czułości osiągnęła ta sama sieć - CenterNet MobileNetV2.

Sieci MobileNetV2 FPNLite osiągnęły bardzo zbliżone wyniki. Sieć o większej rozdzielczości osiągnęła lepszy wynik w metryce AR@1.

We wszystkich sieciach został zachowany balans pomiędzy precyzją a czułością. Żadna z sieci nie ma bardzo zawyżonej jednej metryki kosztem drugiej.

Wszystkie sieci oprócz Faster RCNN Resnet50 v1 osiągnęły wyniki wyższe niż wartość 0,2 w metryce AR@1



Rysunek 6.4: Przebiegi testów walidacyjnych podczas treningu sieci z *framework'a Tensorflow* - na osi pionowej została ukazana metryka AR@1, a na osi poziomej liczbę kroków

6.3. MobileNetV2 FPNLite - wpływ rozmiaru batch'a na trening

Na przykładzie sieci MobileNetV2 FPNLite zostało sprawdzone jak rozmiar *batch'a* wpływa na trening i wyniki ewaluacji. Możemy zauważyć, że najmniejszy czas treningu został osiągnięty przy rozmiarze 16. Wnioski z badania potwierdziły, to co jest stosowane w praktyce - nie warto trenować sieć, kiedy w *batch'u* jest tylko jeden obraz. Czas treningu jest bardzo długi a wyniki końcowe niesatysfakcjonujące. W ramach badania nie był on zmieniany. Wyniki reszty sieci są bardzo zbliżone do siebie. Wnioski z badania są takie, że rozmiar *batch'a* powinien być możliwie największy, uwzględniając ilość pamięci w karcie graficznej. Sieć, która była trenowana z *batch'em* o rozmiarze 32 wymagała zbyt dużo pamięci i trening trwał dłużej niż przy 16 *batch'a*ch.

W przedstawionej tabeli zestawiono wyniki różnych metryk oceny wydajności modelu detekcji obiektów w zależności od rozmiaru *batch'a* oraz liczby kroków podczas treningu. Tabela zawiera informacje na temat różnych metryk precyzji i czułości. Analiza wyników *mAP* w zależności od rozmiaru *batch'a* i liczby kroków wykazała, że dla większych wartości rozmiaru *batch'a* (8, 16, 32) uzyskiwano wyniki średniej precyzji nieznacznie niższe w porównaniu do mniejszych *batch'y* (1, 2, 4). Podobne trendy zaobserwowano w przypadku wyników *mAP* dla obiektów średniej wielkości i małych. Jednak warto zwrócić uwagę, że wyższe wartości *mAP* (medium) odnotowano dla *batch'y* o rozmiarze 4 i 8 w porównaniu do mniejszych *batch'y*. Wyniki *mAP* dla wartości IOU równej 0,50 i 0,75 wykazały podobne tendencje, gdzie większe rozmiary *batch'a* osiągały nieznacznie niższe wyniki w porównaniu do mniejszych *batch'y*. Analiza średnich wskaźników czułości (AR@1, AR@10, AR@100, AR@100 (medium) i AR@100 (small)) wykazała, że większe rozmiary *batch'a* zazwyczaj osiągały nieco niższe wyniki, ale różnice nie były znaczące. Ostatecznie wydaje się, że rozmiar *batch'a* nie ma znaczącego wpływu na wskaźniki czułości.

Ważnym aspektem w tabeli jest również przedstawienie czasu treningu dla różnych rozmiarów *batch'ów*. Zauważono, że dla *batch'a* o rozmiarze 1 czas treningu był długi a wyniki najniższe. Taki rozmiar *batch'a* nie jest stosowany w praktyce i badanie potwierdza bardzo niskie wyniki. Wraz ze wzrostem rozmiaru *batch'a* czas treningu się skraca. To sugeruje, że wykorzystanie większych rozmiarów

batch'ów może być bardziej efektywne pod względem czasowym. Wnioski z badania są takie, że rozmiar *batch'a* powinien być możliwie największy, uwzględniając ilość pamięci w karcie graficznej. Sieć, która była trenowana z *batch'em* o rozmiarze 32 wymagała zbyt dużo pamięci i trening trwał dłużej niż przy 16 *batch'a*ch. Wydaje się, że dla tej konkretnej architektury optymalnymi rozmiarami *batch'y* są wartości od 4 do 8, które osiągają dobre wyniki zarówno w precyzji, jak i czasie treningu. W tabeli 6.2 zaprezentowano wyniki ewaluacji sieci MobileNetV2 FPNLite.

Tabela 6.2: Wyniki ewaluacji sieci MobileNetV2 FPNLite 320x320

metryka\batch	1	2	4	8	16	32
Kroki [k]	200	100	50	25	12,5	6,25
mAP	0,117	0,222	0,221	0,201	0,214	0,207
mAP (medium)	0,218	0,361	0,370	0,344	0,372	0,326
mAP (small)	0,115	0,214	0,215	0,192	0,204	0,200
mAP@.50IOU	0,298	0,500	0,503	0,487	0,496	0,474
mAP@.75IOU	0,072	0,172	0,126	0,104	0,142	0,140
AR@1	0,276	0,364	0,360	0,320	0,348	0,330
AR@10	0,399	0,454	0,464	0,426	0,445	0,431
AR@100	0,440	0,473	0,489	0,450	0,468	0,457
AR@100 (medium)	0,596	0,651	0,648	0,579	0,592	0,564
AR@100 (small)	0,425	0,457	0,473	0,437	0,456	0,446
Czas [min]	158	102	62	49	47	69

6.4. MobileNetV2 FPNLite - wpływ liczby kroków na trening

W procesie treningu sieci neuronowych, w tym przypadku sieci MobileNetV2 FPNLite, istotnym aspektem jest odpowiednie zarządzanie czasem trwania treningu oraz unikanie nadmiernej ekspansji modelu w stosunku do danych treningowych. Przerwanie treningu w odpowiednim momencie ma kluczowe znaczenie, ponieważ może pomóc w minimalizacji zjawiska znanego jako przeuczenie (ang. *overfitting*) oraz umożliwić oszczędność czasu obliczeniowego. Przeuczenie jest powszechnym problemem w uczeniu maszynowym, w którym model nauki zbyt dokładnie dopasowuje się do zbioru treningowego, przez co traci zdolność do skutecznego generalizowania na danych testowych. W przypadku sieci MobileNetV2 FPNLite zbyt długi trening, przekraczający optymalny punkt, może prowadzić do nadmiernego dopasowania do zbioru treningowego, co wpływa negatywnie na wyniki na danych testowych. Zgodnie z wynikami ewaluacji przedstawionymi w tabeli, optymalną liczbę kroków treningowych dla sieci MobileNetV2 FPNLite jest 50 tysięcy, przy założeniu, że rozmiar *batch'a* wynosi 4. Wynik ten został osiągnięty na podstawie analizy miar wydajności, takich jak średnia precyzja (*mAP*) oraz średni współczynnik wykrywalności (AR), które wskazują na poprawę wyników w miarę zwiększania liczby kroków treningowych. Przeprowadzenie wielu eksperymentów oraz analiza wykresów uczenia może być kluczowa dla dokładnego określenia optymalnej liczby kroków treningowych dla konkretnego zbioru danych i zadania.

Wybór odpowiedniej liczby kroków treningowych może mieć istotny wpływ na zdolność modelu do generalizacji na różnych danych testowych, w tym, tych pochodzących z różnych źródeł lub reprezentujących różne warunki. Zbyt krótki trening może prowadzić do niedouczenia, podczas gdy nadmierny trening może prowadzić do przeuczenia. Optymalna liczba kroków treningowych pozwala na znalezienie punktu równowagi pomiędzy tymi skrajnymi przypadkami, co przekłada się na lepszą zdolność uogólniania modelu. W tabeli 6.3 zaprezentowano wyniki ewaluacji sieci MobileNetV2 FPNLite.

Tabela 6.3: Wyniki ewaluacji sieci MobileNetV2 FPNLite 320x320

metryka/kroki	5	10	20	50	100	200
mAP	0,135	0,126	0,154	0,221	0,218	0,208
mAP (medium)	0,226	0,261	0,286	0,370	0,363	0,284
mAP (small)	0,130	0,116	0,142	0,215	0,210	0,209
mAP@.50IOU	0,331	0,354	0,385	0,503	0,499	0,461
mAP@.75IOU	0,076	0,061	0,091	0,126	0,142	0,160
AR@1	0,292	0,294	0,330	0,360	0,353	0,319
AR@10	0,390	0,404	0,442	0,464	0,451	0,430
AR@100	0,429	0,427	0,468	0,489	0,474	0,467
AR@100 (medium)	0,530	0,570	0,589	0,648	0,606	0,595
AR@100 (small)	0,419	0,412	0,456	0,473	0,461	0,455
Czas [min]	7	13	24	62	125	234

6.5. Wyniki treningu i porównanie sieci

Najlepsze wyniki precyzji i czułości ze wszystkich osiągnęła sieć YOLOV8s. Podobne, niewiele gorsze wyniki osiąga bliźniacza sieć YOLOV8n. Średnia precyzja i czułość jest najwyższa w zestawieniu. W tabeli nie ma informacji o reszcie metryk. Wynika to z faktu, że sieć była trenowana za pomocą biblioteki *Ultralytics* zamiast *Tensorflow*. Z tego powodu w tabeli zostały zamieszczone tylko takie metryki, które pokrywają się z tymi wykorzystywanymi w TensorFlow. Sieci, pomimo najwyższych wyników w zestawieniu trenują się bardzo krótko. Czasy treningu są najniższe spośród wszystkich sieci wykorzystanych w badaniu. Może to wynikać z faktu, że sieci potrzebują stosunkowo mało pamięci operacyjnej do treningu i można było wykonać trening wykorzystując duży rozmiar *batch'a*.

Uwzględniając sieci tylko trenowane we *framework'u Tensorflow*, równocześnie pomijając sieci z rodziny YOLO, najlepszą jest CenterNet MobileNetV2. Najlepszą średnią precyzję dla jednego obiektu otrzymały sieci wykorzystujące detektor CenterNet. Model wyróżnia się najwyższymi wartościami *mAP*, *mAP@50IOU* i *mAP@75IOU*. Oznacza to, że ma doskonałą zdolność do precyzyjnego wykrywania obiektów zarówno w ogólności, jak i w przypadku obiektów z dużym pokryciem. Pomimo czasu treningu zbliżonego do SSD MobileNetV2 FPNLite 640, sieć CenterNet MobileNetV2 FPN osiąga zdecydowanie lepsze wyniki średniej precyzji, jak i średniej swoistości. W innych metrykach również osiąga lepsze wyniki.

Sieć CenterNet HourGlass104 wymaga bardzo dużo pamięci operacyjnej do treningu i inferencji. Dla karty graficznej mającej 8GB pamięci udało się uruchomić trening dla dwóch *batch'y*. Trening sieci trwa stosunkowo długo i wyniki osiąga podobne do sieci SSD MobileNetV2 FPNLite 320. Jednocześnie architektura sieci jest większa i bardziej kosztowna obliczeniowo. W przypadku zastosowań mobilnych istnieją lepsze sieci w wykorzystane w badaniu. Wyniki testów sieci detekcji zostały przedstawione w tabelach 6.4 6.5. Niektóre z wartości w tabeli nie zostały uzupełnione. Wynika to z faktu, że dane podawane podczas treningu i testów różnią się w bibliotece *Tensorflow* i Ultralytics. W tabeli podano tylko te wyniki, które są ze sobą kompatybilne. Natomiast w tabeli 6.6 zaprezentowano dane dotyczące przeprowadzonego treningu, takie jak czas i liczba kroków.

EfficientDet D1 osiąga bardzo wysokie wyniki. Spośród sieci trenowanych w bibliotece *Tensorflow* znajduje się na drugim miejscu pod względem średniej precyzji jak i średniej swoistości. Największą wadą sieci jest bardzo długi czas treningu - trwał ponad 14 godzin. Jest on najdłuższy spośród wszystkich sieci znajdujących się w zestawieniu. Pomimo bardzo długiego czasu treningu sieć osiąga gorsze

Tabela 6.4: Wyniki testów sieci detekcji

metryka/sieć	mAP	mAP@ (50IOU)	mAP@ (75IOU)	mAP (small)	mAP (medium)	AR@1
CenterNet HourGlass104	0,234	0,493	0,187	0,225	0,334	0,398
CenterNet MobileNetV2 FPN	0,300	0,624	0,243	0,298	0,348	0,422
EfficentDet D0	0,251	0,594	0,149	0,234	0,442	0,376
EfficentDet D1	0,271	0,599	0,200	0,256	0,411	0,403
SSD MobileNetV2 FPNLite 320	0,218	0,499	0,142	0,210	0,363	0,353
SSD MobileNetV2 FPNLite 640	0,246	0,508	0,199	0,240	0,363	0,371
SSD MobileNetV2	0,234	0,564	0,151	0,224	0,406	0,370
SSD ResNet50	0,207	0,406	0,192	0,193	0,387	0,385
Faster RCNN ResNet50	0,116	0,315	0,053	0,115	0,204	0,297
YOLOV8n	0,415	0,726	N/D*	N/D*	N/D*	N/D*
YOLOV8s	0,423	0,717	N/D*	N/D*	N/D*	N/D*

*Brak danych - Biblioteka *Ultralytics* nie oblicza danych metryk podczas treningu

wyniki niż CenterNet MobileNetV2 FPN. Sieć EfficientDet D0 osiąga nieco gorsze wyniki niż większa sieć EfficientDet D1. Jednocześnie czas treningu sieci jest krótszy o około 40%. Wykazują konkurencyjne wyniki *mAP* i *mAP@50IOU*, ale niższe wartości *mAP@75IOU*, co może sugerować niższą precyzję dla obiektów z wyższym pokryciem.

Zauważono, że ekstraktor cech MobileNetV2 wykazuje znacznie lepsze wyniki w połączeniu z modelem CenterNet niż w przypadku łączenia z modelem SSD. MobileNetV2, pierwotnie zaprojektowany do wyznaczania punktów kluczowych, okazuje się niezwykle skutecznym narzędziem również w zadaniach detekcji obiektów. Sieć CenterNet HourGlass104, stanowiąca kombinację detektora i ekstraktora cech, została stworzona z założeniem wyznaczania punktów kluczowych. Model CenterNet HourGlass104, który pierwotnie miał być wykorzystywany do wykrywania kluczowych punktów na obrazach, okazał się nie tylko efektywny w tej dziedzinie, ale również wykazuje imponujące wyniki w zadaniu detekcji obiektów.

Spośród sieci SSD MobileNetV2 najlepsze wyniki osiąga ta, wykorzystująca FPNLite i detekcja jest wykonywana w rozdzielczości 640 pikseli. Gorsze wyniki osiąga sieć, która dokonuje detekcji w 320 pikselach. Trening tej sieci trwa znacznie krócej. Wnioskując z poniższych tabel, można zauważyć, że sieć SSD MobileNetV2 osiąga niższą precyzję i dokładność od bliźniaczej sieci (SSD MobileNetV2 FPNLite) wykorzystującej detektor FPNLite. Wykorzystanie dodatkowego detektora zmniejsza czas treningu oraz zwiększa precyzję oraz czułość. Modele SSD MobileNetV2 FPNLite i SSD MobileNetV2 osiągają przeciętne wyniki w porównaniu z innymi modelami. Mają niższą wartość *mAP*, *mAP@50IOU* i *mAP@75IOU* niż CenterNet MobileNetV2 FPN i EfficentDet D0/D1. Jednak nadal mogą być użyteczne w niektórych aplikacjach, szczególnie jeśli uwzględnione zostaną inne czynniki, takie jak szybkość działania.

Sieć FasterRCNN ResNet50 osiąga najniższe wyniki w tabeli. Może to być spowodowane tym, że sieć ma duże rozmiary i bardzo dobrze się dopasowuje do bardzo małego jak na taką sieć zbioru danych. Porównując detektory SSD i Faster RCNN możemy zauważyć znaczny spadek w jakości detekcji przy wykorzystaniu RCNN. Jakość detekcji jest niska, a zarazem wymagane jest dużo pamięci operacyjnej. Na urządzeniach docelowych - Raspberry Pi4 (4GB) i Nvidia Jetson (4GB) nie udało się uruchomić

Tabela 6.5: Wyniki testów sieci detekcji

metryka/sieć	AR@10	AR@100	AR@100 (small)	AR@100 (medium)	total loss
CenterNet HourGlass104	0,498	0,498	0,482	0,651	1,083
CenterNet MobileNetV2 FPN	0,517	0,517	0,512	0,573	1,416
EfficentDet D0	0,441	0,454	0,434	0,658	0,415
EfficentDet D1	0,493	0,506	0,661	0,491	0,375
SSD MobileNetV2 FPNLite 320	0,451	0,474	0,461	0,606	0,635
SSD MobileNetV2 FPNLite 640	0,489	0,502	0,489	0,632	0,982
SSD MobileNetV2	0,440	0,446	0,429	0,612	1,018
SSD ResNet50	0,494	0,522	0,507	0,681	3,998
Faster RCNN ResNet50	0,365	0,408	0,395	0,552	0,141
YOLOV8n	N/D*	N/D*	N/D*	N/D*	N/D*
YOLOV8s	N/D*	N/D*	N/D*	N/D*	N/D*

*Brak danych - Biblioteka *Ultralytics* nie oblicza danych metryk podczas treningu

sieci do testów. Średnia precyzja tej sieci jest prawie czterokrotnie mniejsza niż w sieci YOLOV8s. Warto zwrócić uwagę na to, że dobór odpowiedniego modelu może być zależny od konkretnego zastosowania, a model ten może mieć swoje unikalne zalety w innych dziedzinach.

W badaniu, w kontekście osiąganych wyników precyzji i czułości w zadaniu detekcji obiektów, model SSD ResNet50 znalazł się na przedostatnim miejscu w zestawieniu. Tylko sieć FasterRCNN ResNet50 uzyskała wyniki niższe od SSD ResNet50. Otrzymane rezultaty sugerują, że SSD ResNet50 nie osiąga najlepszych wyników w porównaniu z innymi analizowanymi modelami. Warto podkreślić, że trening SSD ResNet50 wymaga stosunkowo długiego czasu, co może stanowić istotny czynnik przy wyborze optymalnej architektury sieci. Skomplikowane modele, takie jak ResNet50, cechują się zazwyczaj znacznie większymi wymaganiami obliczeniowymi podczas treningu. Dłuższy czas trenowania może wynikać z głębokiej architektury ResNet50, która może być bardziej złożona w porównaniu do innych sieci analizowanych w badaniu. Jednakże, pomimo wyników niższych niż niektóre modele, SSD ResNet50 wciąż może być dobra w innych zastosowaniach. Każda architektura ma swoje unikalne zalety i wady, a wybór odpowiedniego modelu powinien być ściśle związany z konkretnymi wymaganiami i kontekstem aplikacji.

Tabela 6.6: Tabela zawierająca dane o przeprowadzonym treningu sieci neuronowych

metryka/sieć	Czas [min]	Batch	Rozdzielczość	Kroki [k]
CenterNet HourGlass104	416	2	512	100
CenterNet MobileNetV2 FPN	170	4	512	100
EfficentDet D0	511	8	512	50
EfficentDet D1	887	4	640	50
SSD MobileNetV2 FPNLite 320	125	4	320	100
SSD MobileNetV2 FPNLite 640	354	4	640	100
SSD MobileNetV2	144	8	320	100
SSD ResNet50	457	2	640	100
Faster RCNN ResNet50	169	1	150	150
YOLOV8n	63	16	640	22
YOLOV8s	91	16	640	11

W ramach badania porównawczego różnych sieci detekcji obiektów, które wykorzystują detektor Single Shot Multibox Detector (SSD) oraz ekstraktor cech, zauważono trend związany z rozdzielczością danych wejściowych. Wykazano, że większa rozdzielczość obrazów przyczynia się do poprawy wyników dokładności i swoistości w detekcji obiektów. Jednym z kluczowych aspektów, który został wzięty pod uwagę podczas analizy, było wykorzystanie Feature Pyramid Network (FPN) w architekturach sieci. FPN jest techniką, która ma na celu poprawienie wykrywania obiektów o różnych rozmiarach na obrazach poprzez hierarchiczne organizowanie cech w różnych skalach. Zauważono, że jedna z analizowanych sieci, która nie korzystała z FPN, uzyskała lepsze wyniki w miarach średniej precyzji *mAP* (ang. *mean Average Precision*) dla klas obiektów o małych i średnich rozmiarach (*mAP* small), *mAP* medium). Ta obserwacja może jest zaskakująca, ponieważ pierwotnym założeniem wykorzystania FPN było polepszenie detekcji obiektów o niewielkich rozmiarach, które zazwyczaj bywają trudniejsze do wykrycia. Istnieje kilka możliwych wyjaśnień dla tego zjawiska. Po pierwsze, sieć, która nie korzystała z FPN, mogła lepiej wykorzystać informacje z wysokiej jakości cech przestrzennych, dzięki czemu osiągnęła lepszą dokładność w wykrywaniu mniejszych obiektów. Może to sugerować, że w niektórych przypadkach FPN może wprowadzać pewne zakłócenia lub nieoptymalnie organizować cechy, co negatywnie wpływa na skuteczność detekcji. Ponadto, różnice w wydajności między sieciami mogą wynikać z różnych hiperparametrów.

6.6. Przykłady detekcji

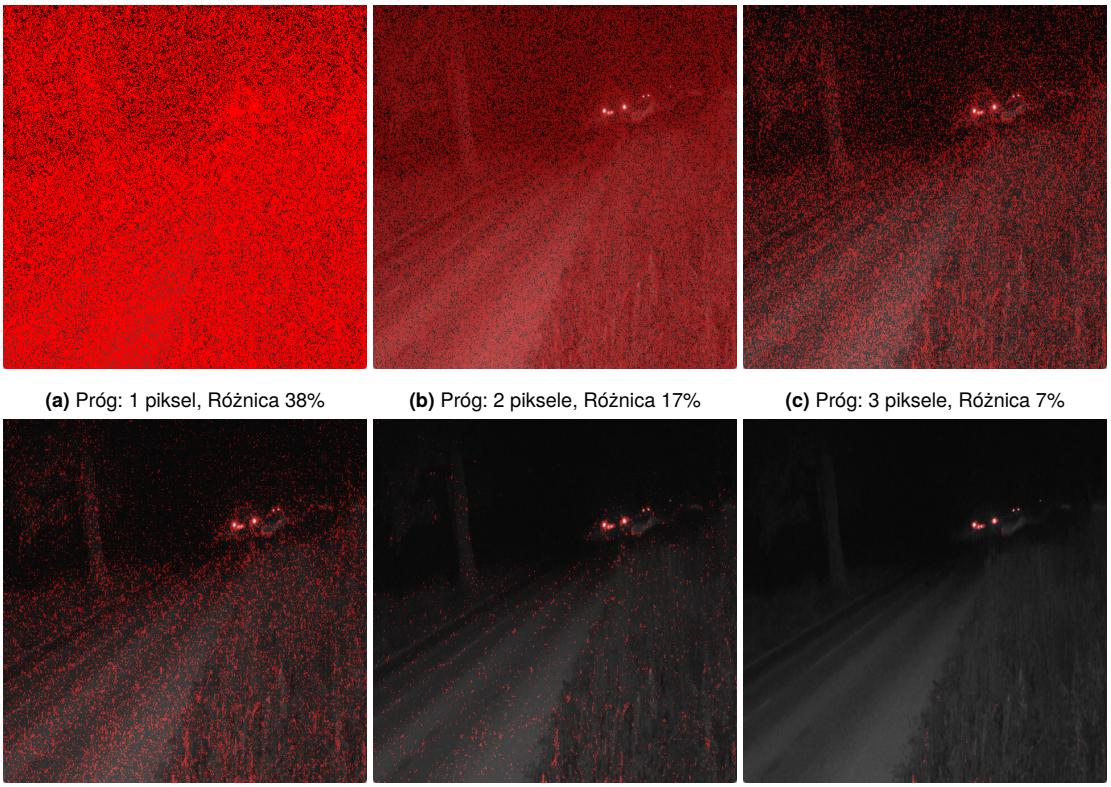
W niniejszej sekcji przedstawiono przykłady detekcji wykonanych przez badane sieci.

Jednym z problemów tych sieci jest różnica w wynikach detekcji na dwóch niemal identycznych obrazów. Z punktu widzenia człowieka oba obrazy są identyczne. Na rysunkach 6.5a i 6.5b przedstawiono wyniki detekcji na takich obrazach. Na rysunku 6.5a jeden sieć dokonała detekcji dwóch obiektów, natomiast na rysunku 6.5b został wykryty tylko jeden obiekt. Istnieje kilka powodów, dlaczego może występować takie zjawisko.

(a) Wynik detekcji na sieci YOLOV8s **(b)** Wynik detekcji na sieci YOLOV8s

Rysunek 6.5: Przykład detekcji wykonanej na sieci YOLOV8s. Zaprezentowane są dwa niemal identyczne obrazy, na których detekcja różni się.

Po pierwsze na obrazach mogą występować szumy, czyli nieznaczne różnice na obrazach. Na rysunku 6.6 ukazano różnice w obu obrazach poprzez nałożenie maski. Do jej nałożenia został wykorzystany próg. Oznacza to, o jaką wartość musi różnić się piksel, żeby został uznany za inny. Rysunek 6.6a przedstawia jakie różnice zachodzą na obrazach dla najmniejszego progu równego jeden. Można zauważyć, że maska pokrywa praktycznie cały obraz na czerwono, co oznacza, że obrazy z pewnością nie są takie same. W tym wypadku pikseli różniących pikseli jest 38% względem wszystkich pikseli. Dla progu równego dwa różnica w obrazach wynosi 17%. Wraz ze wzrostem progu ilość różnic się zmniejsza. Dopiero dla progu wynoszącego 10 pikseli, znalezione różnice odpowiadają tym, które widzi człowiek.



(a) Próg: 1 piksel, Różnica 38% **(b)** Próg: 2 piksele, Różnica 17% **(c)** Próg: 3 piksele, Różnica 7%

(d) Próg: 4 piksele, Różnica 3% **(e)** Próg: 5 pikseli, Różnica 1,3% **(f)** Próg: 10 pikseli, Różnica 0,11%

Rysunek 6.6: Różnice pomiędzy dwoma obrazami niemal nierozróżnialnymi dla człowieka. Na obraz została nałożona maska kolorem czerwonym, wskazującym na różnice na obrazach w zależności od progu.

Percepcja maszyny znacznie różni się od człowieka. Sieci detekcji są czułe i podatne na drobne zmiany w wartościach pikseli. Przede wszystkim, warto zauważyć, że obrazy rzeczywiste są podatne na różnego rodzaju zakłócenia i szumy. To może wynikać z warunków oświetleniowych, jakości sprzętu rejestrującego obrazy czy innych czynników zewnętrznych. W rezultacie dwie pozornie identyczne sceny mogą mieć pewne subtelne różnice pikselowe, które w obliczeniach maszynowych mogą prowadzić do różnic w wynikach detekcji. Dla niskich progów nawet niewielkie różnice pikseli są brane pod uwagę, co może prowadzić do wykrycia różnic tam, gdzie człowiek nie zauważyłby istotnych odmian. W miarę zwiększania progu tylko większe zmiany pikseli są uwzględniane, co bardziej odzwierciedla ludzką percepcję.

Istotnym aspektem jest również różnica w percepcji między ludźmi a maszynami. Ludzki system wzrokowy jest wysoce złożonym układem, który analizuje obrazy, uwzględniając kontekst, znane wzorce i doświadczenie. W przeciwieństwie do tego, sieci neuronowe, w tym modele YOLO, operują na matematycznych operacjach na pikselach i cechach, niekoniecznie biorąc pod uwagę pełen kontekst. Dlatego subtelne różnice, które nie są istotne dla ludzkiego oka, mogą mieć wpływ na działanie algorytmów detekcji.

Drugim powodem może być zbyt mały zbiór treningowy. Jeśli zbiór byłby bardziej liczny i wiele podobnych obrazów byłoby oznaczonych, możliwe, że sieć lepiej nauczyłaby się dokonywać detekcji pojazdów.

W celu zapobiegnięcia takiemu zjawisku można spróbować wykorzystać kwantyzację obrazu wejściowego w celu lepszego uogólniania i mniejszej podatności na drobne różnice w wartościach pikseli pomiędzy obrazami. Możliwe jest również wykorzystanie algorytmów śledzenia obiektów w celu zapobiegnięcia zjawisku "znikających"obiektów.

Na rysunku 6.7 ukazano jak sieć YOLOV8s sprawuje się podczas detekcji tyłu pojazdów w zależności od odległości. Według tego, jak był oznaczany zbiór danych pojazd na rysunku 6.7a nie powinien zostać wykryty i nie został. Natomiast pojazd na rysunku 6.7b powinien zostać wykryty. Algorytmowi sprawia problemy wykrywanie pojazdów z małej odległości. Przyczyną może być, że wskutek ruchu pojazd nie jest wyraźnie widoczny na zdjęciu. Algorytm dobrze sobie radzi z wykrywaniem pojazdu, jeśli jest daleko. W sekwencji trzech zdjęć pojazd został wykryty, dopiero gdy oddalił się wystarczająco od kamery.

(a) Tył pojazdu bardzo blisko **(b)** Tył pojazdu blisko **(c)** Tył pojazdu daleko

Rysunek 6.7: Wyniki detekcji tyłu pojazdu z zależności od odległości

Ponadto w algorytmach detekcji występuje problem kilkukrotnego wykrywania tego samego obiektu. Na rysunku 6.8 przedstawiono kilkukrotne wykrycia tego samego obiektu. Na pierwszym obrazie 6.8a jest przód pojazdu. Poprawnie zostały wykryte przednie światła mijania pojazdu, ale dodatkowo zostały wykryte również odbicia od jezdni tych świateł. W przypadku tego rodzaju błędu można by było zaimplementować algorytm, który by odrzucał te detekcje, gdzie już obok zostały wykryte światła pojazdu. Istnieje jednak zagrożenie, że jest to inny pojazd, który nie zostanie wykryty.

Na drugim obrazie 6.8b został ukazany tył ciężarówki. Samochody ciężarowe charakteryzują się dodatkowym oświetleniem, które z daleka wygląda jak oświetlenie tyłu pojazdu. Widoczne jest, że zostały prawidłowo wykryte światła na dole pojazdu i nieprawidłowo wykryte światła u góry pojazdu. W tym wypadku również dla człowieka sprawiłaby trudność ocena czy to jest oświetlenie ciężarówki, czy samochodu z naprzeciwka.

Na trzecim obrazie 6.8c znajduje się pojazd, który jest w małej odległości od kamery. Podczas detekcji wystąpił błąd i oprócz dołu świateł został oznaczony też mały fragment prawego tylnego światła. Światła z tyłu różnią się pomiędzy sobą. Autor wnioskuje, że algorytm podjął taką, a nie inną decyzję o oznaczeniu obiektu, dlatego, że wiele tylnych oświetleń pojazdów znajduje się nieco powyżej tablic rejestracyjnych. Ciężko natomiast wytłumaczyć, dlaczego algorytm oznaczył prawy górny fragment tylnego oświetlenia jako pojazd. Możliwe, że fragment obrazu jest podobny do pojazdu znajdującego się w dużej odległości. Istnieje duże prawdopodobieństwo, że tego typu sytuacji można by było zapobiec poprzez zastosowanie większego zbioru danych.

(a) Detekcja świateł pojazdu i odbić świateł na drodze

(b) Wykrycie świateł tylnych ciężarówki na dole i na górze

(c) Tył pojazdu znajdującego się blisko urządzenia rejestrującego

Rysunek 6.8: Wyniki detekcji obiektów. Na rysunku zostały przedstawione kilkukrotne wykrycia tego samego obiektu.

Na rysunku 6.9 widoczne są wyniki detekcji przodu pojazdów znajdujących się w dużej odległości od kamery. W większości przypadków obiekty znajdujące się daleko są poprawnie wykrywane. Na rysunku 6.9c widoczne jest światło o dużej intensywności. Pomimo takiego utrudnienia pojazd został poprawnie wykryty.

(a) Detekcja pojazdu z daleka

(b) Detekcja pojazdu z daleka

(c) Detekcja pojazdu z daleka

Rysunek 6.9: Wyniki detekcji przodu pojazdów znajdujących się w dużej odległości od kamery.

Na rysunku 6.10 ukazano wyniki detekcji nietypowych scenariuszy. Na pierwszym obrazie 6.10a znajduje się przyczepa, która nie znajdowała się w zbiorze treningowym. Model dzięki dobrym zdolnościom do generalizacji poprawnie wykrył tylne światła pojazdu. Obraz 6.10b przedstawia ciężarówkę jadącą po przeciwnej stronie. Rzadkim przypadkiem jest, aby światła pojazdu jadącego z przeciwnej strony do kamery dawała światło podobne do tego, jakie dają światła mijania samochodu jadącego wprost do urządzenia rejestrującego. Tutaj pojazd co prawda został wykryty, ale zła część pojazdu. Takim przypadkom można by było zapobiec, tworząc nową klasę świateł, które są odbiciami od jezdni. Na rysunku 6.10c po prawej stronie znajduje się pojazd. Od karoserii odbijają się światła, które są podobne do świateł mijania. W tym przypadku odbicia świetlne zostały wykryte jako pojazd. Jest to dodatkowe utrudnienie występujące w miastach, gdzie jest dodatkowe oświetlenie. Na ostatnim zdjęciu 6.10d znajduje się przyczepa ciężarówki. Tylne światło pojazdu emituje dwa punkty świetlne, które jest na tyle duże, że przypomina pojazd jadący z daleka. W przypadku człowieka nigdy nie wystąpiłaby pomyłka. Maszyna nie uwzględnia kontekstu otoczenia i wykryła nieprawidłowo obiekt.

(a) Poprawna detekcja tyłu przyczepy.

(b) Detekcja świateł mijania pojazdu jadącego z przeciwnej strony.

(c) Po prawej stronie widoczna jest detekcja świateł na karoserii pojazdu.

(d) Detekcja jednego światła jako pojazdu.

Rysunek 6.10: Wyniki detekcji nietypowych przypadków.

W mieście ze względu na więcej źródeł oświetlenia detekcja jest trudniejsza. Na rysunkach 6.11 przedstawiono nieudane detekcje w mieście. Na obrazie 6.11a lampa została wykryta jako światła mijania pojazdu. Składają się z dwóch poziomych strumieni świetlnych i są podobne do jadącego pojazdu. W celu zapobiegnięcia takim sytuacjom powinna zostać dodana dodatkowa klasa latarni w celu poprawy wyników detekcji. Na rysunku 6.11b przedstawione jest oświetlone logo firmy znajdujące się na budynku. W tym przypadku obiekt zdecydowanie mniej przypomina jadący pojazd, jednak przez algorytm detekcji został wykryty jako światła tylne pojazdu. Swoim wyglądem przypominają światło ciężarówki na 6.10d. Przez przypadek kamera została przez chwilę zakryta. Przez krótki czas kamera utraciła parametry dobranej ekspozycji i musiała na nowo je ustawić. W tym czasie obraz był ciemny i zmieniły się wyniki detekcji. Zostały wykryte tylko obiekty, które nie powinny zostać wykryte - latarnie miejskie.

Dodatkowe oświetlenie miejskie sprawia algorytmom detekcji. Z drugiej strony niektóre obiekty są bardzo podobne do nadjeżdżających pojazdów i bez kontekstu człowiek również mógłby się pomylić.

(a) Detekcja lampy jako światła jako pojazd

(b) Detekcja logo na budynku jako pojazd

(c) Przykładowa detekcja w mieście

Rysunek 6.11: Wyniki detekcji w mieście.

6.7. Optymalizacja liczby kolorów

Podjęto próby optymalizacji sieci neuronowych. Po pierwsze zastosowano redukcję liczby kolorów znajdujących się w obrazie wejściowym. Do testów została wybrana sieć YOLOV8n ze względu na osiągnięte wysokie wyniki w testach jak i krótki czas treningu. Za pomocą biblioteki OpenCV [78] wykonano redukcję kolorów. Wykorzystano do tego algorytm k-średnich (ang. *k-means*). Przy użyciu tej metodologii, piksele obrazu były przyporządkowywane do konkretnych klastrów reprezentujących określone barwy. Rezultatem tej operacji była znacznie uproszczona paleta kolorów, przyczyniająca się nie tylko do zmniejszenia złożoności danych, ale także do potencjalnego wyeliminowania nieistotnych detali, skupiając uwagę na elementach kluczowych.

Celem badania było sprawdzenie, czy poprawią się wyniki detekcji i rozwiążą problem znikających obiektów w sekwencji klatka po klatce. Problem ten został zilustrowany na rysunku 6.5. Mniejsza liczba kolorów powinna wpłynąć na mniejsze zróżnicowanie kolejnych klatek między sobą. Metryką sprawdzającą wydajność sieci jest *mAP* i *mAP@50IOU*. Ten zabieg może pogorszyć sieć w innych aspektach, dlatego zdecydowano się na uniwersalną metrykę.

W tabeli 6.7 zamieszczono szczegółowe rezultaty przeprowadzonego badania. Tabela prezentuje wyniki dla różnych liczb kolorów wchodzących w skład obrazów wejściowych.

Niewielka redukcja liczby kolorów (64 i 16) w zbiorze pozytywnie wpłynęła na wyniki sieci. Większa redukcja kolorów poskutkowała gorszymi wynikami w metrykach *mAP* i *mAP@50IOU*. Przy liczbie kolorów równej osiem lub cztery wyniki sieci są akceptowalne. Liczba kolorów równa dwa znacznie pogarsza wyniki sieci.

Jak wynika z analizy, niewielka redukcja liczby kolorów (do 64 lub 16) przyczyniła się do poprawy wyników sieci. Z kolei głębsza redukcja kolorów prowadziła do pogorszenia wyników w metrykach *mAP* oraz *mAP@50IOU*. Przy liczbie kolorów wynoszącej osiem lub cztery, wyniki sieci były akceptowalne. Natomiast przy dwóch kolorach, wyniki sieci istotnie się pogorszyły.

Tabela 6.7: Wyniki testów sieci detekcji dla różnej liczby kolorów wchodzących w skład obrazów wejściowych

sieć	liczba kolorów	mAP	mAP@50IOU
YOLOV8n	256	0,415	0,726
YOLOV8n	64	0,428	0,747
YOLOV8n	16	0,420	0,729
YOLOV8n	8	0,386	0,680
YOLOV8n	4	0,371	0,660
YOLOV8n	2	0,223	0,535

Warto również zaznaczyć, że obok korzyści związanych z poprawą wydajności, redukcja liczbie kolorów może mieć korzystny wpływ na szybkość przetwarzania sieci. Obniżenie rozmiarów obrazu wejściowego umożliwia potencjalnie możliwość wykorzystania mniejszych zasobów pamięciowych oraz obliczeniowych. Taki kierunek wydajnościowy jednakże wymagałby opracowania dedykowanej architektury sieci oraz dostosowania sprzętu do pracy z mniejszymi rozmiarami zmiennych.

7. TESTY WYDAJNOŚCI

Sieci zostały sprawdzone na platformie *Nvidia Jetson Nano* P3450, *Raspberry Pi 4* (4GB). Na obu platformach wykonano też testy z akceleratorem *TPU Google Coral USB Accelerator WA1* [52]. Na żadnej z platform nie były dokonywane modyfikacje. Na *Raspberry Pi 4* nie był wykorzystywany radiator ani wentylator. *Google Coral TPU accelerator* był wykorzystany w wersji bez zwiększania taktowania procesora [79]. Na urządzeniach został wykorzystany *Tensorflow* w wersji 2.12 i *PyTorch* w wersji 1.13.1.

Testy na platformie *Nvidia Jetson Nano* zostały przeprowadzone w dwóch trybach zasilania. Pierwszy tryb wykorzystuje maksymalną moc urządzenia, a drugi tryb ogranicza pobór mocy do 5 watów.

W testach wydajnościowych zostały sprawdzone sieci z *framework'u Tensorflow*. Sieci YOLOV8 zostały wytrenowane i przetestowane we *framework'u Ultralytics*. W celu możliwości odniesienia się do starszych algorytmów niewykorzystujących głębokich sieci neuronowych przebadane zostały również kaskady *Haar'a*, które zostały zaimplementowane w bibliotece OpenCV.

7.1. Formaty eksportowanych sieci

W badaniach nad wydajnością sieci neuronowych istotną częścią procesu jest przetestowanie sieci w różnych formatach. W tym celu sieci zostały były wyeksportowane do różnych formatów. W celu przeprowadzenia testów sieci zostały wyeksportowane. Sieci z *framework'a Tensorflow* zostały wyeksportowane do formatu *tflite* i *pt*. Natomiast sieci YOLO z *framework'a Ultralytics*, który jest oparty na *framework'u PyTorch* zostały wyeksportowane do formatu *onnx* [80] i *pt*. W tabeli 7.1 zostały przedstawione wykorzystane w badaniu formaty zapisu sieci neuronowych.

Tabela 7.1: Zestawione formaty zapisu sieci neuronowych wraz z nazwą i wspierającą natywnie biblioteką

Nazwa formatu	Zastosowanie	Biblioteka
tflite	Mobilne	TensorFlow
tflite (wersja dla TPU)	TPU	TensorFlow
onnx	Mobilne	PyTorch
pb	Ogólnego przeznaczenia	TensorFlow
pt	Ogólnego przeznaczenia	PyTorch

Formaty *pt* (protobuf) i *pt* to formaty danych ogólnego przeznaczenia, które zawierają pełną wersję sieci wraz z wszystkimi parametrami. Protobuf jest językiem opisu danych opracowanym, który umożliwia kompaktowe i wydajne przechowywanie danych strukturalnych, takich jak modele sieci neuronowych. W formacie *pt* sieć jest przedstawiana w postaci binarnej, co pozwala na szybkie i efektywne wczytywanie modelu na różnych platformach sprzętowych. Kolejnym rozważanym formatem jest format *PyTorch* (*pt*), który jest charakterystyczny dla biblioteki *PyTorch*, będącej jednym z popularnych bibliotek do uczenia maszynowego i głębokiego uczenia. Format *pt* również zawiera pełną definicję modelu wraz z wagami i innymi parametrami modelu. Ten format jest użyteczny w przypadku zastosowań, gdzie model jest trenowany i testowany w środowisku *PyTorch*. Warto zaznaczyć, że formaty *pt* i *pt* są przystosowane do platform sprzętowych z większymi zasobami obliczeniowymi, takimi jak komputery osobiste i serwery. Urządzenia z ograniczonymi zasobami, takie jak urządzenia wbudowane, smartfony czy mikrokomputery, mogą wymagać innych formatów danych, które są bardziej zoptymalizowane pod kątem wydajności na takich urządzeniach.

Kolejny rodzaj formatu sieci służy do zastosowań mobilnych. W badaniu przetestowane zostały dwa tego typu formaty - *tflite* i *onnx*. Wykorzystywane są przeważnie w urządzeniach o mniejszych zasobach sprzętowych takich jak urządzenia mobilne. Charakteryzują się lepszą optymalizacją sieci pod kątem wykorzystanych zasobów, kwantyzacją oraz ograniczonym zestawem operacji. Format *onnx* najczęściej jest wykorzystywany we *framework'u PyTorch* a *tflite* we *framework'u Tensorflow*. W związku, że ten format jest dedykowany dla urządzeń mobilnych, to nie są wspierane operacje na karcie graficznej. W celu przyspieszenia inferencji wykorzystywane są akceleratory sprzętowe TPU takie jak *Google Coral*. Sieć powinna zostać specjalnie wyeksportowana przez dedykowany przez producenta kompilator [81].

7.2. Napotkane problemy

Nie udało się wyeksportować sieci Faster-RCNN ResNet 50. Wystąpiły problemy podczas eksportu sieci do formatu *pt*. Niektóre z instrukcji były niekompatybilne.

Drugim problemem było wykorzystanie akceleratorów graficznych na urządzeniach. W przypadku *Raspberry Pi 4*, który posiada zintegrowany układ graficzny Broadcom VideoCore, wspierany jest tylko *OpenCL*. Wykorzystane biblioteki *Tensorflow* i *PyTorch* wykorzystują do przyspieszenia bibliotekę *CUDA*. Znalezione alternatywne rozwiązania były nie były wystarczająco wspierane lub przestarzałe. *OpenCL* nie jest tak popularne, jak *CUDA* w środowisku uczenia maszynowego i głębokiego uczenia. Z tego powodu, wsparcie i rozwijanie bibliotek korzystających z *OpenCL* jest mniejsze, co wpływa na dostępność dokumentacji. W przypadku platformy sprzętowej *Nvidia Jetson Nano* występowały problemy z wersjami biblioteki *CUDA*. Instrukcje, które wykorzystywały wytrenowane algorytmy, były niekompatybilne. Z tego powodu zdecydowano się zrezygnować z testów na akceleratorach sprzętowych. Pomimo ograniczenia w wydajności obliczeniowej, testy na procesorze pozwalają na ocenę podstawowej skuteczności sieci na tych urządzeniach.

Przy próbie migracji modeli z *Tensorflow* 2 Detection Model Zoo do formatu *tflite* zgodnego z *Google Coral TPU* napotkano na pewne trudności techniczne. Sieci z tej architektury składają się z instrukcji, których nie obsługuje *Google Coral TPU*. Ograniczenia te obejmują różnice w obsługiwanych operacjach i warstwach sieciowych, co prowadzi do niekompatybilności między tymi dwoma środowiskami. W konsekwencji proces wyeksportowania sieci neuronowych z *Tensorflow* 2 Detection Model Zoo do formatu zgodnego z *Google Coral TPU* nie pomógł w przyspieszeniu obliczeń. Wręcz przeciwnie doprowadził do wolniejszego działania modeli na dedykowanej jednostce sprzętowej. W przypadku wystąpienia instrukcji, które nie są wspierane przez *Google Coral TPU*, system automatycznie kieruje te operacje do standardowego CPU urządzenia. Wykonywanie tych nieobsługiwanych instrukcji na CPU wprowadza dodatkowy narzut obliczeniowy i ogranicza ograniczyć potencjał przyspieszenia, który można uzyskać dzięki wykorzystaniu akceleratora sprzętowego TPU.

CenterNet MobileNetV2 FPN to model sieci neuronowej o dużej Złożoności obliczeniowej i liczbie parametrów wymagają znacznej ilości pamięci operacyjnej. Podczas prób uruchomienia modelu w formacie *pt* na obu mikrokomputerach napotykany został problem braku wystarczającej ilości dostępnej pamięci operacyjnej. Podczas inicjalizacji modelu czasami występowały błędy. Gdy, udało się uruchomić algorytm, to niezbędne operacje w trakcie działania modelu przekraczały dostępne zasoby pamięciowe, co prowadzało do błędów i niestabilności algorytmu na platformie docelowej.

Problem niewystarczającej pamięci operacyjnej został rozwiązany poprzez konwersję modelu CenterNet MobileNet V2 do formatu *tflite*. W trakcie konwersji model jest zoptymalizowany i kwantyzowany, co doprowadziło do znacznego zmniejszenia jego rozmiaru oraz zapotrzebowania na pamięć operacyjną. Dzięki temu przekształcony model w formacie *tflite* udało się uruchomić na mikrokompute-

rach.

YOLOV8s w formacie *pt* nie została sprawdzona na *Raspberry Pi 4* ze względu na niewystarczającą ilość pamięci RAM w urządzeniu.

7.3. Wyniki

Wśród formatów ogólnego przeznaczenia, najlepsze wyniki osiągnięto za pomocą urządzenia *Nvidia Jetson Nano*. Maksymalna moc wykorzystywana przez to urządzenie w momencie osiągania szczytowej wydajności wyniosła 7,7W, natomiast w trybie ograniczonej wydajności, mikrokomputer wykorzystywał maksymalnie 4,8W. W przypadku aktywacji trybu niskiego zużycia energii urządzenie uzyskiwało nieco niższe wyniki wydajnościowe w zamian za znaczne ograniczenie zużywanej energii. Warto zauważyć, że w kontrastujący sposób prezentuje się sytuacja urządzenia *Raspberry Pi 4*, które osiąga najgorsze wyniki wydajnościowe w tej grupie. Przyczyną takiego stanu rzeczy może być zastosowana architektura procesora, która wpływa negatywnie na efektywność działania tego urządzenia. Jego zużycie energii podczas użytkowania wyniosło 6,4W, co stanowi wyraźne odchylenie od wyników uzyskanych przez konkurencyjne rozwiązanie - *Nvidia Jetson Nano*. W rezultacie, jeśli priorytetem jest uzyskanie najwyższej wydajności przy relatywnie niewielkim zużyciu energii, to *Nvidia Jetson Nano* jawi się jako preferowane rozwiązanie w zestawieniu z *Raspberry Pi 4*, które wykazuje niższą wydajność i nieco wyższe zużycie energii. W tabeli 7.2 ukazano wyniki sieci w formatach *pt* i *pt*.

Tabela 7.2: Wyniki testu wydajności (klatki na sekundę) na sieciach wyeksportowanych do formatów ogólnego przeznaczenia

Sieć/Urządzenie	Jetson 5W	Jetson	Raspberry
SSD MobileNetV2 (pb)	11,30	13,89	8,87
SSD MobileNetV2 FPNLite 320 (pb)	13,38	15,34	7,63
SSD MobileNetV2 FPNLite 640 (pb)	5,25	6,18	2,11
EfficentDet D0 (pb)	4,74	5,81	1,88
EfficentDet D1 (pb)	2,25	2,87	0,86
SSD ResNet50 (pb)	1,39	1,84	0,26
YOLOV8n (pt)	6,04	8,43	1,22
YOLOV8s (pt)	2,72	3,82	N/D*

*Nie udało się uruchomić sieci

W nawiasie został podany format, do którego został wyeksportowany model

Analiza wariantów formatów mobilnych, takich jak *tflite* i *onnx*, ukazuje różnice w wydajności pomiędzy platformami. W tej kontekście, urządzenie *Raspberry Pi 4* wydaje się być najlepszym wyborem. W przypadku sieci z rodziny YOLO, różnice w wydajności są niewielkie. Jednakże, przy użyciu sieci YOLOV8s, platforma Jetson prezentuje się nieco lepiej. Warto zwrócić uwagę na zużycie energii, gdzie *Raspberry Pi* również wypada korzystnie. Natomiast, w przypadku mikrokomputera Jetson, wyniki związane z ograniczonym zużyciem energii nie są zadowalające. Można zauważyć, że wydajność sieci YOLO spada znacznie, nawet trzykrotnie, w porównaniu do trybu maksymalnego zużycia energii. W tabeli 7.3 zaprezentowano wyniki testu wydajności sieci wyeksportowanych do formatów mobilnych.

Tabela 7.3: Test wydajności sieci (klatki na sekundę) wyeksportowanych do formatu *tflite* lub *onnx*

Sieć/Urządzenie	Jetson 5W	Jetson	Raspberry
SSD MobileNetV2	3,65	5,41	6,14
SSD MobileNetV2 FPNLite 320	2,81	4,14	5,23
CenterNet MobileNetV2 FPN	1,01	1,46	1,84
SSD MobileNetV2 FPNLite 640	0,71	1,04	1,20
EfficentDet D0	0,58	0,85	0,98
EfficentDet D1	0,27	0,40	0,45
SSD ResNet50	0,04	0,05	0,05
CenterNet HourGlass104	0,01	0,01	0,01
YOLOV8n	0,44	1,52	1,57
YOLOV8s	0,19	0,69	0,64

W tabeli 7.4 zaprezentowano wyniki sieci wyeksportowanych do formatu zgodnego z *Google Coral TPU*. Niestety eksport do tego formatu się nie powiódł. Sieci wytrenowane we *framework'u Tensorflow* Models ZOO miały niekompatybilne instrukcje. Wskutek tego wiele operacji było wykonywanych na procesorze komputera, do którego był podłączony akcelerator. Komunikacja pomiędzy urządzeniami spowodowała niewielkie polepszenie wyników a czasami nawet pogorszenie mimo potencjalnie większej mocy obliczeniowej.

Tabela 7.4: Test wydajności sieci (klatki na sekundę) wyeksportowanych do formatu *tflite* z akceleracją sprzętową *Google Coral*

Sieć/Urządzenie	Raspberry + Google Coral	Jetson + Google Coral
SSD MobileNetV2	4,13	3,11
SSD MobileNetV2 FPNLite 640	3,27	2,44
CenterNet MobileNetV2 FPN	1,18	0,88
SSD MobileNetV2 FPNLite 320	0,85	N/D*
EfficentDet D0	0,70	0,49
EfficentDet D1	0,31	0,22
SSD ResNet50	0,05	0,03

*Nie udało się uruchomić sieci z powodu zbyt małej ilości pamięci operacyjnej

Z racji, że nie udało się wyeksportować sieci z *Tensorflow* 2 Detection Model Zoo zostały sprawdzone sieci specjalnie wyeksportowane przez producenta. W tabeli 7.5 są wyniki testów wydajności. Każda z sieci została sprawdzona w dwóch wersjach. Jedna z wersji jest domyślnie wyeksportowana do formatu *tflite* i nie są wykonywane instrukcje na TPU. Druga z sieci jest wyeksportowana do formatu, który jest zgodny z *Google Coral* i instrukcje wykonywane są na tym urządzeniu. Analiza wyników zawartych w Tabeli ujawnia znaczące różnice w przyspieszeniu dla poszczególnych sieci. W przypadku sieci EfficientDet Lite0 320 zastosowanie dedykowanego układu TPU przyniosło niemal trzykrotne przyspieszenie. Natomiast dla sieci SSD MobileNetV1 wynikające z optymalizacji przyspieszenie było nawet 18-krotne większe w porównaniu do wersji niewykorzystującej TPU. Ponadto można zauważyć, że w przypadku sieci *tflite* wykonywanych na CPU, komputer działa znacznie wolniej od mikrokomputerów. Może to wynikać z architektury procesora. Oba komputery mobilne wykorzystują architekturę x64 na-

tomiast procesor PC wykorzystuje architekturę procesora x86. Drugim powodem, dlaczego występują takie różnice, może być inny system operacyjny, na którym były wykonywane testy. W przypadku mikrokomputerów Nvidia Jetson działał na specjalnie przystosowanym przez producenta systemie Ubuntu. *Raspberry Pi 4* wykorzystuje system operacyjny *Raspberry Pi* OS oparty na dystrybucji Linux'a Debian. W przypadku komputera osobistego został wykorzystany system operacyjny Windows 11 z poprawkami 22H2. W badaniu [82] dokonano testów Pythona na Linuksie i Windowsie. W tym przypadku system operacyjny Windows okazał się szybszy. Badanie jednak może wskazywać na różnice w działaniu tych samych programów.

Tabela 7.5: Test wydajności sieci (klatki na sekundę) wyeksportowanych do formatu *tflite* lub *onnx* z akceleracją sprzętową *Google Coral* (sieci pobrane z googlecoral)

Sieć/Urządzenie	PC	Raspberry	Jetson
MobileNet_v2_1,0_224_inat_bird_quant.tflite	0,76	12,11	7,76
MobileNet_v2_1,0_224_inat_bird_quant_edgetpu.tflite	249,23	45,58	93,85
efficientdet_lite0_320_ptq.tflite	0,23	3,45	2,10
efficientdet_lite0_320_ptq_edgetpu.tflite	25,19	9,47	5,67
efficientdet_lite1_384_ptq.tflite	0,11	1,82	1,11
efficientdet_lite1_384_ptq_edgetpu.tflite	16,48	6,67	3,90
tf2_ssd_MobileNet_v1_fpn_640x640_coco17_ptq.tflite	0,003	0,12	0,08
tf2_ssd_MobileNet_v1_fpn_640x640_coco17_ptq_edgetpu.tflite	4,15	2,21	1,41

Przeprowadzono również dodatkowy test sprawdzający test wydajności algorytmów, które nie są sieciami neuronowymi. W tabeli 7.6 ukazano wyniki algorytmów. Metoda polegająca na zliczaniu liczby jasnych pikseli osiągnęła najwyższe wyniki. Największa wydajność została osiągnięta na platformie Jetson bez limitu zużycia energii. Dziesięć filtrów *Haar'a* osiąga podobną liczbę klatek na sekundę jak sieci SSD MobileNetV2. Z racji, że istnieją lepsze rozwiązania detekcji obiektów (na TPU sieć MobileNetV2 osiąga wynik w szybkości inferencji zbliżone do zliczania jasnych pikseli) to nie były mierzone metryki dokładności na tych algorytmach.

Tabela 7.6: Test wydajności (klatki na sekundę) algorytmów bez sieci neuronowych

Algorytm/Urządzenie	Jetson 5W	Jetson	Raspberry
Haar (1 filtr)	8,45	12,82	11,45
Haar (10 filtrów)	1,22	1,82	1,51
Haar (50 filtrów)	0,83	1,26	1,23
liczba jasnych pikseli	317,58	486,64	319,18

8. CZUJNIK INTENSYWNOŚCI ŚWIATŁA

Metody detekcji obiektów i zjawisk w różnych dziedzinach, takich jak przetwarzanie obrazu stanowią istotny element badań naukowych i aplikacji praktycznych. Wykorzystując zaawansowane algorytmy, te metody mogą okazać się bardzo skomplikowane obliczeniowo, co wymaga wydajnego sprzętu, zdolnego do przetwarzania znacznych ilości danych w krótkim czasie. Skomplikowane metody detekcji często niosą za sobą duże wymagania związane zarówno z wydajnością sprzętu, jak i z zapotrzebowaniem na energię elektryczną. Istnieją również mniej złożone i metody wykrywania pojazdów, które pozwalają na osiągnięcie satysfakcjonujących wyników. Jednym z takich podejść jest klasyfikacja na podstawie danych z czujnika intensywności światła. Dane o intensywności światła były zbierane równolegle z kamerą.

Aby zapewnić poprawność wyników i skuteczność algorytmu, niezbędne jest odpowiednie połączenie danych z kamery i czujnika. W tym celu dokonano synchronizacji klatek z kamery z danymi z czujnika intensywności światła. Istotnym wyzwaniem było dopasowanie różnych częstotliwości próbkowania - dane z kamery zbierane były z częstotliwością 25 klatek na sekundę, podczas gdy dane z czujnika z częstotliwością około 50 próbek na sekundę. Aby osiągnąć odpowiednią synchronizację, zdecydowano się wykorzystać co drugą próbkę z czujnika, co umożliwiło skuteczną korelację z danymi z kamery. Warto zaznaczyć, że taki proces synchronizacji może być wymagający pod względem czasowym i obliczeniowym, ale jest kluczowy dla osiągnięcia precyzyjnych rezultatów detekcji.

Do badania zebrano ogółem nieco ponad 14000 próbek intensywności światła, które zostały podzielone na część treningową i testową, stosując popularną proporcję 80/20. Taka praktyka pozwala na efektywną naukę klasyfikatora oraz jego dokładne testowanie na nieznanych wcześniej danych.

Przed przystąpieniem do samego procesu adnotacji. dane zostały poddane normalizacji przy użyciu standaryzacji Z. Jest to jedna z technik używanych w statystyce i analizie danych do przekształcenia wartości zmiennych w taki sposób, aby miały średnią równą zero i odchylenie standardowe równa jeden. Wykorzystywana jest w normalizacji sygnałów EEG [83], predykcji w finansach [84] lub w księgowości do wykrywania przestępstw [85]. Jest to przydatna technika w przypadku, gdy zmienne mają różne zakresy i jednostki, a chcemy przekształcić je tak, aby można je porównywać na wspólnej skali. W tym celu trzeba obliczyć średnią (μ) i odchylenia standardowego (σ) dla tej zmiennej w całym zbiorze danych. Sumę arytmetyczną μ obliczono przy wykorzystaniu wzoru 8.1.

$$\mu = \frac{\sum_{i=0}^{N} x_i}{N} \tag{8.1}$$

Gdzie N to liczba próbek.

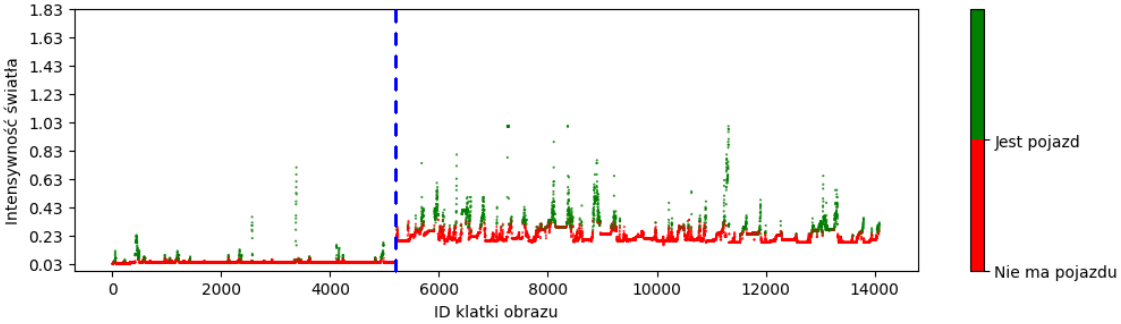
Odchylenie standardowe obliczono przy wykorzystaniu wzoru 8.2.

$$\sigma = \sum_{i=0}^{N} (x_i - \mu)^2 \tag{8.2}$$

Przekształcenie każdej wartości zmiennej x na wartość standaryzacji Z (z) wykonano przy użyciu wzoru 8.3.

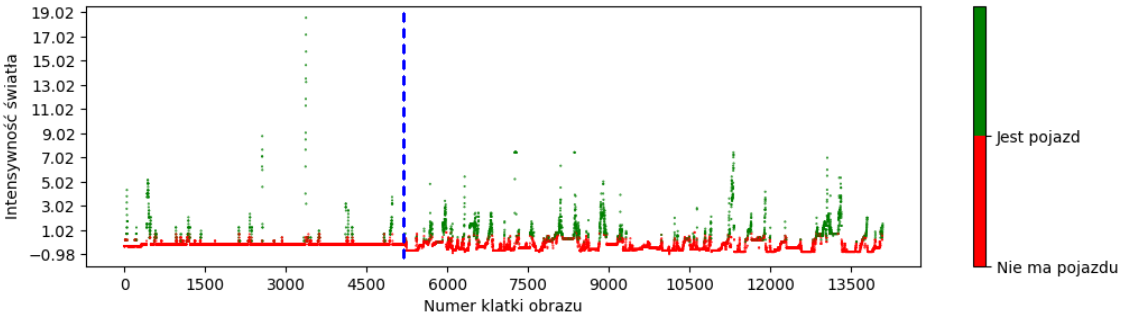
$$z_i = (x_i - \mu)/\sigma \tag{8.3}$$

Na rysunku 8.1 przedstawiono wykres wartości po standaryzacji Z po całym zbiorze. Trudno jest utworzyć linię obdzielającą klasy "jest pojazd"i "nie ma pojazdu".



Rysunek 8.1: Wykres danych z czujnika po normalizacji po całym zbiorze z uwzględnieniem klas, kolorem niebieskim rozdzielono nagrania (z prawej strony wieś, z prawej strony miasto)

W przypadku tego badania średnia i odchylenie standardowe było liczone osobno dla każdego uruchomienia urządzenia (4 razy). Każde miejsce różniło się pod względem oświetlenia jak lub umiejscowienia urządzenia rejestrującego. Dzięki takiemu sposobowi normalizacji na poniższym wykresie 8.2 wyraźnie rozróżnialne są światła przejeżdżającego pojazdu.



Rysunek 8.2: Wykres danych z czujnika po normalizacji po każdym nagraniu z uwzględnieniem klas, kolorem niebieskim rozdzielono nagrania (z prawej strony wieś, z prawej strony miasto)

W kolejnym etapie pozyskane dane zostały poddane procesowi adnotacji w celu przydzielenia odpowiednich etykiet, rozróżniających dwa przypadki: "jest pojazd"oraz "nie ma pojazdu". Warto zaznaczyć, że adnotacja w przypadku czujnika intensywności światła różni się od standardowych technik stosowanych w przypadku analizy obrazów. Model czujnika, który był wykorzystywany w badaniu, wykazywał pewne trudności w wykrywaniu pojazdów znajdujących się w większej odległości od urządzenia rejestrującego. Fakt ten wynikał z ograniczeń samej technologii czujnika, który nie był w stanie wykryć światła emitowanego przez pojazdy znajdujące się w większej odległości. Podczas procesu adnotacji, klasa "jest pojazd"została przypisana tylko w przypadku, gdy pojazd znajdował się w stosunkowo bliskiej odległości od urządzenia rejestrującego, a światło emitowane przez ten pojazd padało na czujnik z dużą intensywnością. Dzięki temu podejściu uniknięto błędnej klasyfikacji, które mogłaby wyniknąć z próby identyfikacji pojazdu na podstawie zbyt słabych lub niezauważalnych sygnałów świetlnych dla czujnika. Wybór takiego kryterium adnotacji stanowi kluczowy aspekt badania, mający istotny wpływ na jakość klasyfikatora. Warto zwrócić uwagę, że podejście to, choć mogło pomóc w poprawnym wykrywaniu pojazdów znajdujących się w bezpośrednim sąsiedztwie czujnika, może być bardziej ograniczone, jeśli celem analizy są pojazdy oddalone od urządzenia rejestrującego. W takich przypadkach, zastosowanie innych technik detekcji może okazać się bardziej odpowiednie.

Do badania wykorzystano dwie sieci neuronowe. Pierwsza składa się z bloku LSTM [86] i jednej

warstwy płaskiej o rozmiarze 128. Funkcję straty, jaką wykorzystano to entropia krzyżowa. Do optymalizacji został wykorzystany algorytm Adam.

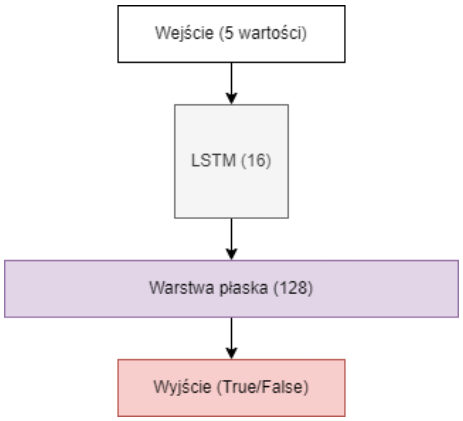
W badaniu zastosowano dwie sieci neuronowe w celu rozwiązania konkretnej problematyki. Pierwsza z tych sieci została zbudowana z wykorzystaniem bloku LSTM (ang. *Long Short-Term Memory*) [86], będącego specjalnym rodzajem rekurencyjnej jednostki sieci neuronowej, zdolnym do skutecznego modelowania zależności czasowych w danych sekwencyjnych. W skład tej pierwszej sieci wchodzi także jedna warstwa płaska o rozmiarze 128, która ma za zadanie dalszą przetwarzanie informacji pochodzących z bloku LSTM.

Ważnym elementem uczenia maszynowego jest dobór odpowiedniej funkcji straty (ang. *loss function*), która służy jako miara różnicy między przewidywanymi wartościami a rzeczywistymi danymi. Zdecydowano się na wykorzystanie entropii krzyżowej (ang. *cross-entropy*) jako funkcji straty. Funkcja ta jest szczególnie popularna w zadaniach klasyfikacji i przewidywania, gdyż skutecznie odzwierciedla stopień odległości między przewidywaniami modelu a rzeczywistymi etykietami.

Proces optymalizacji sieci neuronowej jest niezwykle istotnym etapem, który ma na celu dostosowanie wag i parametrów modelu w taki sposób, aby zmniejszyć wartość funkcji straty i poprawić jakość predykcji. W niniejszym badaniu wykorzystano popularny i skuteczny algorytm optymalizacji znany jako Adam [87]. Algorytm Adam łączy w sobie zalety algorytmów momentu (ang. *momentum*) oraz metody gradientu stochastycznego (ang. *SGD - stochastic gradient descent*), co pozwala na efektywniejszą aktualizację parametrów sieci neuronowej w trakcie procesu uczenia.

Zastosowanie bloku LSTM w pierwszej sieci neuronowej pozwala na uwzględnienie zależności czasowych w danych sekwencyjnych, co może być kluczowe w przypadku analizy danych o charakterze sekwencyjnym, takich jak ciągi czasowe, dane tekstowe czy sygnały czasowe. Natomiast warstwa płaska o rozmiarze 128 pozwala na dalsze przetwarzanie reprezentacji zwróconych przez blok LSTM, co może pomóc w ekstrakcji bardziej abstrakcyjnych cech i wzorców występujących w danych. Entropia krzyżowa, jako funkcja straty, ma zdolność do skutecznego kształtowania zachowania sieci neuronowej podczas procesu uczenia. Poprzez minimalizację wartości entropii krzyżowej, model jest zachęcany do generowania bardziej pewnych i trafnych predykcji, co z kolei wpływa na poprawę jakości ogólnego modelu.

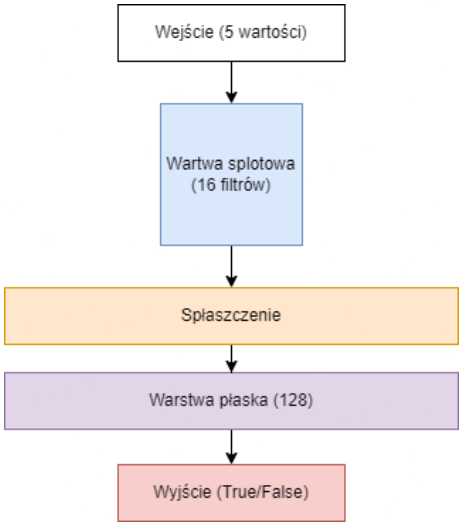
Algorytm Adam stanowi połączenie zalet kilku technik optymalizacyjnych, co przekłada się na wysoką efektywność w procesie uczenia sieci neuronowych. Dzięki możliwości wykorzystania informacji o momencie (poprzednich aktualizacjach wag) i wykorzystaniu adaptacyjnych współczynników uczenia dla poszczególnych wag, algorytm ten ma zdolność do szybszego osiągania optymalnych rozwiązań i unikania problemu zatrzymywania się w minimach lokalnych. Na rysunku 8.3 zaprezentowane architekturę sieci z wykorzystaniem bloku LSTM.



Rysunek 8.3: Architektura sieci z wykorzystaniem bloku LSTM

W tabeli 8.1 przedstawiono sieci neuronowych na zbiorze treningowym.

Druga sieć ma zbliżoną architekturę. Zamiast bloku LSTM wykorzystano blok neuronowej jednowymiarowej sieci splotowej wykorzystującej 16 filtrów. Obie sieci zostały utworzone w bibliotece *Tensorflow*. Reszta parametrów została pozostawiona domyślnie. W przypadku obu sieci trening trwał 10 epok i jest to wystarczający czas do przeprowadzenia treningu. Większa liczba epok skutkuje zbyt dużym dopasowaniem się do danych i pogorszeniem wyników. Na rysunku 8.4 zaprezentowano architekturę sieci z wykorzystaniem bloku sieci splotowej.



Rysunek 8.4: Architektura sieci z wykorzystaniem bloku CNN

Sieć z blokiem długiej pamięci krótkotrwałej okazała się najlepsza pod względem dokładności predykcji. Uzyskała również lepszą precyzję kosztem mniejszej czułości. Sieć z blokiem warstwy splotowej również osiągnęła satysfakcjonujące wyniki, jednak nieco gorsze niż sieć LSTM. Blok warstwy splotowej charakteryzuje się zdolnością do efektywnego wyodrębniania lokalnych cech i wzorców, co jest korzystne w przypadku detekcji pojazdów. Niemniej jednak nie była ona w stanie dorównać dokładności osiągniętej przez sieć LSTM. Wydajność sieci została również oceniona pod kątem precyzji i czułości. Sieć z blokiem LSTM wykazała lepszą precyzję, co oznacza, że miała mniejszą tendencję do generowania fałszywych alarmów. Z drugiej strony, sieć z blokiem warstwy splotowej wykazała się bardzo dobrym wynikiem czułości. Wykorzystując metrykę F1 można stwierdzić, że sieć z blokami LSTM jest lepsza. W tabeli 8.1 zaprezentowano wyniki badanych sieci.

Tabela 8.1: Wyniki sieci neuronowych uczonych na danych z czujnika intensywności światła

	Dokładność	Precyzja	Czułość	F1
LSTM	0,96	0,83	0,69	0,80
CNN	0,92	0,46	0,95	0,61

9. KLASYFIKACJA OBRAZU

Istnieje szereg różnych podejść do wykrywania pojazdów na obrazie. Oprócz detekcji istnieje mniej skomplikowana metoda - klasyfikacja. Choć obie metody są używane w praktyce, klasyfikacja obrazu oferuje pewne zalety i ograniczenia w porównaniu z bardziej zaawansowaną techniką detekcji.

Klasyfikacja obrazu polega na przypisywaniu obrazu do jednej z wcześniej określonych kategorii. W przypadku detekcji samochodów klasyfikacja polega na odpowiedzi na pytanie, czy na obrazie znajduje się samochód, czy też nie. Wadą takiego podejścia jest brak możliwości dokładnego wykrycia pozycji czy liczby samochodów na obrazie. Wszystkie detale, takie jak punkty brzegowe, wymiary czy orientacja samochodów, są utracone, co znacząco ogranicza możliwość dalszej analizy.

Z kolei detekcja pozwala na identyfikację i lokalizację obiektów na obrazie poprzez zaznaczenie prostokątnymi ramkami (bounding boxes). Pozwala to uzyskać dokładne informacje na temat pozycji oraz liczby samochodów na obrazie, co jest niezwykle przydatne w wielu zastosowaniach, takich jak systemy wspomagające kierowcę czy monitorowanie ruchu drogowego.

Ważną kwestią, która odróżnia klasyfikację od detekcji, jest interpretowalność decyzji podejmowanych przez algorytm. W przypadku klasyfikacji obrazu proces podejmowania decyzji jest prostszy, ponieważ model przypisuje obraz bezpośrednio do jednej kategorii. W przypadku detekcji otrzymujemy ramki z wykrytymi pojazdami. Algorytm łatwiej zrozumieć co jest szczególnie przydatne w przypadku błędnej detekcji pojazdu.

Detekcja samochodów na drodze wymaga dodatkowo precyzyjnych adnotacji, tj. oznaczenia pozycji i kształtu każdego samochodu na obrazie. Proces adnotacji detekcji jest bardziej czasochłonny i pracochłonny dla ludzi w porównaniu do adnotacji stosowanej w przypadku klasyfikacji, gdzie wystarczy jedynie oznaczyć przynależność do konkretnej kategorii. To ograniczenie może skutkować dłuższym czasem potrzebnym na stworzenie odpowiedniego zbioru danych treningowych dla modeli detekcji.

Jednak pomimo tych wad, klasyfikacja obrazu może być wciąż preferowana w niektórych zastosowaniach, szczególnie w przypadkach, gdy precyzyjna lokalizacja obiektów nie jest wymagana, a kluczową rolę odgrywa szybkość działania algorytmu czy mniejsze wymagania sprzętowe. Wybór odpowiedniej metody zależy od specyfiki zadania, dostępnych zasobów obliczeniowych oraz wymagań interpretowalności wyników.

9.1. Metoda klasyfikacji

Utworzony został zbiór danych w celu klasyfikacji obrazów. Zbiór zawiera ponad 100 tysięcy zdjęć. Wstępna klasyfikacja została dokonana na podstawie średniej jasności obrazu. Następnie błędnie zaklasyfikowane obrazy były poprawiane na stosowne klasy klas. Obrazy były klasyfikowane następująco. Na obrazie uznawano, że znajduje się pojazd, jeśli były widoczne dwa oddzielne strumienie światła. Ignorowane były pojazdy z oddali. Na rysunku 9.1 zaprezentowano przykłady zaklasyfikowanych obrazów, na których nie znajduje się pojazd.

(a) Sceneria miejska bez pojazdów

(b) Pojazd z oddali nie jest uwzględniany

(c) Sceneria poza miastem

(d) Pojazd z oddali nie jest uwzględniany

Rysunek 9.1: Przykład zaklasyfikowanych obrazów, na których nie znajduje się pojazd

Na rysunku 9.1b światło nadjeżdżającego pojazdu nie zostało uwzględnione dlatego, że nie są widoczne jeszcze dwa oddzielne strumienie światła. Podobna sytuacja występuje na rysunku 9.1d, gdzie światło jest ledwo widoczne.

Na poniższym rysunku 9.2 zostały ukazane obrazy zaklasyfikowane jako "jest pojazd":

(a) Sceneria miejska bez pojazdów

(b) Pojazd z oddali nie jest uwzględniany

(c) Sceneria poza miastem

(d) Pojazd z oddali nie jest uwzględniany

Rysunek 9.2: Przykład zaklasyfikowanych obrazów, na których znajduje się pojazd

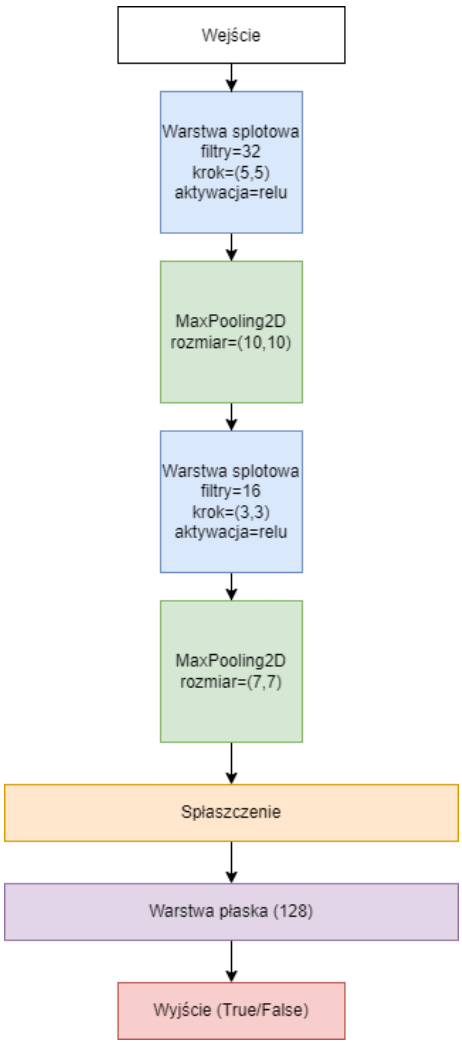
Na rysunku 9.2a oraz 9.2d widoczne są dwa oddzielne strumienie światła i obraz został zaklasyfikowany jako "jest pojazd". W przypadku pojazdów, które znajdują się blisko urządzenia rejestrującego oba obrazy 9.2b i 9.2c została nadana ta sama klasa.

9.2. Architektura sieci

W badaniu klasyfikatorów autorska sieć ma na celu sprawdzenie, jak sprawuje się bardzo prosta sieć podczas wykrywania pojazdów w porze nocnej. Zostaje ona porównana z siecią MobileNetV2 z biblioteki *Tensorflow*. Autorska sieć składa się z dwóch warstw splotowych. Pierwsza z nich składa się z 32 filtrów oraz kroku (5,5). Po warstwie splotowej została zastosowana warstwa pooling'u o wymiarach (10,10). Następnie została zastosowana druga warstwa splotowa składająca się z 16 filtrów o kroku (3,3). Obie warstwy wykorzystują funkcję aktywacji ReLU (ang. *Rectified Linear Unit*) [88], która od wielu lat znajduje wykorzystanie między innymi w sieciach neuronowych przetwarzających obrazy [89, 90]. Sieć składa się łącznie z 15 505 parametrów w wersji z obrazem wejściowym o szerokości 224 piksele, 40.081 pikseli w wersji z obrazem wejściowym o szerokości 320 pikseli i 138.385 w wersji z obrazem wejściowym o szerokości 640 pikseli. Wraz ze wzrostem rozdzielczości obrazu wejściowego wzrasta liczba

parametrów sieci neuronowej. Zastosowanie *max pooling'u* poprzedzonego warstwą splotową, zostało zaczerpnięte z architektury sieci AlexNet [91]. Warstwa płaska o rozmiarze 128 neuronów została wybrana empirycznie. W przypadku jednego wyjścia o wartości logicznej taka liczba jest wystarczająca.

Sieć MobileNetV2 ma bardziej zaawansowaną architekturę. Zamiast *max pooling'u*, w celu zmniejszenia obrazu wykorzystano warstwę depthwise. Zamiast funkcji aktywacji ReLU wykorzystano lepiej sprawdzającą się wersję ReLU6.



Rysunek 9.3: Architektura sieci splotowej utworzonej przez autora

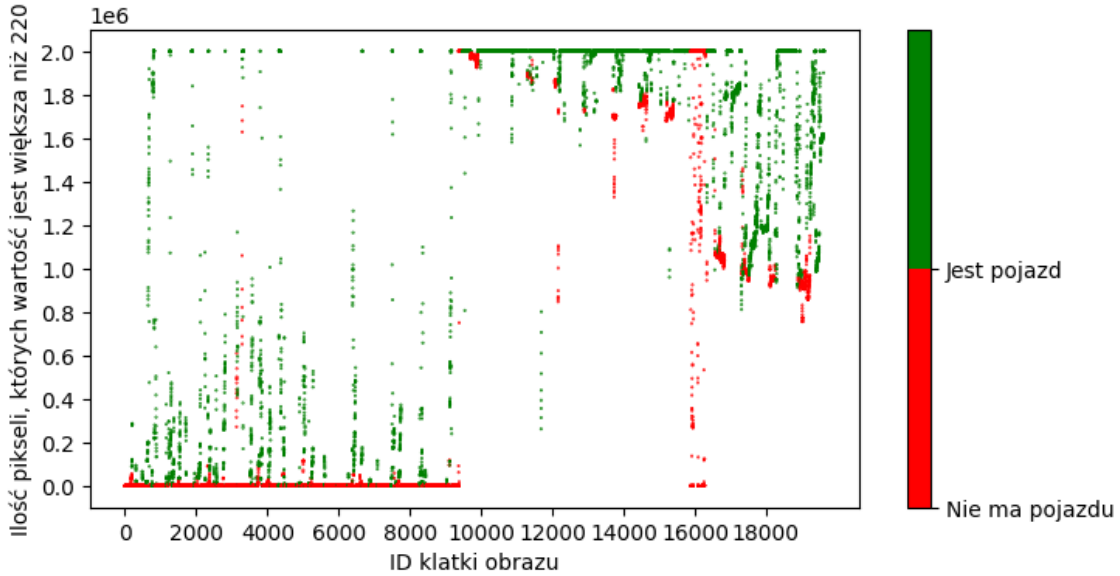
9.3. Wyniki

W poniższej tabeli zaprezentowano wyniki autorskiej sieci neuronowej w porównaniu do sieci MobileNetV2. Autorska sieć charakteryzuje się bardzo małą liczbą parametrów. Algorytm z rozmiarem wejściowym 224x224 pikseli, ma zaledwie 15 tysięcy parametrów. W przypadku sieci o tym rozmiarze wejściowym lepsze wyniki dokładności osiągnęła sieć MobileNetV2. Autorska sieć osiągnęła lepsze wyniki w precyzji kosztem niskich wyników w przypadku czułości. Porównując sieci o rozmiarze wejściowym 320x320, autorska sieć osiągnęła lepsze wyniki z uwzględnionych w pracy metryk. Sieci o największym rozmiarze wejściowym osiągnęły takie same wyniki w metryce dokładności. Sieć autorska osiąga lepszą precyzję kosztem mniejszej czułości. Najlepszą siecią według metryki F1 jest sieć MobileNetV2 o rozdzielczości wejściowej 640x640 pikseli. Autorska sieć o rozdzielczości wejściowej 640x640 pikseli osiąga lepsze wyniki od sieci MobileNetV2, pomimo że ma zdecydowanie mniej parametrów.

Sieć	Dokładność	Precyzja	Czułość	F1	Liczba parametrów
Autorska 224	0,87	0,98	0,75	0,85	15 505
Autorska 320	0,95	0,98	0,91	0,94	40 081
Autorska 640	0,95	0,97	0,92	0,94	138 385
MobileNetV2 224	0,91	0,87	0,97	0,92	959 457
MobileNetV2 320	0,92	0,88	0,97	0,92	959 457
MobileNetV2 640	0,95	0,96	0,94	0,95	959 457

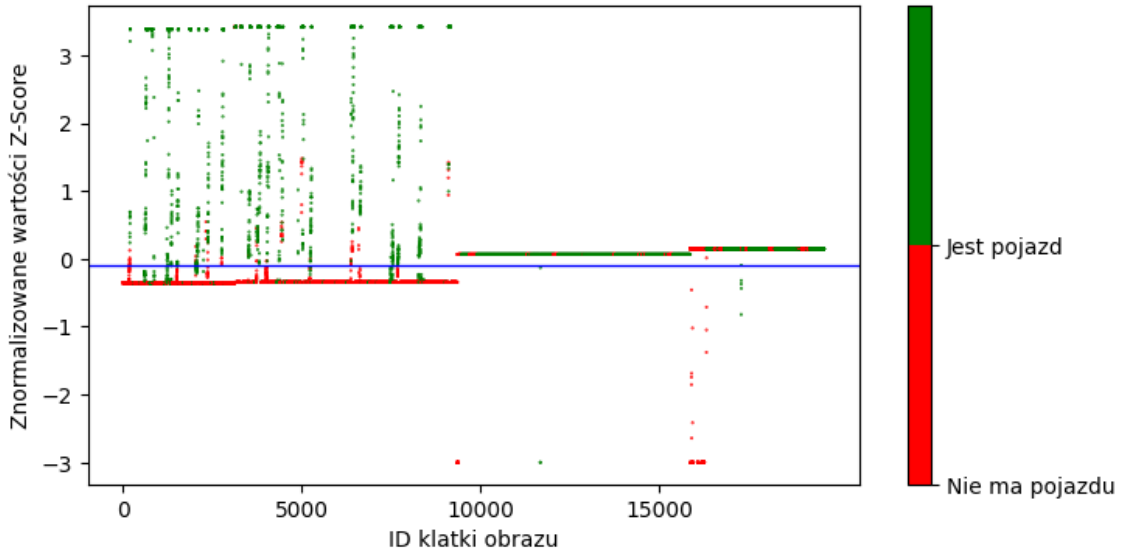
Tabela 9.1: Wyniki sieci neuronowych uczonych na danych do klasyfikacji

Sieci neuronowe klasyfikujące obraz osiągają satysfakcjonujące wyniki na testowym zbiorze danych. Mogą być dwa powody, dlaczego mniejsza sieć neuronowa z mniejszą liczbą parametrów osiąga podobne, a nawet lepsze rezultaty niż większa sieć. Po pierwsze zadanie klasyfikacji zamiast wykrywania kształtu poszukiwanych obiektów, można sprowadzić do wykrywania pojawiających się bardzo jasnych pikseli na obrazie. Na rysunku 9.4 przedstawiony został wykres liczby pikseli, gdzie wartość piksela jest większa niż 220 według klatek obrazu. Na wykresie występuje podobny problem, jak na danych zebranych za pomocą czujnika intensywności światła. Klatki przed około 10.000 zostały zebrane na wsi, gdzie zanieczyszczenie świetlne jest minimalne. Tutaj można zauważyć, że klatki, gdzie liczba pikseli jest większa niż 200, w większości są zaklasyfikowane jako "jest pojazd". Druga część klatek została zebrana w mieście, które charakteryzuje się zdecydowanie mocniejszym oświetleniem scenerii. Na tych danych o wiele trudniej jest oddzielić obie klasy.



Rysunek 9.4: Wykres przedstawiający liczbę pikseli, których wartość jest większa niż 220

Następnie dane zostały znormalizowane przy wykorzystaniu standaryzacji Z. Niebieską linią oznaczono parametr algorytmu decydujący, do której klasy należy obraz. Jeżeli wartość jest większa od -0,1 to obraz jest klasyfikowany jako "jest pojazd", w przeciwnym wypadku nadawana jest klasa "nie ma pojazdu". Na rysunku 9.5 zaprezentowano wyniki normalizacji.



Rysunek 9.5: Wykres przedstawiający wartości znormalizowane przy wykorzystaniu standaryzacji Z - niebieską linią oznaczono parametr algorytmu decydujący, do której klasy należy obraz.

Następnie sprawdzono, jak na podstawie jednej instrukcji warunkowej sprawdzającej wartość sprawuje się algorytm. W tabeli 9.2 zaprezentowane wyniki prostego algorytmu w porównaniu do autorskiej sieci splotowej.

Analizując wyniki, możemy zauważyć, że są najbardziej zbliżone do autorskiej sieci splotowej z rozmiarem wejściowym 224x224 piksele. Tak prosty algorytm może mieć zastosowanie na urządzeniach o bardzo niskiej mocy obliczeniowej i poborze energii. Wyniki we wszystkich trzech metrykach są lepsze w bardziej skomplikowanej sieci wykorzystującej sploty.

Tabela 9.2: Wyniki prostego algorytmu z jedną instrukcją warunkową w porównaniu do autorskiej sieci splotowej.

Sieć	Dokładność	Precyzja	Czułość	F1
Prosty algorytm	0,85	0,91	0,82	0,86
Autorska sieć splotowa 224	0,87	0,98	0,75	0,85

Wykorzystanie algorytmu bazującego na liczbie jasnych pikseli wiąże się z pewnymi zagrożeniami. Przede wszystkim wykonane badania zostały wykonane na małym zbiorze danych, w krótkim czasie. Kolejnym istotnym czynnikiem, który może wpłynąć na skuteczność tego rodzaju algorytmu, jest zmienność oświetlenia w trakcie różnych faz dnia. Od zachodu do wschodu słońca obserwujemy dynamiczne zmiany poziomu oświetlenia sceny, co może znacząco wpłynąć na jakość analizy obrazu. Algorytmy oparte na liczbie jasnych pikseli mogą być szczególnie podatne na takie fluktuacje, prowadząc do fałszywych wyników lub utraty obiektów z pola widzenia. Aby przeciwdziałać tym problemom, proponuje się zastosowanie technik normalizacji z oknem przesuwnym. Polega to na analizie większej liczby obrazów, które są normalizowane w odniesieniu do lokalnych kontekstów. Na przykład, normalizacja oparta na tysiącu obrazów pozwala uwzględnić różnorodność warunków oświetleniowych, co może poprawić odporność algorytmu na zmienne warunki środowiskowe. Ważne jest również, że samo rozpoznawanie obiektów na podstawie liczby jasnych pikseli nie pozwala na pełną identyfikację tych obiektów. Nie jest możliwe jednoznaczne stwierdzenie, czy obiekt jest pojazdem osobowym, ciężarówką czy jednośladem tylko na podstawie analizy jasności. Dodatkowo brak możliwości określenia, z której strony nadjeżdża pojazd, jest istotnym ograniczeniem w wielu zastosowaniach, takich jak bezpieczeństwo ruchu drogo-

wego. Algorytm oparty na liczbie jasnych pikseli może być użyteczny w niektórych kontekstach, jednak należy zdawać sobie sprawę z jego ograniczeń.

9.4. Wyniki klasyfikacji dla sieci detekcji

W testach klasyfikacji obrazów sprawdzono również sieci detekcji. Wybór algorytmów został dokonany na podstawie wyników osiągniętych w trakcie badań nad detekcją, będących punktem odniesienia w tej sferze analiz. Wybrane modele detekcji obiektów to YOLOV8s oraz jego mniej złożona wersja YOLOV8n, które wykazały się najwyższą skutecznością w odniesieniu do wskaźnika średniej precyzji (*mAP*) w ramach prowadzonych badań, jak zilustrowano w tabeli 6.4.

Badanie zostało przeprowadzone na dwóch zbiorach treningowych. Pierwszy to zbiór detekcji, który bardzo przypomina część pozamiejską zbioru klasyfikacji. Nie ma w nim obrazów scenerii typowo miejskiej z dużą ilością pojazdów, wielopasmową jezdnią i intensywnym oświetleniem tła. Celem badania sieci detekcji trenowanej na innym zbiorze jest sprawdzenie, w jakim stopniu sieć detekcji jest zdolna do generalizacji, czyli do rozpoznawania obiektów w nowych i nieznanych środowiskach, pomimo ewentualnych różnic w charakterystyce tła, oświetlenia oraz kontekstu. Drugi zbiór treningowy został rozszerzony o zdjęcia ze scenerii miejskiej. Dodane zostało 215 obrazów i każdy z nich został oznaczony w formacie YOLO. Badanie ma na celu porównanie, jak dodanie nowych obrazów zmieni wyniki w metrykach dokładności i jak dobre rezultaty sieci detekcji są względem sieci klasyfikujących obrazy.

Ważnym aspektem analizy było również uwzględnienie specyfiki zbioru danych wykorzystanych w badaniach. Mając na uwadze zróżnicowanie obrazów pozyskanych w kontekście obszaru miejskiego oraz obszaru wiejskiego, dokonano podziału zbioru testowego na dwa podzbiory. Pierwsza część zbioru miała charakter reprezentatywny dla obrazów zgromadzonych na obszarze wiejskim, druga część obejmowała obrazy zarejestrowane w kontekście miejskim. Wykonano również testy na całym zbiorze testowym, czyli obrazy zarówno z terenów pozamiejskich, jak i miejskich.

Podejście to pozwoliło na przeprowadzenie analizy w kontekście różnic w wydajności detekcji pomiędzy dwoma omawianymi modelami w zróżnicowanych warunkach. Tym samym, możliwe było osiągnięcie bardziej precyzyjnych wniosków dotyczących ogólnej efektywności oraz zastosowalności testowanych modeli w zależności od rodzaju otoczenia, w jakim miała miejsce detekcja obiektów. Istotnie, wyniki uzyskane w ramach omawianych testów dostarczyły głębszego wglądu w zdolności detekcyjne obu modeli, uwydatniając zarówno ich mocne strony, jak i obszary wymagające ewentualnych udoskonaleń w kontekście konkretnego typu scenariuszy.

W tabeli 9.3 przedstawiono wyniki sieci detekcji dla klasyfikacji obrazów. Możliwe jest wyciągnięcie kilku istotnych wniosków na temat skuteczności dwóch modeli detekcji obiektów, tj. YOLOV8s oraz YOLOV8n, w zróżnicowanych scenariuszach testowych. Zdjęcia były klasyfikowane jako "jest pojazd", jeśli sieć wykryła przynajmniej jeden pojazd na obrazie, jeśli nie został wykryty żaden obiekt, to wynik predykcji był oznaczany jako "Nie ma pojazdu".

W pierwszej kolejności, porównując ogólną dokładność, YOLOV8n osiąga wyższą wartość dokładności w porównaniu do YOLOV8s na całym zbiorze testowym. Najlepsze wyniki metryce F1 osiągnięte zostały dla sieci YOLOV8n. Wskazuje to, że mniej złożona architektura YOLOV8n przekłada się na lepszą ogólną jakość klasyfikacji. Należy zwrócić uwagę na różnice między modelami w zależności od rodzaju otoczenia. W kontekście obszarów miejskich YOLOV8n osiąga wyższą dokładność niż YOLOV8s. To może wskazywać, że mniej zaawansowana architektura sieci lepiej radzi sobie z detekcją obiektów w warunkach miejskich, gdzie scenariusze są bardziej złożone. Wartości precyzji i czułości również dostarczają istotnych informacji. W większości przypadków YOLOV8n osiąga wyższe wartości precyzji niż YOLOV8s, sugerując lepszą zdolność do identyfikacji prawdziwie pozytywnych przypadków.

Natomiast YOLOV8s wydaje się osiągać wyższą czułość na niektórych zbiorach, co może wskazywać na zdolność do wykrywania większej liczby pozytywnych przypadków, ale może prowadzić do większej liczby wyników fałszywie pozytywnych. Porównując wyniki na obszarach wiejskich i miejskich, można zauważyć, że YOLOV8n osiąga wyższą dokładność na obszarach wiejskich, podczas gdy YOLOV8s osiąga wyższą czułość na obszarach miejskich. To może sugerować, że modele te różnią się w zdolnościach detekcyjnych w zależności od charakterystyki otoczenia.

Sieci detekcji w porównaniu do sieci klasyfikującej osiągają gorsze wyniki na całym zbiorze testowym. Metryki na zbiorze na obszarze niezabudowanym są bardzo wysokie, natomiast w mieście wyniki są znacznie niższe. Algorytmy uzyskują najniższe wyniki w scenerii miejskiej. Jednak pomimo braku takich przykładów w zbiorze treningowym sieci YOLOV8n udało się osiągnąć dość dobry rezultat (spodziewana była dokładność poniżej 50% ze względu na latarnie miejskie, które mogły być pomylone ze światłami mijania).

Tabela 9.3: Wyniki sieci wytrenowanych na zbiorze detekcji i przetestowanych na zbiorze do klasyfikacji

Sieć	Zbiór testowy	Dokładność	Precyzja	Czułość	F1
YOLOV8s	cały	0,63	0,43	0,77	0,55
YOLOV8s	wieś	0,88	0,55	0,89	0,68
YOLOV8s	miasto	0,41	0,41	0,74	0,53
YOLOV8n	cały	0,78	0,75	0,82	0,78
YOLOV8n	wieś	0,90	0,66	0,87	0,75
YOLOV8n	miasto	0,67	0,77	0,81	0,79

Następnie przeprowadzono trening na rozszerzonym zbiorze danych. Wyniki sieci wytrenowanych na rozszerzonym zbiorze danych pozytywnie wpłynęły na dokładność, precyzję i czułość klasyfikacji obrazów. Obie sieci mają bardzo zbliżone wyniki i nieco lepsze ma ponownie YOLOV8n. Na wsi zostały osiągnięte niższe wyniki w metryce czułości, ale większa dokładność i precyzja. W tabeli 9.4 przedstawiono wyniki.

Tabela 9.4: Wyniki sieci wytrenowanych na rozszerzonym zbiorze detekcji i przetestowanych na zbiorze do klasyfikacji

Sieć	Zbiór testowy	Dokładność	Precyzja	Czułość	F1
YOLOV8n	cały	0,91	0,91	0,92	0,91
YOLOV8n	wieś	0,93	0,95	0,70	0,81
YOLOV8n	miasto	0,89	0,90	0,98	0,94
YOLOV8s	cały	0,90	0,91	0,91	0,91
YOLOV8s	wieś	0,92	0,91	0,70	0,79
YOLOV8s	miasto	0,89	0,91	0,97	0,94

9.5. Wyniki klasyfikacji dla algorytmów śledzenia ruchu

Częstym problemem w sieciach detekcji jest gubienie obiektu w sekwencji klatek. W celu wyeliminowania problemu fałszywie negatywnych wyników zastosowano algorytmy śledzenia.

Pierwszym z wykorzystanych algorytmów śledzenia ruchu to BoT-SORT [92] (ang. *Bounding Box Tracker - Simple Online and Realtime Tracking*). Omawiany algorytm osiągnął trzecie miejsce w renomowanym zestawieniu MOT20 [93] znanym z platformy Papers with Code. MOT (ang. *Multiple Object*

Tracking) to miara wykorzystywana do oceny wydajności algorytmów śledzenia wielu obiektów w sekwencjach wideo. Wykorzystana została implementacja [94] z *framework'u Ultralytics*.

Drugim z algorytmów kluczowych w badaniach był ByteTrack [95]. Znajduje się na czwartym miejscu w powyżej wspomnianym rankingu MOT20. Implementacja [96] algorytmu ByteTrack, pochodząca od samego autora tego rozwiązania, została wykorzystana do prowadzenia analizy w ramach badawczego eksperymentu.

W kontekście szeroko pojętej nauki i technologii śledzenie wielu obiektów w sekwencjach wideo ma kluczowe zastosowania. Przykładem jest rozwijająca się dziedzina pojazdów autonomicznych, gdzie precyzyjne i nieprzerwane śledzenie obiektów w otoczeniu jest kluczowe dla bezpiecznej i skutecznej nawigacji. Ponadto, techniki śledzenia obiektów znajdują zastosowanie również w dziedzinie monitoringu, umożliwiając efektywne monitorowanie obszarów publicznych, infrastruktury krytycznej czy obiektów przemysłowych.

W tabeli 9.5 przedstawiono wyniki sieci wytrenowanych na rozszerzonym zbiorze detekcji i przetestowanych na zbiorze do klasyfikacji. Do śledzenia obiektów wykorzystano algorytm BoT-SORT.

Tabela 9.5: Wyniki sieci wytrenowanych na rozszerzonym zbiorze detekcji i przetestowanych na zbiorze do klasyfikacji. Do śledzenia obiektów wykorzystano algorytm BoT-SORT

Sieć	Zbiór testowy	Dokładność	Precyzja	Czułość	F1
YOLOV8n	cały	0,89	0,85	0,96	0,90
YOLOV8n	wieś	0,94	0,92	0,80	0,86
YOLOV8n	miasto	0,84	0,84	0,99	0,91
YOLOV8s	cały	0,91	0,88	0,95	0,91
YOLOV8s	wieś	0,93	0,89	0,79	0,84
YOLOV8s	miasto	0,88	0,88	0,99	0,93

W przypadku sieci YOLOV8n na całym zbiorze i obszarach pozamiejskich i miejskich spadła dokładność i precyzja, a wzrosła czułość. W sieci YOLOV8s dokładność różniła się w nieznacznym stopniu i wzrosła czułość kosztem precyzji. W tabeli 9.6 przedstawiono wyniki sieci wytrenowanych na rozszerzonym zbiorze detekcji i przetestowanych na zbiorze do klasyfikacji. Do śledzenia obiektów wykorzystano algorytm ByteTrack.

W przypadku sieci YOLOV8s i YOLOV8n nie zaszły żadne zmiany w metrykach dokładności, precyzji i czujności. Z punktu widzenia zastosowanych metryk nie ma znaczenia, który algorytm śledzenia zostanie wykorzystany.

Tabela 9.6: Wyniki sieci wytrenowanych na rozszerzonym zbiorze detekcji i przetestowanych na zbiorze do klasyfikacji. Do śledzenia obiektów wykorzystano algorytm ByteTrack

Sieć	Zbiór testowy	Dokładność	Precyzja	Czułość	F1
YOLOV8n	cały	0,89	0,85	0,96	0,90
YOLOV8n	wieś	0,94	0,92	0,80	0,86
YOLOV8n	miasto	0,84	0,84	0,99	0,91
YOLOV8s	cały	0,91	0,88	0,95	0,91
YOLOV8s	wieś	0,93	0,89	0,79	0,84
YOLOV8s	miasto	0,88	0,88	0,99	0,93

Algorytmy śledzenia w przypadku testowanych sieci obniżały nieznacznie dokładność i precyzję

i zwiększały czułość detekcji. W przypadku metryki F1, można wnioskować, że wyniki na całym zbiorze są nieco gorsze z wykorzystaniem algorytmów śledzenia ruchu. Lepsze wyniki na wsi mogły być spowodowane mniejszą ilością zanieczyszczeń świetlnych, co jest zauważalne w wysokich wartościach metryk precyzji. Gorsze wyniki w mieście mogły być spowodowane dodatkowym oświetleniem, które było wykrywane jako obiekt i następnie śledzone przez algorytm. Wysokie wyniki w metryce czułości w stosunku do wartości w metryce precyzji potwierdzają tę tezę.

10. PODSUMOWANIE

W pracy sprawdzono wybrane metody wykrywania pojazdów w porze nocnej, które mogą być wykorzystane w celu zwiększenia widoczności pieszych na przejściach. W tym celu zebrano nagrania za pomocą kamery i czujnika intensywności światła. Z zebranych nagrań dokonano adnotacji i uzyskano 2000 pojazdów w zbiorze detekcji, 100 000 obrazów w zbiorze do klasyfikacji obrazu i ponad 14 000 próbek intensywności światła.

Zdefiniowano kryteria i wybrano jako platformę docelową *Nvidia Jetson Nano*,*Raspberry Pi 4* i *Google Coral TPU*. Wybrane urządzenia dobrze sprawdziły się w testach wydajnościowych i algorytmy były łatwe w implementacji, dzięki dobremu wsparciu producentów. Duża ilość mocy obliczeniowej i pamięci operacyjnej umożliwiła uruchomienie sieci detekcji o dużych rozmiarach i porównanie, czy zastosowanie większej sieci przynosi dodatkowe korzyści. Jednym z kryteriów była odporność środowiskowa. Wybrane urządzenia mogą działać w najniższej temperaturze 0 $^{\circ}C$, jeśli by miały działać poniżej temperatury zamarzania wody, to powinny być dodatkowo zabezpieczone przed czynnikami zewnętrznymi. Na rynku dostępne jest wiele alternatywnych rozwiązań, które są bardziej ekonomiczne i są odporniejsze na zewnętrzne warunki pogodowe.

Porównując wydajności sieci detekcji, zauważono, że modele wyeksportowane do formatów ogólnego przeznaczenia (pb i *pt*) osiągają większą wydajność niż modele wyeksportowane do formatów mobilnych (tflite i onnx). W testach nie uwzględniono ilości wykorzystywanej pamięci operacyjnej podczas inferencji. Możliwe, że wyniki mogłyby się różnić na urządzeniach z mniejszą ilością pamięci RAM.

Wszystkie sieci uruchomione na urządzeniu *Nvidia Jetson Nano* spełniły wymagania co do wydajności (powyżej 0,37 klatek na sekundę), a na*Raspberry Pi 4* nie spełniły wymagań dwie sieci - SSD ResNet50 ze względu na wolną inferencję (0,26 klatek na sekundę) i YOLOV8s prawdopodobnie ze względu na zbyt małą ilość pamięci operacyjnej. Spośród sieci wyeksportowanych do formatu mobilnego (tflite i onnx), to wymagań dotyczących wydajności nie spełniły sieci CenterNet HourGlass104 i SSD ResNet50. Udało się natomiast w formacie mobilnym (onnx) uruchomić sieć YOLOV8s na wszystkich platformach.

Sieci zostały również wyeksportowane do formatu tflite zgodnego z instrukcjami obsługiwanymi przez TPU. Oczekiwano dużych zysków wydajnościowych, jednak wiele instrukcji wytrenowanych sieci nie było kompatybilnych do eksportu. Wskutek tego w niektórych sieciach uzyskano niewielkie przyspieszenie inferencji, a w niektórych nawet spowolnienie. Wynika to z faktu, że instrukcje, które nie były obsługiwane przez TPU, były wykonywane na CPU urządzenia. Duży narzut komunikacyjny spowodował, że pomimo potencjalnie większej mocy obliczeniowej wyniki wydajności inferencji nieznacznie się zmieniły.

W pracy nie udało się poprawnie wyeksportować modeli wykorzystywanych w badaniu. W celu oszacowania, jakie wyniki są możliwe do uzyskania, sprawdzono podobne architektury sieci, które były wyeksportowane przez producenta *framework'a Tensorflow*. Te sieci działały prawidłowo i wykorzystanie akceleratora przyniosło bardzo duże zyski wydajnościowe - od 2 do 18 razy szybsza inferencja w zależności od wykorzystanej sieci. Gdyby badanie było przeprowadzane drugi raz, to sieci byłyby najpierw sprawdzone, czy można je poprawnie wyeksportować do formatu obsługującego instrukcje wykonywane przez TPU.

Wytypowano najlepsze sieci pod kątem szybkości inferencji, są nimi SSD MobileNetV2, SSD MobileNetV2 FPNLite, CenterNet MobileNetV2 FPN, EfficentDet D0, YOLOV8n. Na szczególną uwagę zasługuje sieć SSD MobileNetV2, która osiągnęła bardzo wysokie wyniki na CPU i na TPU.

Sama wydajność nie jest wystarczająca, dlatego przeprowadzono testy na metrykach dla sieci detekcji. Wśród przebadanych sieci faworytem jest sieć YOLOV8, które osiągnęła najwyższe *mAP* (0,423) w wersji "small"i niewiele mniej w wersji "nano"(0,415 *mAP*). Na uwagę zasługują również sieci, które osiągnęły niższe wyniki, ale nadal dobre - CenterNet MobileNetV2 FPN (0,300 *mAP*), EfficentDet D1 (0,271 *mAP*), EfficentDet D0 (0,251 *mAP*), SSD MobileNetV2 FPNLite (0,218 lub 0,246 *mAP* w zależności od rozmiaru obrazu wejściowego) i CenterNet HourGlass104 (0,234 *mAP*).

Sprawdzono jak metody klasyfikacji radzą sobie w wykrywaniu pojazdów. Z badań wynika, że sieć stworzona przez autora i MobileNetV2 okazały się lepsze od sieci detekcji YOLOV8s w klasyfikacji binarnej obrazów. Taki stan rzeczy może wynikać z tego, że tworzenie zbioru do klasyfikacji jest łatwiejsze i udało się zebrać więcej próbek. Istnieją jednak pewne zagrożenia wynikające z wykorzystywania klasyfikatorów zamiast sieci detekcji. Algorytm jest trudniejszy do rozwijania ze względu na trudniejszą wytłumaczalność decyzji.

Ponadto został sprawdzony bardzo prosty algorytm, który zliczał ilość pikseli, o wartościach większych niż 220 i na tej podstawie dokonywał klasyfikacji binarnej. Badanie tego algorytmu dowiodło, że w czasach dominacji skomplikowanych sieci neuronowych jest możliwość wykorzystania bardzo prostych algorytmów na platformach o bardzo niskiej mocy obliczeniowej. Proste rozwiązanie okazało się porównywalne z autorską siecią splotową (z obrazem wejściowych o rozmiarze 224 piksele). W metryce F1 osiągnął lepszy wynik, ale gorszy w metryce dokładności. Wykorzystanie takiego algorytmu wiąże się z wieloma zagrożeniami. Parametr wyznaczający liczbę jasnych pikseli, która decydowałaby o tym, czy na obrazie jest pojazd, musi być ustanowiona osobno dla każdego miejsca instalacji urządzenia. Ponadto w przypadku pojawienia się dodatkowego źródła światła (księżyc, latarnia, baner reklamowy) algorytm mógłby podawać całkowicie nieprawdzie informacje i potencjalnie stanowić zagrożenie dla uczestników ruchu.

Uwzględniając wydajność i osiągnięte wyniki wytypowano następujące sieci:

- YOLOV8 - do zastosowań na bardziej wydajnych platformach sprzętowych*Raspberry Pi 4* lub *Nvidia Jetson Nano* (4GB pamięci RAM lub więcej), gdzie zależy na bardzo wysokiej dokładności detekcji,
- SSD MobileNetV2 (z wykorzystaniem lub bez FPNLite) - do zastosowań na urządzeniach docelowych z mniejszą ilością pamięci operacyjnej niż 4GB i opcjonalnie z akceleratorem sprzętowym TPU,
- Autorska sieć klasyfikacji lub MobileNetV2 - do zastosowań na urządzeniach z bardzo małymi zasobami sprzętowymi jak Arduino Portenta H7.

Sprawdzone zostały również sieci wykorzystujące dane z czujnika intensywności światła. Osiągnęły one gorsze wyniki niż sieci algorytmy wykorzystujące obrazy. Takie rozwiązanie może mieć zastosowanie w urządzeniach o bardzo małej mocy obliczeniowej lub jako drugi algorytm wspierający decyzję o wykryciu pojazdu.

Pomyślnie udało się dokonać optymalizacji sieci poprzez redukcję liczby kolorów w obrazie wejściowym. Dzięki zmniejszeniu liczby kolorów z 256 do 16 udało się uzyskać lepsze wyniki w metryce *mAP*. Obniżenie liczby kolorów może potencjalnie mieć korzystny wpływ szybkości sieci, jeśli obliczenia tego typu byłyby wspierane.

Dalszy przebieg pracy skupiłby się na badaniu optymalizacji sieci detekcji poprzez zmniejszenie ich rozmiarów i zwiększeniu wyników w metrykach dokładności oraz dokładniejszej analizie algorytmów śledzenia obiektów. Na platformach docelowych nie udało się wykorzystać kart graficznych podczas infe-

rencji. Z tego powodu skupiono by się na wykorzystaniu pełnego potencjału obliczeniowego urządzenia. Wykorzystywane urządzenia mogły osiągnąć znacznie wyższe wyniki podczas inferencji.

SPIS RYSUNKÓW

SPIS TABEL

BIBLIOGRAFIA

[1] Park pojazdów zarejestrowanych w Polsce 1990 - 2020, https://www.pzpm.org.pl/pl/Rynek-motoryzacyjny/Park-pojazdow-zarejestrowanych/Park-pojazdow-zarejestrowanych-w-Polsce-1990-2020-xlsx, dostęp dnia 05.03.2023.

[2] OECD (2023), Passenger car registrations (indicator). doi: 10.1787/c58fcf22-en (dostęp dnia 05.03.2023).

[3] Transport drogowy 2018-2019 https://stat.gov.pl/download/gfx/portalinformacyjny/pl/defaultaktualnosci/5511/6/6/1/td_w_pl_2018_2019.pdf, dostęp dnia 05.03.2023.

[4] Irtad road safety anual report ,https://www.itf-oecd.org/sites/default/files/docs/irtad-road-safety-annual-report-2022.pdf, dostęp dnia 05.03.2023.

[5] Komenda główna policji biuro ruchu drogowego wypadki drogowe w polsce w 2022 roku, https://statystyka.policja.pl/download/20/403374/Wypadkidrogowe2022.pdf , dostęp dnia 08.08.2023.

[6] Ana I. Maqueda, Carlos R. del Blanco, Fernando Jaureguizar, and Narciso Garcia. Structured learning via convolutional neural networks for vehicle detection, 2017.

[7] Xiaowei Hu, Xuemiao Xu, Yongjie Xiao, Hao Chen, Shengfeng He, Jing Qin, and Pheng Ann Heng. Sinet: A scale-insensitive convolutional neural network for fast vehicle detection, 2019.

[8] Hossein Tehrani Niknejad, Koji Takahashi, Seiichi Mita, and David McAllester. Vehicle detection and tracking at nighttime for urban autonomous driving, 2011.

[9] Ravi Kumar Satzoda and Mohan Manubhai Trivedi. Looking at vehicles in the night: Detection and dynamics of rear lights, 2019.

[10] Gunnar Rätsch, Takashi Onoda, and Klaus-Robert Müller. Soft margins for adaboost, 03 2001.

[11] Xiaofei Li, Lingxi Li, Fabian Flohr, Jianqiang Wang, Hui Xiong, Morys Bernhard, Shuyue Pan, Dariu M. Gavrila, and Keqiang Li. A unified framework for concurrent pedestrian and cyclist detection, 2017.

[12] Hulin Kuang, Xianshi Zhang, Yong-Jie Li, Leanne Lai Hang Chan, and Hong Yan. Nighttime vehicle detection based on bio-inspired image enhancement and weighted score-level feature fusion, 2017.

[13] Andrés Bell, Tomás Mantecón, César Díaz, Carlos R. del Blanco, Fernando Jaureguizar, and Narciso García. A novel system for nighttime vehicle detection based on foveal classifiers with real-time performance, 2022.

[14] Jin-Woo Park and Byung Cheol Song. Night-time vehicle detection using low exposure video enhancement and lamp detection. In *2016 International Conference on Electronics, Information, and Communications (ICEIC)*, pages 1–2, 2016.

[15] Wei Zhang, Q. M. J. Wu, Guanghui Wang, and Xinge You. Tracking and pairing vehicle headlight in night scenes, 2012.

[16] Mahdi Rezaei, Mutsuhiro Terauchi, and Reinhard Klette. Robust vehicle detection and distance estimation under challenging lighting conditions, 2015.

[17] Chih-Chang Yu, Hsu-Yung Cheng, and Yi-Fan Jian. Raindrop-tampered scene detection and traffic flow estimation for nighttime traffic surveillance, 2015.

[18] Xuerui Dai, Xue Yuan, Jing Zhang, and Liping Zhang. Improving the performance of vehicle detection system in bad weathers, 2016.

[19] Komenda główna policji biuro ruchu drogowego wypadki drogowe w polsce w 2020 roku, https://statystyka.policja.pl/download/20/361900/Wypadkidrogowe2020.pdf , dostęp dnia 12.09.2023.

[20] Komenda główna policji biuro ruchu drogowego wypadki drogowe w polsce w 2021 roku, https://statystyka.policja.pl/download/20/381967/Wypadkidrogowe2021.pdf , dostęp dnia 12.09.2023.

[21] Palak Dave, Naga Mounika Gella, Nihil Saboo, and Apurba Das. A novel algorithm for night time vehicle detection even with one non-functional taillight by ciof (color inherited optical flow), 2016.

[22] Che-Tsung Lin, Sheng-Wei Huang, Yen-Yi Wu, and Shang-Hong Lai. Gan-based day-to-night image style transfer for nighttime vehicle detection. *IEEE Transactions on Intelligent Transportation Systems*, 22(2):951–963, 2021.

[23] Joseph Redmon and Ali Farhadi. Yolo9000: Better, faster, stronger. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 6517–6525, 2017.

[24] Kefu Yi, Kai Luo, Tuo Chen, and Rongdong Hu. An improved yolox model and domain transfer strategy for nighttime pedestrian and vehicle detection. *Applied Sciences*, 12(23), 2022.

[25] Zheng Ge, Songtao Liu, Feng Wang, Zeming Li, and Jian Sun. Yolox: Exceeding yolo series in 2021. *arXiv preprint arXiv:2107.08430*, 2021.

[26] Yan Miao, Fu Liu, Tao Hou, Lu Liu, and Yun Liu. A nighttime vehicle detection method based on yolo v3, 2020.

[27] Joseph Redmon and Ali Farhadi. Yolov3: An incremental improvement, 2018.

[28] Kai Ding and TaiLin Han. A lightweight vehicle detection algorithm combining image enhancement and multi-scale feature fusion, 2023.

[29] Sabrie Soloman. *Sensors Handbook*. McGraw-Hill, Inc., USA, 2 edition, 2009.

[30] Arduino portenta h7 - vision shield, https://docs.arduino.cc/tutorials/portenta-vision-shield/getting-started-camera , dostęp dnia 12.09.2023.

[31] TensorFlow, https://www.tensorflow.org/, dostęp dnia 05.03.2023.

[32] TensorFlow Lite, https://www.tensorflow.org/lite, dostęp dnia 05.03.2023.

[33] PyTorch, https://pytorch.org/, dostęp dnia 05.03.2023.

[34] Prawo ruchu drogowego, art. 20, ust. 1 (dz.u. z 2022 r. poz. 988).

[35] Charles Arthur Nagler and William Merle Nagler. Reaction time measurements, 1973.

[36] Cindy Orosy-Fildes and Robert W. Allan. Psychology of computer use: Xii. videogame play: Human reaction time to visual stimuli, 1989.

[37] Nickolay Podoprigora, Viktor Dobromirov, Alexander Pushkarev, and Vladimir Lozhkin. Methods of assessing the influence of operational factors on brake system efficiency in investigating traffic accidents. *Transportation Research Procedia*, 20:516–522, 2017. 12th International Conference "Organization and Traffic Safety Management in large cities", SPbOTSIC-2016, 28-30 September 2016, St. Petersburg, Russia.

[38] Arduino Portenta H7, https://docs.arduino.cc/hardware/portenta-h7, dostęp dnia 26.03.2023.

[39] Raspberry pi 4, https://docs.arduino.cc/hardware/portenta-h7, dostęp dnia 27.03.2023.

[40] NVIDIA Jetson Nano, https://developer.nvidia.com/embedded/jetson-nano-developer-kit, dostęp dnia 07.03.2023.

[41] NVIDIA Jetson Nano, https://developer.nvidia.com/embedded/jetson-modules , dostęp dnia 07.03.2023.

[42] T Maragatham, P Balasubramanie, and M Vivekanandhan. Iot based home automation system using raspberry pi 4, feb 2021.

[43] Mallikarjun Anandhalli and Vishwanath P. Baligar. A novel approach in real-time vehicle detection and tracking using raspberry pi. *Alexandria Engineering Journal*, 57(3):1597–1607, 2018.

[44] Apeksha P Kulkarni and Vishwanath P Baligar. Real time vehicle detection, tracking and counting using raspberry-pi, 2020.

[45] Muhammad Syihabuddin Az Zuhair, Andi Widiyanto, and Setiya Nugroho. Comparison of tensorflow and tensorflow lite for object detection on raspberry pi 4.

[46] Nvidia-ai-iot/jetbot, https://github.com/NVIDIA-AI-IOT/jetbot, dostęp dnia 27.03.2023.

[47] Stephen Cass. Nvidia makes it easy to embed ai: The jetson nano packs a lot of machine-learning power into diy projects - [hands on], 2020.

[48] Ahmet Ali Süzen, Burhan Duman, and Betül Şen. Benchmark analysis of jetson tx2, jetson nano and raspberry pi using deep-cnn, 2020.

[49] Antti Martikkala, Joe David, Andrei Lobov, Minna Lanz, and Iñigo Flores Ituarte. Trends for low-cost and open-source iot solutions development for industry 4.0, 2021. FAIM 2021.

[50] Anargyros Gkogkidis, Vasileios Tsoukas, Stefanos Papafotikas, Eleni Boumpa, and Athanasios Kakarountas. A tinyml-based system for gas leakage detection, 2022.

[51] Idrissi Idriss, Mostafa Azizi, and Mostafa Moussaoui Omar. A lightweight optimized deep learning-based host-intrusion detection system deployed on the edge for iot, 2021.

[52] Google Edge TPU - Coral, https://coral.ai/products/accelerator/ , dostęp dnia 29.05.2023.

[53] Kiran Seshadri, Berkin Akin, James Laudon, Ravi Narayanaswami, and Amir Yazdanbakhsh. An evaluation of edge tpu accelerators for convolutional neural networks. In *2022 IEEE International Symposium on Workload Characterization (IISWC)*, pages 79–91, 2022.

[54] Google edge tpu - specyfikacja, https://coral.ai/docs/accelerator/datasheet/ , dostęp dnia 20.07.2023.

[55] Apds-9960 - specyfikacja, https://cdn.sparkfun.com/assets/learn_tutorials/3/2/1/Avago-APDS-9960-datasheet.pdf , dostęp dnia 08.08.2023.

[56] Pascal VOC 2007, http://host.robots.ox.ac.uk/pascal/VOC/voc2007/ , dostęp dnia 26.04.2023.

[57] Metryki COCO, https://cocodataset.org/detection-eval , dostęp dnia 03.04.2023.

[58] Tensorflow 2 detection model zoo, https://github.com/tensorflow/models/blob/master/research/object_detection/g3doc/tf2_detection_zoo.md , dostęp dnia 03.04.2023.

[59] Tsung-Yi Lin, Michael Maire, Serge Belongie, Lubomir Bourdev, Ross Girshick, James Hays, Pietro Perona, Deva Ramanan, C. Lawrence Zitnick, and Piotr Dollár. Microsoft coco: Common objects in context, 2015.

[60] Pedro F. Felzenszwalb, Ross B. Girshick, David McAllester, and Deva Ramanan. Object detection with discriminatively trained part-based models, 2010.

[61] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection, 2016.

[62] YOLOv5, https://github.com/ultralytics/yolov5, dostęp dnia 29.03.2023.

[63] YOLOv8, https://github.com/ultralytics/ultralytics, dostęp dnia 29.03.2023.

[64] Mingxing Tan, Ruoming Pang, and Quoc V. Le. Efficientdet: Scalable and efficient object detection, 2020.

[65] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. Mobilenetv2: Inverted residuals and linear bottlenecks, 2019.

[66] Andrew G. Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications, 2017.

[67] Alejandro Newell, Kaiyu Yang, and Jia Deng. Stacked hourglass networks for human pose estimation, 2016.

[68] Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Rich feature hierarchies for accurate object detection and semantic segmentation, 2014.

[69] Ross Girshick. Fast r-cnn, 2015.

[70] H. Andrews and C. Patterson. Singular value decomposition (svd) image coding, 1976.

[71] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Spatial pyramid pooling in deep convolutional networks for visual recognition, 2014.

[72] Shaoqing Ren, Kaiming He, Ross B. Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks., 2015.

[73] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, and Alexander C. Berg. Ssd: Single shot multibox detector, 2016.

[74] Tsung-Yi Lin, Piotr Dollár, Ross B. Girshick, Kaiming He, Bharath Hariharan, and Serge J. Belongie. Feature pyramid networks for object detection, 2016.

[75] Kaiwen Duan, Song Bai, Lingxi Xie, Honggang Qi, Qingming Huang, and Qi Tian. Centernet: Keypoint triplets for object detection, 2019.

[76] Hei Law and Jia Deng. Cornernet: Detecting objects as paired keypoints, 2019.

[77] Xin Lu, Buyu Li, Yuxin Yue, Quanquan Li, and Junjie Yan. Grid r-cnn, 2018.

[78] Opencv, `https://opencv.org/` , dostęp dnia 17.08.2023.

[79] Google Coral - maximum operating frequency, `https://coral.ai/docs/accelerator/get-started` , dostęp dnia 29.05.2023.

[80] Onnx, `https://onnx.ai/` , dostęp dnia 29.05.2023.

[81] Edge tpu compiler - coral, `https://coral.ai/docs/edgetpu/compiler/` , dostęp dnia 29.05.2023.

[82] Gillings Michael R. Fourment Mathieu. A comparison of common programming languages used in bioinformatics. In *2022 IEEE International Symposium on Workload Characterization (IISWC)*, 2008.

[83] Thomas F. Collura PhD, PhD Joseph Guan MM.ED, Jeffrey Tarrant PhD, John Bailey PhD, and Fred Starr MD. Eeg biofeedback case studies using live z-score training and a normative database. *Journal of Neurotherapy*, 14(1):22–46, 2010.

[84] Edward I. Altman, Małgorzata Iwanicz-Drozdowska, Erkki K. Laitinen, and Arto Suvas. Financial distress prediction in an international context: A review and empirical analysis of altman's z-score model. *Journal of International Financial Management & Accounting*, 28(2):131–171, 2017.

[85] Ganga Bhavani and Chris Tabi. M-score and z-score for detection of accounting fraud. *Accountancy Business and the Public Interest*, pages 68–86, 07 2017.

[86] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Computation*, 9(8):1735–1780, 1997.

[87] Sashank J. Reddi, Satyen Kale, and Sanjiv Kumar. On the convergence of adam and beyond, 2019.

[88] Richard Hahnloser, Rahul Sarpeshkar, Misha Mahowald, Rodney Douglas, and H. Seung. Digital selection and analogue amplification coexist in a cortex-inspired silicon circuit. *Nature*, 405:947–51, 07 2000.

[89] Kevin Jarrett, Koray Kavukcuoglu, Marc'Aurelio Ranzato, and Yann LeCun. What is the best multi-stage architecture for object recognition? In *2009 IEEE 12th International Conference on Computer Vision*, pages 2146–2153, 2009.

[90] Vinod Nair and Geoffrey Hinton. Rectified linear units improve restricted boltzmann machines vinod nair. In ,, volume 27, pages 807–814, 06 2010.

[91] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C.J. Burges, L. Bottou, and K.Q. Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 25. Curran Associates, Inc., 2012.

[92] Nir Aharon, Roy Orfaig, and Ben-Zion Bobrovsky. Bot-sort: Robust associations multi-pedestrian tracking, 2022.

[93] Paperswithcode - ranking MOT17, `https://paperswithcode.com/sota/multi-object-tracking-on-mot20-1` , dostęp dnia 20.08.2023.

[94] Ultralytics - BoT-SORT implementacja, `https://github.com/ultralytics/ultralytics/blob/main/ultralytics/trackers/bot_sort.py` , dostęp dnia 20.08.2023.

[95] Yifu Zhang, Peize Sun, Yi Jiang, Dongdong Yu, Fucheng Weng, Zehuan Yuan, Ping Luo, Wenyu Liu, and Xinggang Wang. Bytetrack: Multi-object tracking by associating every detection box, 2022.

[96] ByteTrack implementacja, `https://github.com/ifzhang/ByteTrack` , dostęp dnia 20.08.2023.