

A G H

AKADEMIA GÓRNICZO-HUTNICZA IM. STANISŁAWA STASZICA W KRAKOWIE

WYDZIAŁ INFORMATYKI, ELEKTRONIKI I TELEKOMUNIKACJI

Praca dyplomowa

*Analiza ruchu i postoju pojazdów w miejskiej przestrzeni parkingowej
Analysis of Vehicle Movement and Parking in the City Parking Space*

Autor:

Jan Wojdat

Kierunek studiów:

Telenformatyka

Opiekun pracy:

prof. dr hab. Mikołaj Leszczuk

Kraków, 2024

Uprzedzony o odpowiedzialności karnej na podstawie art. 115 ust. 1 i 2 ustawy z dnia 4 lutego 1994 r. o prawie autorskim i prawach pokrewnych (t.j. Dz.U. z 2006 r. Nr 90, poz. 631 z późn. zm.): „Kto przywłaszcza sobie autorstwo albo wprowadza w błąd co do autorstwa całości lub części cudzego utworu albo artystycznego wykonania, podlega grzywnie, karze ograniczenia wolności albo pozbawienia wolności do lat 3. Tej samej karze podlega, kto rozpośrednia bez podania nazwiska lub pseudonimu twórcy cudzy utwór w wersji oryginalnej albo w postaci opracowania, artystycznego wykonania albo publicznie zniekształca taki utwór, artystyczne wykonanie, fonogram, videogram lub nadanie.”, a także uprzedzony o odpowiedzialności dyscyplinarnej na podstawie art. 211 ust. 1 ustawy z dnia 27 lipca 2005 r. Prawo o szkolnictwie wyższym (t.j. Dz. U. z 2012 r. poz. 572, z późn. zm.): „Za naruszenie przepisów obowiązujących w uczelni oraz za czyny uchybiające godności studenta student ponosi odpowiedzialność dyscyplinarną przed komisją dyscyplinarną albo przed sądem koleżeńskim samorządu studenckiego, zwanym dalej «sądem koleżeńskim».”, oświadczam, że niniejszą pracę dyplomową wykonałem(-am) osobiście i samodzielnie i że nie korzystałem(-am) ze źródeł innych niż wymienione w pracy.

Serdecznie dziękuję swoim rodzicom za nieustanne wspieranie mnie. To dzięki Wam mogę, po 3,5 roku, złożyć tę pracę.

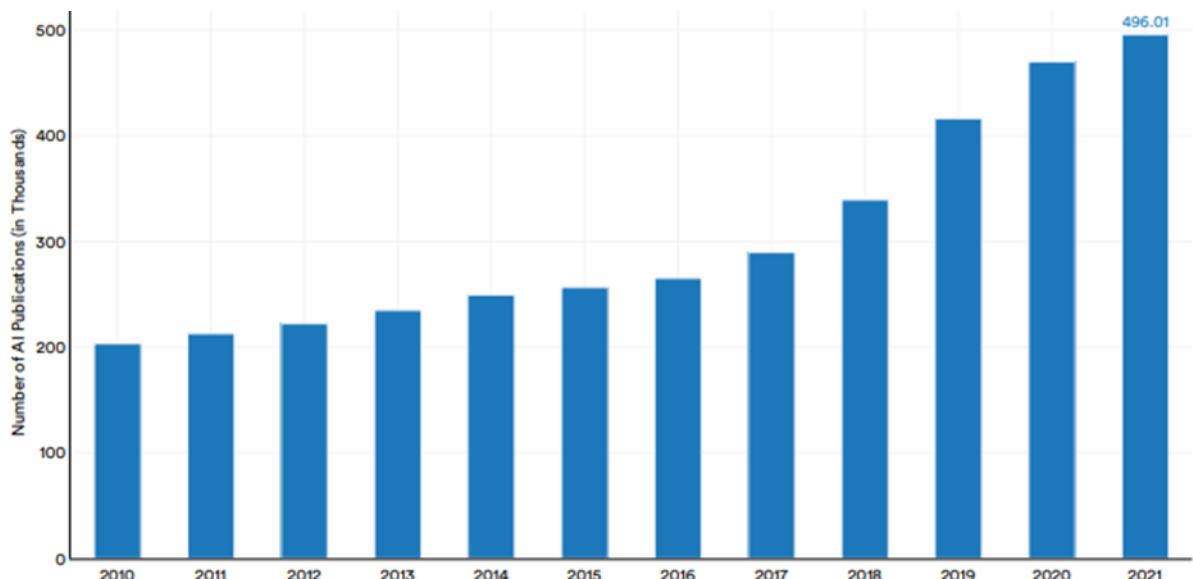
Spis treści

1. Wstęp.....	7
2. Zagadnienia teoretyczne.....	9
2.1. Sztuczne sieci neuronowe.....	9
2.2. Sieci splotowe.....	11
2.2.1. Przykładowe architektury sieci CNN.....	12
2.3. Modele detekcji obiektów	12
2.4. Metryki w sieciach neuronowych.....	13
2.5. Transformacja Hough'a	14
3. Stan dziedziny, założenia, cele, architektura	16
3.1. Założenia i cele.....	17
3.2. Wykorzystane narzędzia	18
4. Implementacja.....	20
4.1. Proponowana architektura	20
4.2. Detekcja linii	20
4.3. Model I - binarna klasyfikacja zajętości miejsc	22
4.3.1. Przygotowanie zbiorów: uczącego i walidacyjnego	22
4.3.2. Przygotowanie zbioru testowego	22
4.3.3. Propozycja I – prosta sieć CNN.....	23
4.3.4. Propozycja II – model oparty o architekturę AlexNet	32
4.3.5. Propozycja III – model oparty o architekturę EfficientNet.....	33
4.3.6. Porównanie wyników.....	35
4.4. Detekcja pojazdów poza miejscami parkingowymi	35
4.5. Trzeci model – klasyfikacja pojazdów poza miejscami parkingowymi	36
4.5.1. Przygotowanie zbiorów danych	36
4.6. Pełna implementacja.....	39
4.6.1. Wersja I - bazująca na GPU	39
4.6.2. Wersja II - działająca na CPU	39

4.7. Pełna wersja demonstracyjna.....	40
5. Podsumowanie.....	42

1. Wstęp

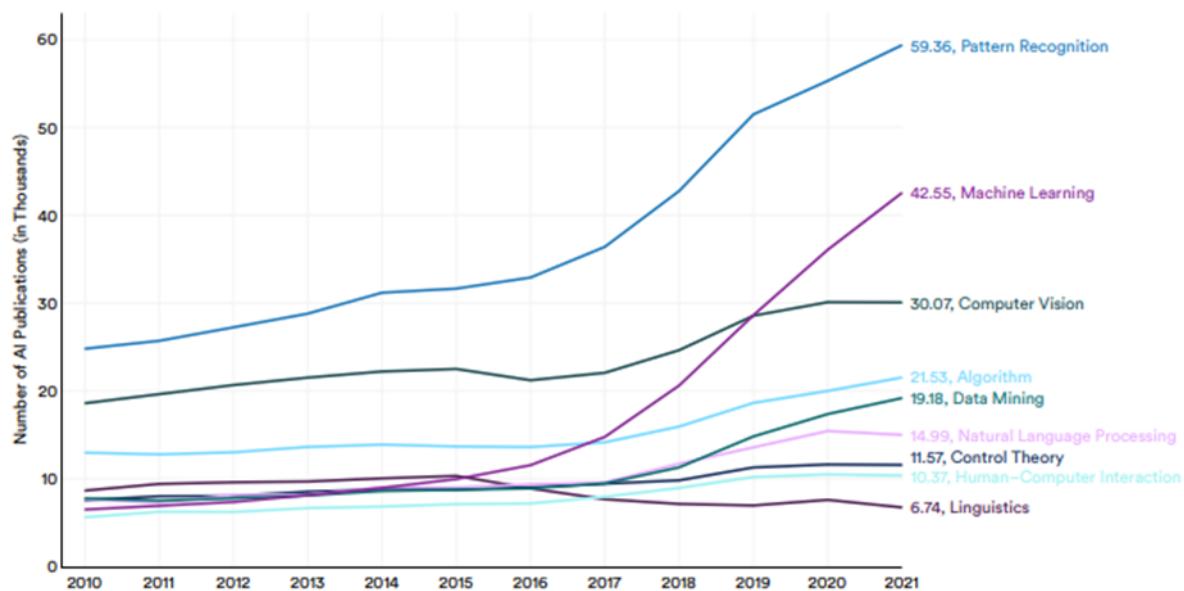
Na przestrzeni ostatnich kilku lat możemy zaobserwować znaczący wzrost zainteresowania technologią sztucznej inteligencji. Lata badań i rozwoju sprawiły, że przestaje ona być już tylko ciekawostką akademicką, a coraz częściej znajduje swoje zastosowanie w rozwiązaniach komercyjnych, czego przykładami mogą być tworzone w ostatnich latach modele językowe czy samochody autonomiczne. Wzrost zainteresowania SI wśród badaczy na przestrzeni ostatniej dekady prezentuje rys.1.1.



Rys. 1.1. Liczba publikacji dotyczących sztucznej inteligencji na przestrzeni lat 2010-2021 [1]

Sama SI jest pojęciem bardzo szerokim: współczesne systemy są w stanie wspierać człowieka w wielu dziedzinach, a w części z nich zdolnością analizy przewyższają człowieka. Jak prezentuje rys.1.2, jedną z najbardziej rozwijanych w ostatnich latach ścieżek w dziedzinie sztucznej inteligencji jest wizjoner maszynowe. Polega na analizie obrazów z użyciem modeli uczenia maszynowego. Najczęściej realizują one:

- klasyfikację, czyli rozpoznawanie, co przedstawia dany obraz,
- detekcję obiektów, czyli wykrywanie i oznaczanie określonych obszarów na obrazie.



Rys. 1.2. Liczba publikacji dotyczących poszczególnych dziedzin sztucznej inteligencji na przestrzeni lat 2010-2021 [1]

Celem projektu jest pokazanie, jak widzenie komputerowe może wspierać człowieka w jego zadańach na przykładzie systemu do analizy obrazu z kamer parkingowych.

2. Zagadnienia teoretyczne

2.1. Sztuczne sieci neuronowe

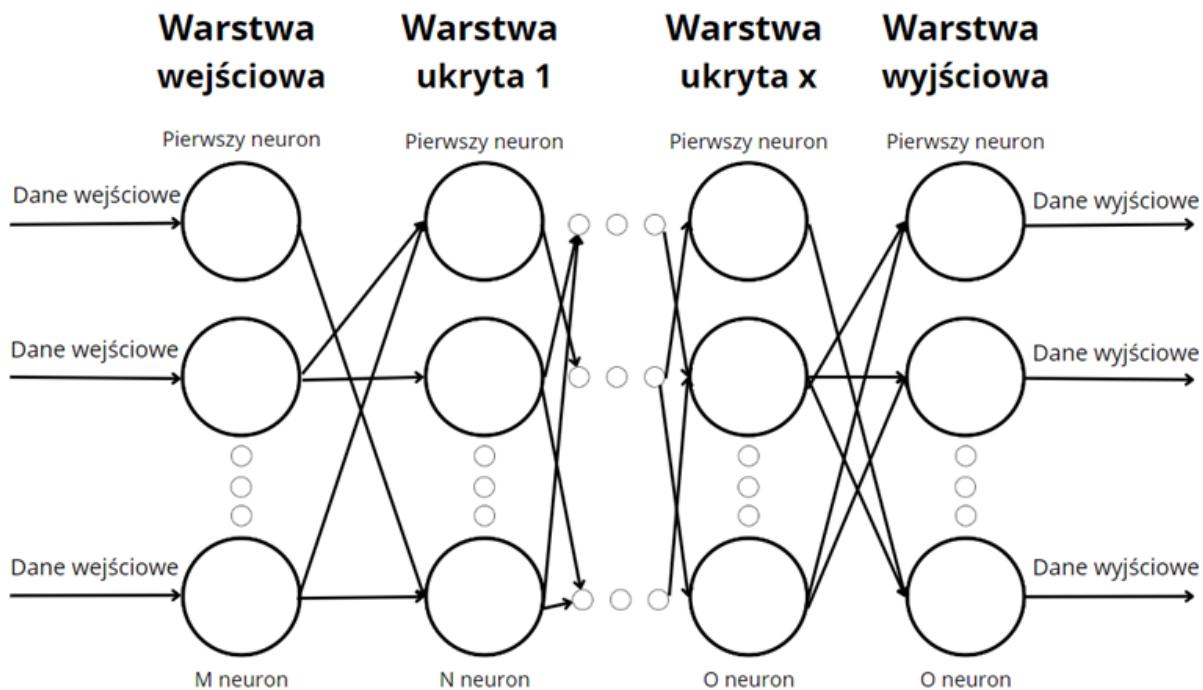
Koncept sztucznych sieci neuronowych (ang. Artificial Neural Networks, w skrócie ANN) wywodzi się ze sposobu działania ludzkiego mózgu. Podobnie zresztą jak w przypadku mózgu jego działanie opiera się o neurony. Podstawowymi zaletami sieci neuronowych, które stanowiły w ogóle podstawę dla ich rozwoju, są „ich zdolność do przetwarzania informacji w sposób równoległy, całkowicie odmienny od szeregowej pracy komputera oraz proces uczenia, zastępujący tradycyjne programowanie” [2]. Sieć ANN możemy podzielić na 3 części:

- warstwę wejściową, która przyjmuje informacje o zadanym kształcie,
- szereg warstw ukrytych, które na podstawie swoich parametrów obliczają wartość na wyjściu, którą przekazują do kolejnych warstw,
- warstwę wyjściową, w której na podstawie danych otrzymanych z poprzednich warstw podejmowana jest decyzja o wyniku wyjściowym.

Celem działania sieci ANN jest znalezienie takiego zbioru parametrów, przy którym funkcja straty (ang. *loss function*) będzie posiadała najniższą wartość. Optymalizacja najczęściej odbywa się poprzez uczenie nadzorowane (ang. *supervised learning*), co oznacza, że na wejście modelu podawane są dane z przyznaną przez człowieka etykietą (ang. *label*). Celem modelu jest optymalizacja funkcji kosztu (ang. *loss function*), co powinno prowadzić do dobierania parametrów neuronów w taki sposób, aby wyniki na warstwie wyjściowej były zgodne z etykietą. Alternatywną metodą jest uczenie nienadzorowane (ang. *unsupervised learning*), w której model otrzymuje surowe dane bez etykiet. W przypadku tego projektu wykorzystywane będzie tylko uczenie nienadzorowane. Więcej na temat historii oraz podstaw sieci neuronowych można znaleźć w książce „Sieci neuronowe” Ryszarda Tadeusiewicza z 1992 roku [2]. Współcześnie sieci neuronowe znaczco zyskały na popularności, a ich rodzina się poszerzyła, co prezentuje rys.2.2. W dodatku same sieci stały się bardziej rozbudowane: posiadają do nawet kilku tysięcy warstw ukrytych, a każda z nich może być zbudowana z olbrzymiej liczby neuronów.

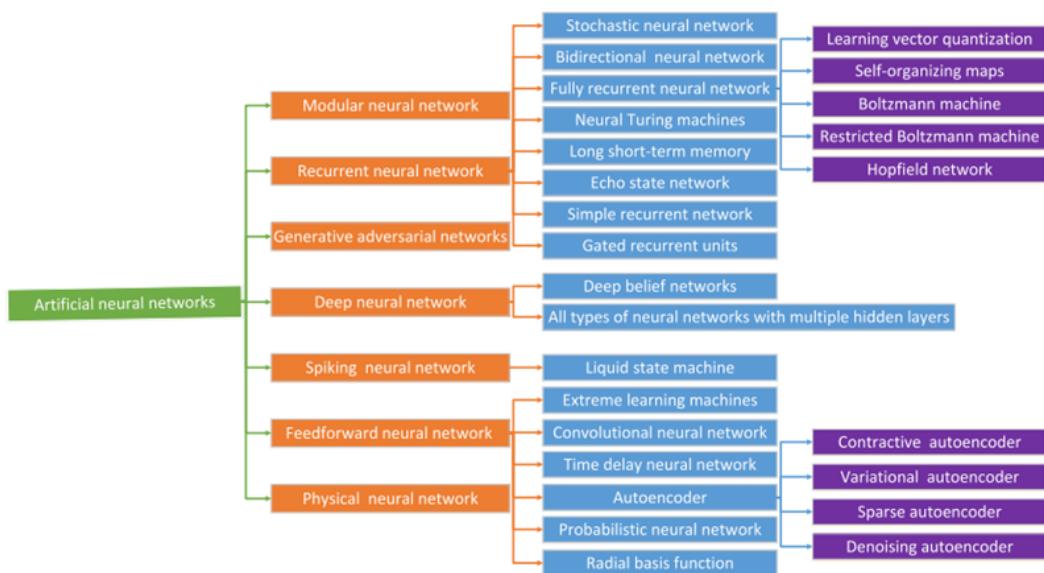
Sieci neuronowe są powszechnie używane m.in. do:

- detekcji obiektów [3],



Rys. 2.1. Architektura sieci neuronowej

- klasyfikacji obrazów [4],
- analizy efektywności/optymalizowaniu systemów [5],
- tworzeniu prognoz [6].



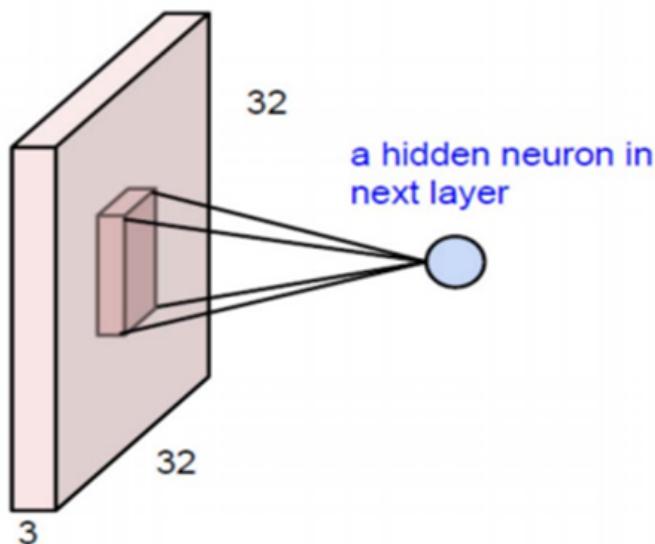
Rys. 2.2. Podział sztucznych sieci neuronowych [7]

2.2. Sieci splotowe

Sieci splotowe (ang. *Convolutional Neural Networks*, CNN) to specyficzny rodzaj sztucznej sieci neuronowej, dedykowanej analizie obrazów. W modelach sieci CNN rolę warstw ukrytych pełnią warstwy:

- konwolucyjne (ang. *convolution layers*) – każdy neuron w takiej warstwie ma na celu analizę pewnego obszaru obrazu,
- łączące (ang. *pooling layers*) – które „następują po warstwie konwolucyjnej; ich głównym zadaniem jest redukcja częstotliwości próbkowania” [7],
- w pełni połączone (ang. *fully-connected layers*, FC) – używane powszechnie także w innych rodzajach sieci ANN; znajdują się zazwyczaj na końcu modelu.

Schemat działania neuronu warstwy konwolucyjnej w takiej sieci przedstawia rys. 2.3.



Rys. 2.3. Działanie neuronu warstwy konwolucyjnej [7]

Liczba warstw konwolucyjnych i łączących jest różna w zależności od modelu, zazwyczaj jest ich od kilku do kilkudziesięciu. Warstw FC jest zazwyczaj mniej zdecydowanie mniej. Niewątpliwą zaletą tego typu sieci jest ich zdolność szybkiej analizy dużych ilości danych. Zorientowanie neuronów na poszczególne części obrazu oraz redukcje między warstwami sprawiają, że modele CNN przetwarzają obrazy szybciej niż tradycyjne sieci neuronowe, a jednocześnie są w stanie pozyskać z nich cechy niezbędne do klasyfikacji.

2.2.1. Przykładowe architektury sieci CNN

Opracowanie własnej, efektywnej architektury dla zadanego problemu jest niezwykle czasochłonne oraz wymaga znaczących zasobów obliczeniowych. Trenowanie pojedynczego modelu zajmuje zazwyczaj od kilku minut do nawet kilku godzin (choć czas ten może być znacznie dłuższy), a w celu optymalizacji należy przeprowadzić analizę wielu modeli (tj. o różnej strukturze warstw), przy różnym zestawieniu hiperparametrów. W związku z tym pospolitym zjawiskiem jest wykorzystanie gotowych architektur, które sprawdziły się już w rozwiązyaniu innych problemów. Przykładami popularnych rodzin architektur (gdyż w ramach każdej z nich możemy wyróżnić kilka modeli, zazwyczaj różniących się głębokością) sieci są:

- ResNet,
- EfficientNet
- VGG,
- Xception.

Każda z nich posiada od kilkunastu do nawet kilkuset warstw, ułożonych w różny sposób. Szczegółowe informacje na ich temat można znaleźć w źródłach [8], [9], [10], [11].

2.3. Modele detekcji obiektów

Modele detekcji stanowią kolejną odmianę sieci neuronowych. Mają one nieco inne, a właściwie szersze zastosowanie. Celem ich działania jest odnalezienie konkretnych obiektów na obrazie oraz prawidłowe ich sklasyfikowanie. „Działanie takiego systemu detekcji możemy podzielić na:

- wybranie obszarów obrazu, na którym potencjalnie znajdują się interesujące obiekty (obszary te mogą, a nawet powinny się one nakładać w sytuacji, gdy w pewnej przestrzeni obrazów znajduje się wiele obiektów),
- wyznaczenie cech poszczególnych części obrazu,
- klasyfikację” [12].

Obecnie istnieje duża liczba modeli detekcji, które, choć realizują te same zadania, różnią się potencjalnymi zastosowaniami. Najczęściej dzieli się je na:

- modele dwustopniowe (m.in. R-CNN, Fast R-CNN, Faster R-CNN) - zazwyczaj określane jako dokładniejsze, ale przy tym wolniej działające,
- modele jednostopniowe (m.in. cała rodzina modeli YOLO) – wykorzystywane w sytuacjach, gdy potrzebne jest szybkie przetwarzanie obrazu, np. w systemach czasu rzeczywistego [13].

Kwestia efektywności jest tutaj sporna i zależna od problemu, gdyż późne wersje YOLO osiągają wyniki porównywalne do sieci dwustopniowych także w zakresie czasu przetwarzania. [12]. Bardziej szczegółową analizę dostępnych modeli detekcji można znaleźć w opracowaniach [12], [13], [14].

2.4. Metryki w sieciach neuronowych

Przy testowaniu modeli wykorzystano szereg metryk. Pozwoliło to na uzyskanie dokładnych informacji dotyczących progresu w uczeniu a także na wykrycie niedoskonałości, takich jak niedouczenie (ang. *underfitting*) czy przeuczenie (ang. *overfitting*).

Macierz błędów

Macierz błędów (ang. confusion matrix) to właściwie nie metryka, a zbiór metryk, które dzięki sposobie przedstawienia są czytelniejsze. Strukturę binarnej macierzy tego typu przedstawia tabela 2.1.

Tabela 2.1. Wzór macierzy błędów dla klasyfikacji binarnej

	Wynik rzeczywisty 0	Wynik rzeczywisty 1
Wynik przewidywany 0	Wynik prawdziwy ujemny (ang. true positive, TP)	Wynik fałszywy dodatni (ang. false positive, FP)
Wynik przewidywany 1	Wynik fałszywy ujemny (ang. false negative, FN)	Wynik prawdziwy dodatni (ang. true positive, TP)

Wyniki z macierzy błędów stanowią podstawę dla szeregu innych metryk.

Dokładność

Dokładność (ang. *accuracy*) to jedna z podstawowych metryk używanych do ewaluacji. Wyraża się wzorem:

$$\frac{TN + TP}{TN + TP + FN + FP} \quad (2.1)$$

Gdzie zmienne to metryki przedstawione w tabeli 2.1. Warto jednak zaznaczyć, że może się ona okazać niemiarodajna, szczególnie w przypadku zbiorów nierównomiernie rozłożonych (tj., gdy jedna klasa jest znacznie mniej liczna niż pozostałe). Wówczas detekcja najmniej licznej klasy może być zupełnie niepoprawna, a przy tym wskaźnik może być bardzo wysoki.

Wynik funkcji straty

Wynik funkcji straty (ang. *loss*) to metryka, na podstawie której model dokonuje optymalizacji parametrów. Jej wartość wynosi między 0 a 1. „Dla przykładu przy binarnej entropii krzyżowej:

$$L = -\frac{1}{M} \sum_{m=1}^M [y_m * \log(h_0(x_m)) + (1 - y_m) * \log(1 - h_0(x_m))] \quad (2.2)$$

gdzie

M – liczba elementów treningowych,

- h_0 – to model sieci neuronowej,
- x_m – wartość wejściowa,
- y_m – to faktyczna etykieta próbki [15].

Jest to jednak miara podobnie jak dokładność niedoskonała. Może się zdarzyć, iż mimo dość niekorzystnego wyniku funkcji straty model może prawidłowo klasyfikować dane (choć nie jest to sytuacja typowa).

Precyzja

Precyzja (ang. precision) wyraża się jako:

$$\text{Precyzja} = \frac{TP}{TP + FP} \quad (2.3)$$

Gdzie zmienne są opisane zgodnie ze skrótami z tabeli 2.1.

Czułość

Czułość (ang. recall) można wyrazić wzorem:

$$Czuo = \frac{TP}{TP + FN} \quad (2.4)$$

Gdzie zmienne są opisane zgodnie ze skrótami z tabeli 2.1. Często parametr ten analizuje się razem z precyzją, gdyż ich kombinacja daje nam dość dobrą ogólną informację na temat ogólnych wyników.

Miara F1

Miara F1 (ang. F1 score) to właściwie pewne połączenie czułości oraz precyzji. Można je wyrazić wzorem:

$$F1 = \frac{2 * Recall * Precision}{Recall + Precision} \quad (2.5)$$

Często jest przyjmowane jako informacja dotycząca czułości i precyzji, ale zawarta w jednej zmiennej.

2.5. Transformacja Hough'a

Transformacja Hough'a to metoda pozwalająca znaleźć część charakterystycznych elementów (kształtów) w obrazie. Współcześnie najczęściej są to linie- chociaż sama metoda została opracowana już pod koniec lat 50., do dziś stanowi jedno z podstawowych narzędzi ich wykrywania (choćże obecnie coraz częściej w tym celu wykorzystuje się sztuczną inteligencję) [16]. Sam sposób implementacji będzie zależy od tego, jaki kształt w obrazie chcemy wykryć, natomiast najczęściej wykorzystuje się wzory reprezentując daną figurę w przestrzeni kartezjańskiej. Przykładowo: dla prostej wykorzystuje się „równanie prostej w postaci normalnej”:

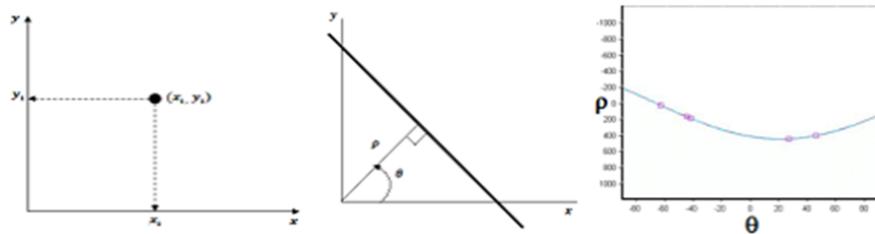
$$p = x * \cos(0) + y * \sin(0) \quad (2.6)$$

gdzie

p – liczba elementów treningowych,

θ – to model sieci neuronowej [17].

Każda para parametrów (p, θ) może stanowić reprezentację potencjalnej prostej w układzie współrzędnych.



Rys. 2.4. Transformacja od prostej w układzie kartezjańskim do krzywej w układzie (p, θ) [17]

Więcej informacji na temat transformacji Hough'a, także dla przypadków innych niż prosta, można znaleźć w źródłach [16], [17], [18].

3. Stan dziedziny, założenia, cele, architektura

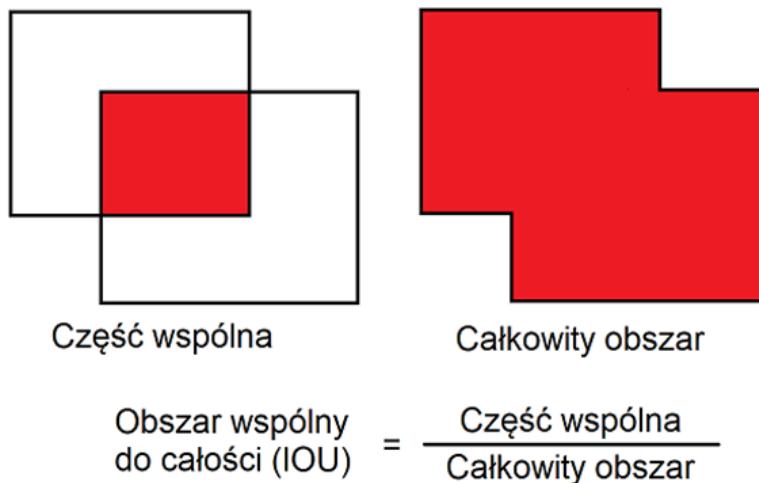
Obecnie (tj. w grudniu 2023 roku) na rynku istnieje wiele systemów służących monitorowaniu przestrzeni parkingowej. Są to zarówno rozwiązania komercyjne jak i typu otwarto źródłowego. Systemy te możemy podzielić na dwa rodzaje:

- oparte o rozbudowaną infrastrukturę fizyczną: pętle indukcyjne, sensory akustyczne, sensory podczerwieni czy sensory soniczne [19]; praktycznie zawsze wymagają one dodatkowego nakładu finansowego na ich rozbudowę oraz utrzymanie,
- pozyskujące niezbędne informacje z systemów wideo, najczęściej wykorzystujące różne typy sztucznej inteligencji.

Z drugiej grupy możemy zaś wyłonić dwa kolejne rodzaje systemów:

- wymagające specjalnego dostosowania- ustawienia kamer pod odpowiednim kątem (w większości wypadków rzut z góry); w tym wypadku potencjalny koszt dodatkowych systemów powoduje wzrost skuteczności systemu (w rozumieniu poprawności określania zajętości miejsc),
- mogące korzystać z kamer ustawionych pod różnym kątem, a zatem wykorzystujące pospolicie występującą architekturę; takie podejście jest oszczędniejsze (przynajmniej w przypadku, gdy mamy istniejący system monitoringu wideo), ale jest zazwyczaj okupione niższą efektywnością systemu (lub przynajmniej wyższą złożonością), szczególnie w przypadku parkingów rozbudowanych, w przypadku których samochód stojący na jednym miejscu parkingowym zasłania (przynajmniej częściowo) inny slot.

Same rozwiązania uczenia maszynowego są najczęściej modelami detekcji lub binarnymi klasyfikatorami. W pierwszym przypadku wykorzystuje się technikę obliczania tzw. obszaru wspólnego do całości (ang. Intersection of Union [20]), której fundamentalnym aspektem jest porównanie obszaru miejsca pośadowego (wskazanego przez administratora lub inny model uczenia) z lokalizacją pojazdu (którego położenie jest określone właśnie przez model detekcji). Miejsce jest uznawane za zajęte w przypadku, gdy stopień pokrycia jednej wartości przez drugą przekroczy pewną wartość. W drugim rozwiązaniu model wykorzystuje lokalizacje (wskażane przez administratora systemu lub inny model uczenia) slotów parkingowych na obrazie i na podstawie wskazanych obszarów podejmuje decyzję o zajętości danego slotu. Sposób obliczania IOU przedstawia rys. 3.1.



Rys. 3.1. Wizualne przedstawienie koncepcji IOU

Warto zauważyć, że wiele z projektów automatycznego monitorowania parkingu oferuje także inne usługi niż sam monitoring miejsc parkingowych, na przykład odczyt tablic rejestracyjnych czy system naliczania kosztów [21]. Zdecydowana część z nich zdaje się być rozwiązaniem dostosowującym owe systemy do potrzeb komercjalizacji.

3.1. Założenia i cele

Celem projektu było stworzenie narzędzia do efektywnego monitorowania przestrzeni parkingowej, które będzie zdolne do:

- wyznaczania miejsc parkingowych,
- wyznaczania poziomu zajętości miejsc udostępnionych do parkowania w danej chwili oraz oznaczania miejsc wolnych lub zajętych na obrazie,
- monitorowania przestrzeni parkingowej poza wyznaczonymi slotami w celu detekcji niepoprawnie ustawionych pojazdów,
- określania, czy dany pojazd jest uprzywilejowany (tj. należy do służb państwowych, takich jak Policja, Straż Pożarna, Pogotowie Ratunkowe).

System miał działać w oparciu o istniejącą infrastrukturę, tj. powinien być w stanie operować na podstawie obrazu z typowych, nieruchomych kamer monitoringu. Wymogiem było jednak to, aby z punktu widzenia obserwatora dane miejsce było dość dobrze widoczne (tzn. pojazd, który pojawi się na danym miejscu będzie widoczny w 30% lub więcej). Obraz miał być analizowany przez modele uczenia maszynowego (w szczególności modele klasyfikacji oraz detekcji), wspierane przez oprogramowanie do

przetwarzania obrazów. Miały one osiągać wysoką efektywność, nawet kosztem czasu predykcji. Zakładano minimum 95% dokładność (ang. Accuracy) w zbiorze testowym w przypadku modeli klasyfikacji, chociaż oczekiwane były wartości wyższe (97% lub więcej). Samochody poza miejscami parkingowymi miały być natomiast wykrywane z najwyższą możliwą skutecznością, przy wstępnej zakładanej dokładności na poziomie 90% w zbiorze testowym.

Przy tym system powinien działać porównywalnie dobrze niezależnie od:

- kąta ustawienia kamery (zarówno względem podłożu jak i slotu),
- modelu pojazdu,
- rodzaju podłożu (szczególnie w przypadku miejsc pustych),
- ustawienia pojazdu (zaparkowany przodem/tyłem),
- rozdzielczości obrazu,
- pogody (przynajmniej w kwestii zachmurzenia oraz umiarkowanych opadów deszczu)

W zakresie tempa przetwarzania preferowana była zdolność analizy 1 klatki nagrania monitoringu na sekundę lub więcej. W praktyce jednak dopuszczalny był system działający z niższą częstotliwością (do 1 klatki na 3 sekundy) przy założeniu, że będzie to podyktowane wzrostem poprawności uzyskiwanych wyników. Taka częstotliwość była w zupełności wystarczająca z uwagi na fakt, że zdarzenia na parkingu zachodzą w czasie znacznie dłuższym. Znacznie istotniejsze było uzyskanie jak najkorzystniejszych wyników w zakresie poprawności predykcji. Należy jednak zaznaczyć, że możliwości programu są zależne od zdolności obliczeniowych urządzenia, na którym został on uruchomiony. Powyższe oczekiwania dotyczą systemu działającego na podzespołach badawczych.

3.2. Wykorzystane narzędzia

Cały projekt był realizowany w języku Python. Jego niewątpliwą zaletą jest posiadanie szerokiego zakresu wsparcia dla uczenia maszynowego, w szczególności bibliotek: Tensorflow (oraz zawartego w niej pakietu Keras) i PyTorch. Dodatkowym atrybutem jest także dostęp do szeregu bibliotek związanych z przetwarzaniem obrazu, w szczególności OpenCV oraz Pillow. Jak możemy przeczytać na stronie producenta: „Tensorflow to otwarto źródłowa platforma dla uczenia maszynowego. Posiada wszechstronne, elastyczne środowisko z wieloma narzędziami, bibliotekami oraz zasobami wsparcia, które pozwalają badaczom rozwijać obecny poziom uczenia maszynowego oraz w łatwy sposób budować aplikacje oparte o uczenie maszynowe” [22]. Szczególnie warto zwrócić tu uwagę na pakiet wsparcia dla modeli detekcji oraz klasyfikacji obrazów, jak chociażby prosty, a zarazem efektywny system do tworzenia zbiorów danych na podstawie struktury katalogowej czy możliwość tworzenia własnych modeli na podstawie komponentów (warstw) dostępnych w ramach pakietu Keras. OpenCV oraz Pillow to natomiast biblioteki służące przede wszystkim do przetwarzania oraz analizowania obrazu. Początkowo wszystkie programy były uruchamiane na systemie operacyjnym Windows 10. W późniejszej fazie projektu pojawiła

się potrzeba zwiększenia tempa uczenia oraz predykcji modelu w celu spełnienia założeń projektowych. Taką możliwość dawało wykorzystanie pamięci graficznej zamiast procesora. Tensorflow jednak nie oferuje wsparcia GPU dla systemu Windows od wersji Tensorflow 2.10. Wymusiło to wykorzystanie WSL zamiast systemu Windows. Większość operacji została wykonana na jednostce fizycznej, wyposażonej CPU Intel(R) Core(TM) i5-10500H CPU @ 2.50GHz 2.50 GHz oraz procesor graficzny Nvidia GeForce RTX 3050 z 4GB pamięci. Dodatkowo na potrzeby optymalizacji hiperparametrów sieci CNN wykorzystane zostały zasoby chmurowe w Google Colaboratory, gdzie rolę jednostki GPU pełniła Nvidia V100 16GB. Rolę środowiska programistycznego pełnił program PyCharm. Oferuje on znaczny pakiet wsparcia z punktu widzenia programisty i przyspiesza proces tworzenia programowania. Dodatkowo początkowe testy modeli były uruchamiane poprzez wewnętrzny terminal w programie, natomiast zostało to zmienione na wywoływanie bezpośrednio poprzez konsolę PowerShell.

4. Implementacja

4.1. Proponowana architektura

Architektura modelu miała składać się z 4 elementów:

- systemu do detekcji miejsc parkingowych, który zapisuje informacje w postaci pliku .csv,
- modelu klasyfikacji określającego zajętość poszczególnych miejsc (których pozycję pozyskuje z wcześniej wspomnianego pliku),
- modelu detekcji wykrywającego pojazdy poza miejscami parkingowymi,
- modelu klasyfikacji, określający czy dany samochód nieprawidłowo zaparkowany jest pojazdem uprzywilejowanym.

4.2. Detekcja linii

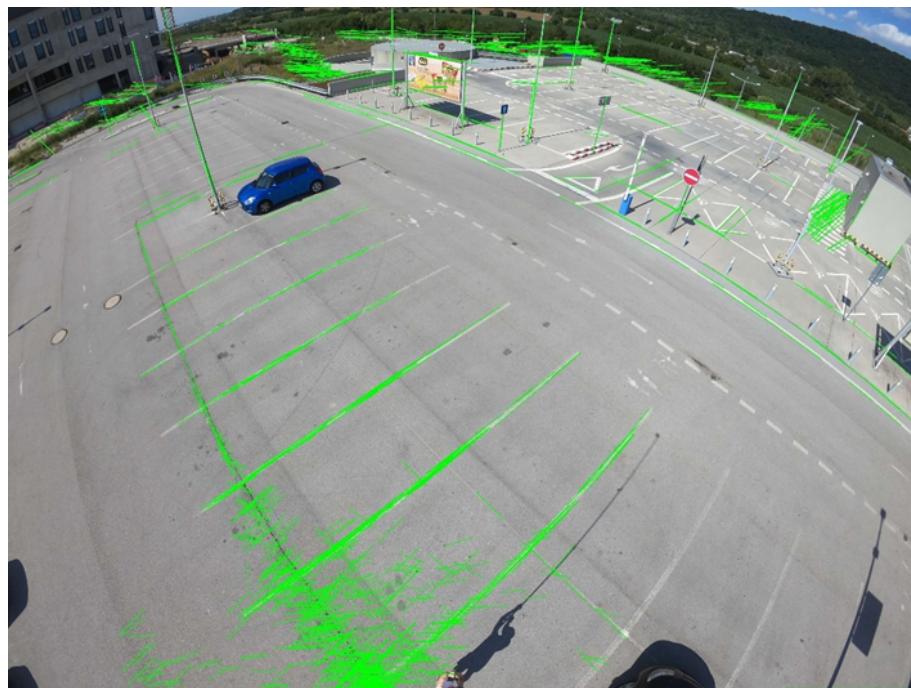
Celem detekcji linii wykorzystana została funkcja *HoughLines()* dostępna w bibliotece OpenCV, będąca implementacją Transformaty Hough'a w języku Python [23]. Okazała się ona nieefektywna w przypadku tego problemu. Chociaż wyniki częściowo pokrywały się z realną pozycją linii (w zależności od próbki), to nie były one zadowalające. Algorytm jest niestety podatny na szum oraz występowanie innych linii na obrazie (nie tylko tych oczekiwanych), a przez to nie wszystkie linie są uwzględnione lub pojawiają się w miejscach nieodpowiednich. Dodatkowym problemem był fakt, iż funkcja zwraca całe linie, a nie tylko odcinki, co utrudniało uzyskanie danych. Przykładowy wynik jej działania widzimy na rys.4.1.

Wobec fiasco owego rozwiązania przetestowana została alternatywna funkcja: *HoughLinesP()*, która także jest elementem biblioteki OpenCV i stanowi implementację probabilistycznej transformaty Hough'a, a zatem wykrywającą nie proste, a odcinki. Wyniki jednak były również niewystarczające. Efekt jej działania na obrazie testowym widzimy na rys.4.2.

Wobec niskiej efektywności powyższych algorytmów zdecydowano, że proces automatyzacji wyznaczania slotów parkingowych zostanie porzucony. Zostało to podiktowane przede wszystkim faktem, iż prawidłowa detekcja miejsc determinowała działanie wszystkich pozostałych elementów systemu, wobec czego wymogiem była jej bardzo wysoka efektywność. W związku z powyższym stworzony został



Rys. 4.1. Efekt działania funkcji HoughLines() na jednym z obrazów. Czerwony obszar to nakładające się wykryte linie



Rys. 4.2. Efekt działania funkcji HoughLinesP() na jednym z obrazów. Zielone odcinki reprezentują wykryte linie

program *LineSelector()*, w którym administrator może przekazać pozycje slotów z użyciem interfejsu graficznego. Wykorzystuje on biblioteki: matplotlib i OpenCV. Użytkownik podaje na wejściu ścieżkę

do obrazu przedstawiającego parking (musi być on w jednym z formatów obsługiwanych przez bibliotekę OpenCV) [23]. Po uruchomieniu w oknie aplikacji wyświetlony zostaje interaktywny panel z tłem w postaci podanego obrazu. Użytkownik może zaznaczyć pojedyncze miejsca, które model ma monitorować (nie muszą one być w żaden sposób oznaczone). Wprowadzone dane są następnie zapisywane w pliku o rozszerzeniu .CSV, co pozwala na wczytanie ich w przyszłości.

4.3. Model I - binarna klasyfikacja zajętości miejsc

4.3.1. Przygotowanie zbiorów: uczącego i walidacyjnego

Spośród kilku dostępnych zbiorów danych najbliższy oczekiwaniom był zbiór *Action-Camera Parking Dataset* [24]. Składał się on z 293 zdjęć parkingów. Na podstawie etykiet zostały z nich wyodrębnione obrazy przedstawiające pojedyncze pozycje parkingowe. Wszystkie pozyskane dane zostały następnie przeniesione do jednego z dwóch katalogów, oznaczonych jako '0' (w którym znalazły się miejsca wolne), oraz '1' (z miejscami zajętymi). Dane zostały następnie oczyszczone - usunięte zostały obrazy z nieprawidłowymi etykietami oraz te o skrajnym zaszumieniu (czyli takie, gdzie miejsce było w znacznym stopniu zasłonięte). Ostatecznie z danych z tego zbioru uzyskano 5146 zdjęć miejsc zajętych i 5395 zdjęć miejsc wolnych (sumarycznie: 10541).

4.3.2. Przygotowanie zbioru testowego

Aby jak najlepiej odzwierciedlić działanie prawdziwego systemu, dane na potrzeby testów zostały przygotowane ręcznie: z użyciem narzędzia administracyjnego opisanego w podrozdziale 4.1. Na obrazach pochodzących z grafiki Google, zostały wyznaczone pojedyncze miejsca postojowe [25]. Na tej podstawie powstały obrazy przedstawiające pojedyncze sloty parkingowe. Przykładowe zdjęcia zawarte w każdym ze zbiorów przedstawia rys.4.3.



Rys. 4.3. Przykładowe obrazy należące do zbiorów: 0 (miejsca wolne, po prawej) oraz 1 (miejsca zajęte, po lewej)

Łącznie uzyskano w ten sposób 1585 zdjęć miejsc zajętych oraz 1130 ujęć miejsc wolnych. Zdecydowano, że ilość ta jest nieco zbyt duża (to ponad 20% objętości zbioru), w związku z czym zbiór ograniczono do 1000 elementów na klasę, a nadmiarowe dane przeniesiono do zbioru treningowego.

Należy podkreślić, że w zbiorze testowym znajdowały się możliwie różnorodne zdjęcia- pozwoliło to na sprawdzenie, czy model rzeczywiście spełnia oczekiwania opisane w rozdziale 3.1. Sumaryczne zestawienie objętości każdego ze zbiorów danych prezentuje tabela 4.1.

Tabela 4.1. Suma zdjęć w poszczególnych zbiorach danych

	Zbiór treningowy	Zbiór walidacyjny	Zbiór testowy
Miejsca wolne (0)	4501	1125	1000
Miejsca zajęte (1)	4585	1146	1000
Suma	9086	2271	2000

4.3.3. Propozycja I – prosta sieć CNN

W celu wstępniego sprawdzenia czy zbiór treningowy dobrze wpisuje się w dane testowe (które są przykładem danych mogących pojawić się w systemie) został stworzony prosty model do klasyfikacji binarnej. Dodatkowo, z uwagi na jego prostotę oraz przewidywane niskie zużycie zasobów, rozważano jego wykorzystanie w wersji systemu dedykowanej urządzeniom o niższej mocy obliczeniowej. Model ten zaczerpnięto z dokumentacji biblioteki Keras i wykorzystuje on dostępne w niej warstwy [26]. Jego strukturę prezentuje tabela 4.2.

Tabela 4.2. Struktura modelu CNN [22]

Nazwa warstwy	Wybrane wartości
Pierwsza warstwa konwolucyjna	Filtры: 32, wymiary jądra: 3 x 3
Pierwsza warstwa łącząca	2 x 2
Druga warstwa konwolucyjna	Filtры: 64, wymiary jądra: 3 x 3
Druga warstwa łącząca	2 x 2
Trzecia warstwa konwolucyjna	Filtры: 128, wymiary jądra: 3 x 3
Trzecia warstwa łącząca	2 x 2
Pierwsza warstwa w pełni połączona	64
Druga warstwa w pełni połączona	1

W celu optymalizacji funkcji kosztu wybrano optymalizator ADAM. Jest to obecnie jeden z najpopularniejszych algorytmów tego typu. Chociaż nie musi być on optymalny, wiele badań z dziedziny przetwarzania obrazów z użyciem uczenia maszynowego wykazuje jego efektywność, a nawet wyższość na innymi rozwiązaniami, takimi jak na przykład SGD (ang. *stochastic gradient descent*) czy Adadelta [27], [28] [29].

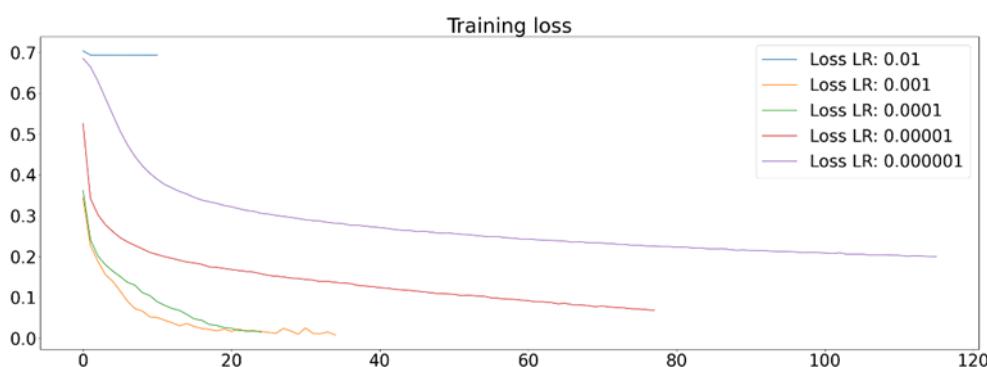
Za funkcję straty przyjęta binarną entropię krzyżową (ang. *binary cross-entropy*).

W celu ewaluacji modelu wybrano następujące metryki:

- dokładność (ang. *accuracy*),

- stratę (ang. *loss*),
- precyzyję (ang. *precision*),
- czułość (ang. *recall*),
- miara F1 (ang. *F1 score*),

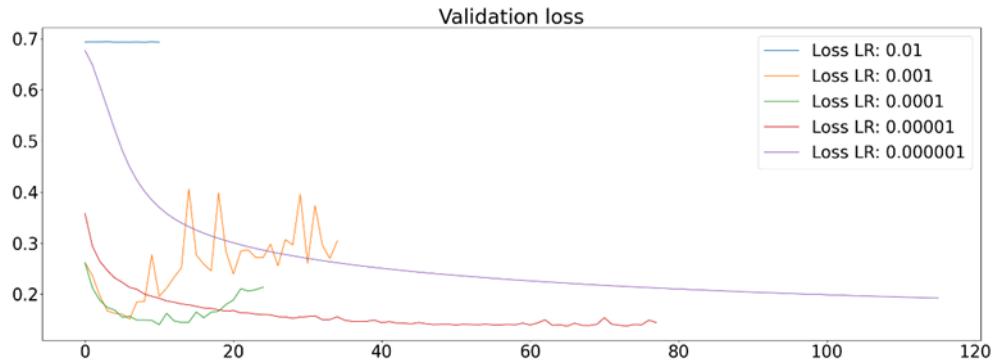
Przy czym trzy ostatnie miary były mierzone tylko dla klasy 1. Była to jednak miara wystarczająca z uwagi na bardzo podobną liczebność klas. Każda z metryk była mierzona zarówno dla zbioru treningowego, jak i walidacyjnego. Dodatkowe testy zostały przeprowadzone na zbiorze testowym. Należy tu jednak zaznaczyć, że nie zawsze wszystkie z tych metryk są niezbędne do ewaluacji modelu, wobec czego ich wyniki nie będą opublikowane w samej pracy. Będą natomiast dostępne w załączniku. W pierwszej kolejności dobrano parametr szybkości uczenia (ang. *learning rate*). Do tego celu wszystkie elementy w zbiorach: treningowym, walidacyjnym oraz testowym zostały przeskalowane do wymiarów 128 na 128 pikseli. Był to jedynie wymiar proponowany, dość standardowy dla klasyfikatorów, a jednocześnie bliski rozmiarowi większości oryginalnych zdjęć. W celu uniknięcia przeuczenia oraz skrócenia czasu uczenia zdecydowano się wykorzystać przy nauce szybkiego zatrzymania, która zatrzymuje proces uczenia w sytuacji, gdy przez 10 epok z rzędu nie nastąpi poprawa parametru dokładności binarnej. W tej fazie testów na wejście modelu było podawane 16 próbek jednocześnie (dokonano tego przez ustawienie parametru batch size). Za kluczową metrykę w tym przypadku uznano parametr straty, gdyż wskazywała ona ogólną tendencję nauki modelu. Uzyskane wyniki prezentują rys. 4.4 oraz 4.5.



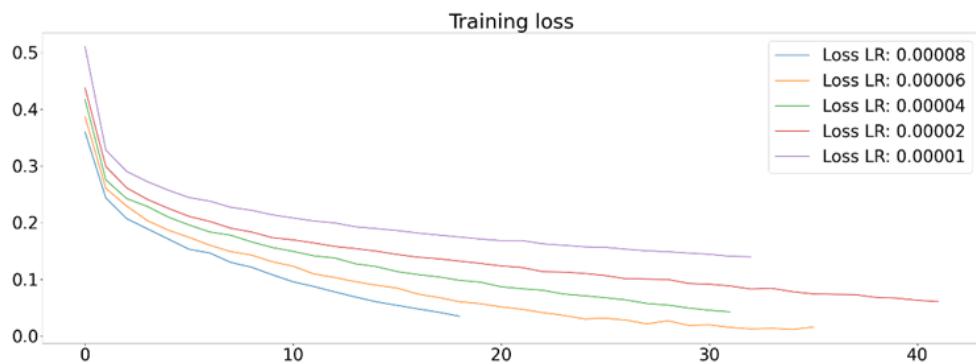
Rys. 4.4. Wykres zależności wartości funkcji straty dla zbioru treningowego w zależności od liczby epok dla różnych szybkości uczenia

Jak widać, o ile w przypadku zbioru treningowego najbardziej obiecujące wyniki uzyskano przy szybkości uczenia na poziomie 0.001 oraz 0.0001, o tyle w przypadku zbioru walidacyjnego efekty były najlepsze przy wartościach 0.0001 oraz 0.00001. Wobec tego dalsze testy prowadzono w drugim z podanych przedziałów. Ich rezultat prezentują rys. 4.6 i 4.7.

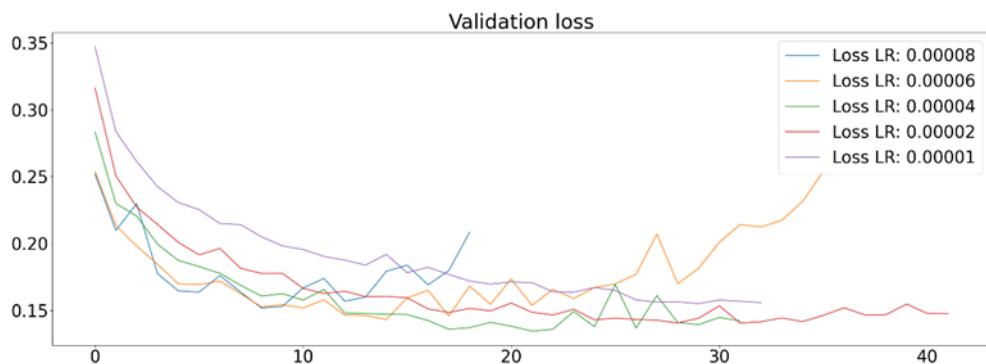
Wobec dużych skoków wartości metryk na zbiorze walidacyjnym zdecydowano się przeprowadzić jeszcze jedną próbę, w której parametr batch size został zwiększony do 32. Wyniki tych testów przedstawią rys. 4.8 i 4.9.



Rys. 4.5. Wykres zależności wartości funkcji straty dla zbioru walidacyjnego w zależności od liczby epok dla różnych szybkości uczenia

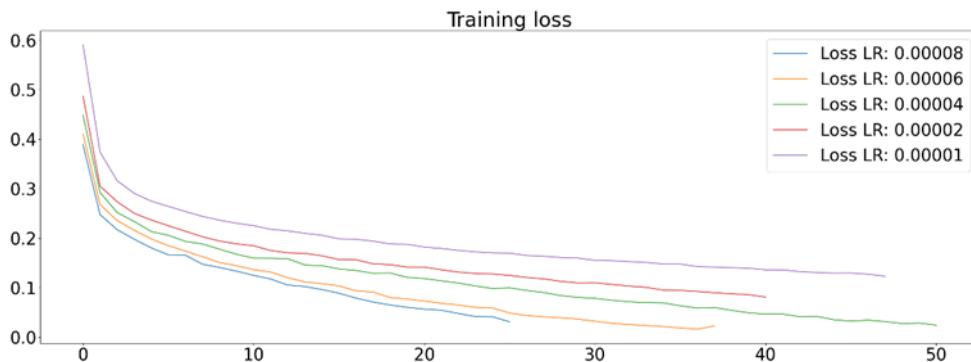


Rys. 4.6. Wykres zależności wartości funkcji straty dla zbioru treningowego w zależności od liczby epok dla różnych szybkości uczenia

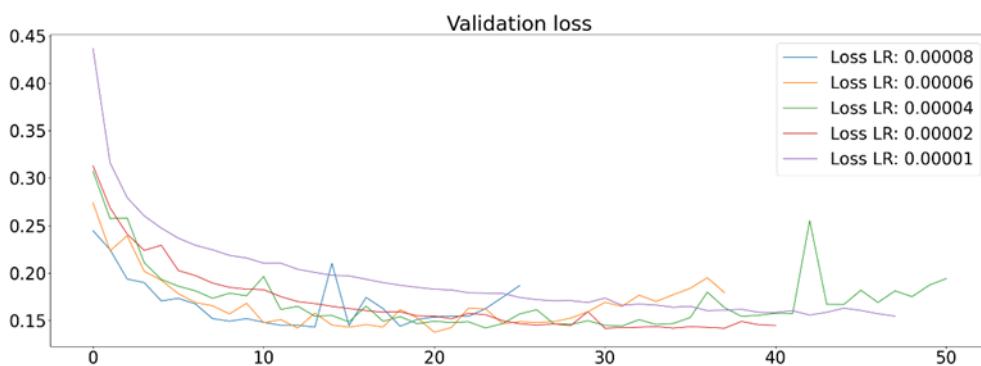


Rys. 4.7. Wykres zależności wartości funkcji straty dla zbioru walidacyjnego w zależności od liczby epok dla różnych szybkości uczenia

Na podstawie wyników zdecydowano, że pierwszą przyjętą wartością szybkości uczenia będzie 0.0002, które w razie potrzeby zostanie zredukowane. Jednocześnie wykresy wykazały, że model niezależnie od przyjętego LR ma problem z przeuczeniem, tj. wykazuje zdecydowanie lepsze wyniki dla



Rys. 4.8. Wykres zależności wartości funkcji straty dla zbioru treningowego w zależności od liczby epok dla różnych szybkości uczenia



Rys. 4.9. Wykres zależności wartości funkcji straty dla zbioru walidacyjnego w zależności od liczby epok dla różnych szybkości uczenia

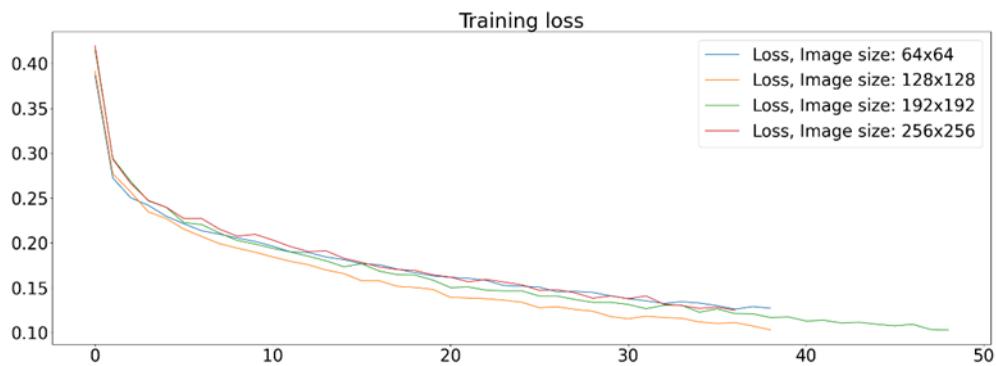
zbioru treningowego niż uczącego. Należy także zaznaczyć, że sam model, mimo bardzo niskiej liczby parametrów (2,19 miliona) okazał się dość skuteczny- osiągnął zadowalające wyniki parametrów dokładności, precyzji oraz czułości. Następnym etapem rozwoju modelu był wybór odpowiedniego rozmiaru danych wejściowych. Parametr ten był ważny nie tylko z uwagi na potencjalny wpływ na wyniki uczenia modelu, ale także czas przetwarzania. „Wyzsza rozdzielcość powinna generalnie zapewniać lepszy wynik” [30], ale przy tym zwiększać zapotrzebowanie na zasoby obliczeniowe bądź czas. Ostatecznie zdecydowano się przetestować 4 wymiar obrazów:

- 64 na 64 piksele,
- 128 na 128 pikseli,
- 192 na 192 pikseli,
- 256 na 256 pikseli.

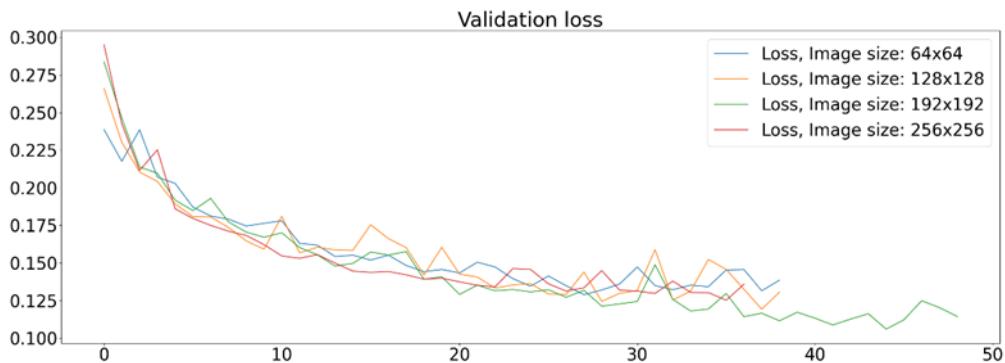
W każdym z przypadków zastosowano szybkość uczenia na poziomie 0.0002. Zdecydowano także, że z uwagi na problem przeuczenia w dotychczasowych testach, do modelu zostanie dodana warstwa augmentująca dane. Będą one polegały na losowym:

- odbiciu horyzontalnym,
- rotacji w zakresie od 0 do 11 stopni, w dowolną stronę.

Należy zaznaczyć, że zmiany te nie wpływały na sam rozmiar obrazów wejściowych. Uzyskane wyniki prezentują rys. 4.10 i 4.11.



Rys. 4.10. Wynik funkcji straty modelu na zbiorze treningowym w zależności od liczby epok dla różnych rozmiarów wartości wejściowych



Rys. 4.11. Wykres funkcji straty modelu na zbiorze walidacyjnym w zależności od liczby epok dla różnych rozmiarów wartości wejściowych

Wyniki wszystkich modeli w zakresie funkcji straty oraz dokładności, niezależnie od przyjętych wymiarów wejściowych, są do siebie bardzo zbliżone. Najlepsze efekty osiągnięto przy rozdzielczości obrazów na poziomie 192 na 192 piksele (w skrócie *px*). Funkcja straty utrzymuje się na wysokim poziomie, natomiast nie jest ona wysoce szkodliwa z punktu widzenia efektywności, co prezentuje tabela 4.3.

Tabela 4.3. Wyniki działania modelu w zależności od rozmiaru obrazu wejściowego

	64 na 64 px	128 na 128 px	192 na 192 px	256 na 256 px
Dokładność	94,5%	93,5%	95,1%	91,55%
Czułość	94%	93,1%	94,1%	88,2%
Precyza	95,1%	95%	96%	95,8%
Wynik F1	94,5%	94%	95%	91,8%
Liczba parametrów	617665	2190529	4811969	8481985
Średni czas przewidywania (GPU)	13 ms	13 ms	14 ms	16 ms
Średni czas przewidywania (CPU)	16 ms	21 ms	31 ms	41 ms

Niepokojący dla dalszego rozwoju był brak postępów w zakresie dokładności po przekroczeniu wartości 95%. Było to podyktowane najpewniej faktem, iż tak prosty model nie jest w stanie dobrze wpasować się w dane, które są dość różnorodne w obrębie klasy. Statystyki precyzyji oraz czułości sugerowały, że dane w obydwu zbiorach są klasyfikowane w podobnym stopniu poprawnie, z niewielką korzyścią na rzecz zbioru przedstawiającego miejsca zajęte; z uwagi na fakt, że na części zdjęć miejsca puste są zasłonięte przez samochody, było to spodziewane. Jednocześnie należy zwrócić uwagę na fakt, iż najprostszy model (a przez to zdolny do relatywnie szybkiego przetwarzania nawet przy ograniczonych zasobach), mający znacznie mniej parametrów niż większość typowych, obecnie wykorzystywanych architektur i tak osiągnął dokładność na poziomie 94,5%, a zatem bardzo bliską minimalnej wartości oczekiwanej. Wobec tego zdecydowano się dokonać jego optymalizacji, w celu potencjalnego wykorzystania w systemie dedykowanym urządzeniom o niskiej mocy obliczeniowej.

Optymalizacji dokonano z użyciem funkcji *Tuner()*, dostępnej w bibliotece Keras [26]. Pozwala ona na wybór zbiorów hiperparametrów, których kombinacje są następnie automatycznie testowane. Celem optymalizatora jest znalezienie modelu, który będzie najlepiej spełniał wskazane przez użytkownika kryterium. W tym przypadku będzie to metryka binarnej dokładności w zbiorze walidacyjnym. Taki wybór zapewnia, optymalizator nie będzie faworyzował modeli przeuczonych, a przy porównywalnym rozmiarze zbiorów oraz oczekiwany wartościom dokładności powyżej 95% nie ma właściwie szans na dyskryminację jednej z klas; dodatkowo problemy w obrębie jednej klasy będzie wskazywała metryka precyzyji lub czułości.

Z uwagi na duże wymagania obliczeniowe zdecydowano, że testy zostaną przeprowadzone w środowisku chmurowym Google Collaboratory, a testom zostanie poddana tylko część wartości, w stosunkowo małym zakresie. Szczegółowe parametry testowe z zachowaniem kolejności warstw prezentuje tabela 4.4.

Poza wyżej wymienionymi parametrami należy dodać, że:

- szybkość uczenia mogła przyjąć jedną z wartości: 0.00004, 0.00002, 0.00001, 0.000008,

Tabela 4.4. Zaproponowane wartości hiperparametrów

Warstwa	Minimalna wartość lub wielkość filtra	Krok	Maksymalna wartość lub wielkość filtra	Funkcja
Pierwsza warstwa konwolucyjna	16	16	64	ReLU
Pierwsza warstwa łącząca	2 x 2	—	2 x 2	—
Pierwsza warstwa porzucenia	0,0	0,1	0,2	ReLU
Druga warstwa konwolucyjna	32	32	256	ReLU
Druga warstwa łącząca	2 x 2	—	2 x 2	—
Druga warstwa porzucenia	0,0	0,1	0,2	—
Trzecia warstwa konwolucyjna	32	32	256	ReLU
Trzecia warstwa łącząca	2 x 2	—	2 x 2	—
Trzecia warstwa porzucenia	0,0	0,1	0,2	—
Pierwsza warstwa w pełni połączona	32	32	256	ReLU
Druga warstwa w pełni połączona	1	1	1	Sigmoid

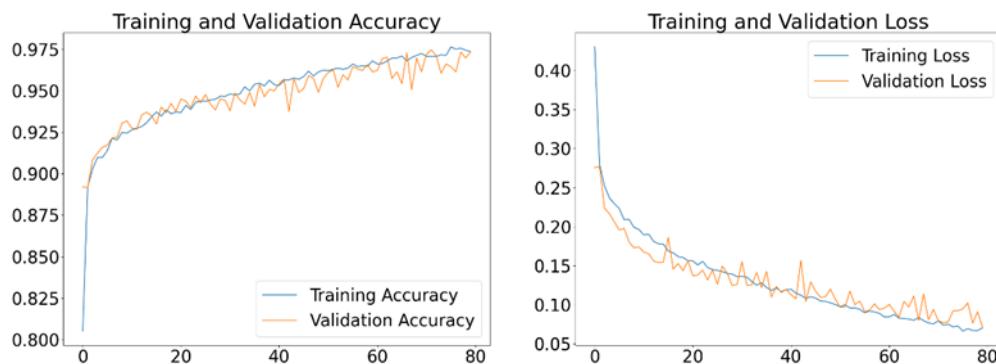
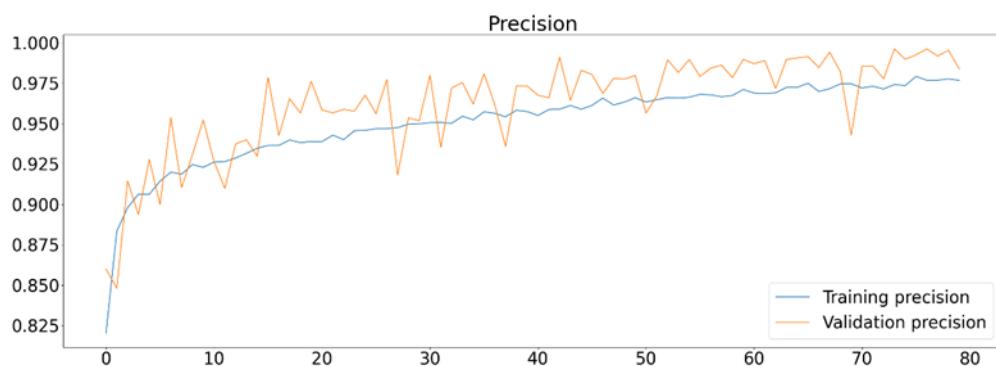
- rozmiary jąder w przypadku warstw konwolucyjnych mogły przyjmować wartości: (3,3), (4,4), (5,5) (6,6),
- maksymalna liczba epok została wyznaczona na 128, ale została włączona funkcja szybkiego zatrzymania, która sprawiała, że trening był przerywany w przypadku braku progresu w wartości binarnej dokładności na przestrzeni ostatnich 8 epok.

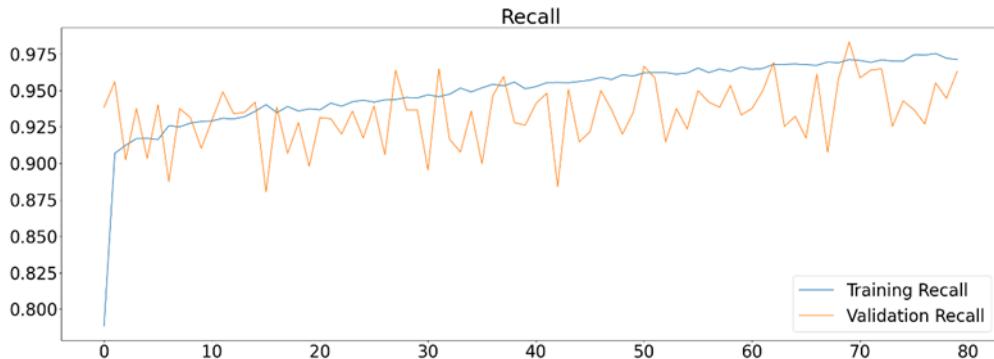
Wynikiem działania optymalizatora było otrzymanie wartości hiperparametrów przedstawionych w tabeli 4.5.

Uzyskany model został ponownie wytrenowany, już na standardowym urządzeniu, oraz poddany testom. Wyniki trenowania prezentują rys. 4.12 – 4.15.

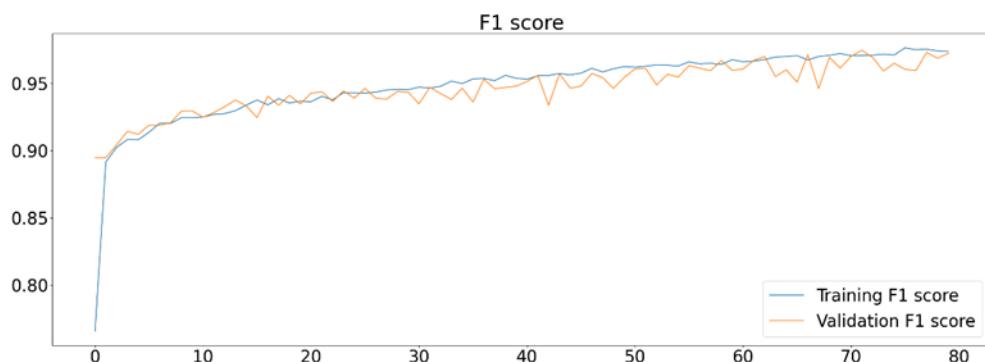
Tabela 4.5. Hiperparametry modelu uzyskane w procesie optymalizacji

Nazwa warstwy	Wybrane wartości
Pierwsza warstwa konwolucyjna	Filtry: 32, wymiary jądra: 6 x 6
Pierwsza warstwa łącząca	2 x 2
Pierwsza warstwa porzucenia	0,1
Druga warstwa konwolucyjna	Filtry: 192, wymiary jądra: 6 x 6
Druga warstwa łącząca	2 x 2
Druga warstwa porzucenia	0,2
Trzecia warstwa konwolucyjna	Filtry: 256, wymiary jądra: 6 x 6
Trzecia warstwa łącząca	2 x 2
Trzecia warstwa porzucenia	0,1
Pierwsza warstwa w pełni połączona	224
Druga warstwa w pełni połączona	1
Szybkość uczenia	0,00002

**Rys. 4.12.** Wykresy dokładności i wyniku funkcji straty w zależności od liczby epok**Rys. 4.13.** Wykres zależności precyzji od numeru epoki



Rys. 4.14. Wykres zależności czułości od numeru epoki



Rys. 4.15. Wykres zależności wyniku F1 od numeru epoki

Ponadto działanie modelu zostało sprawdzone na zbiorze testowym. Rezultaty zostały przedstawione w tabeli 4.6.

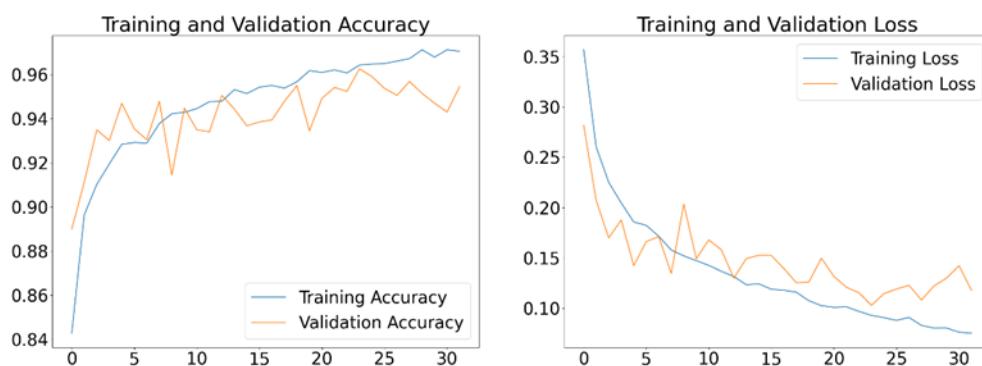
Tabela 4.6. Wartości metryk uzyskane przez zoptymalizowany zbiór

Dokładność	97,3%
Czułość	96,31%
Precyzja	98,38%
Wynik F1	97,22%
Liczba parametrów	5665057
Średni czas przewidywania (GPU)	16 ms
Średni czas przewidywania (CPU)	30 ms

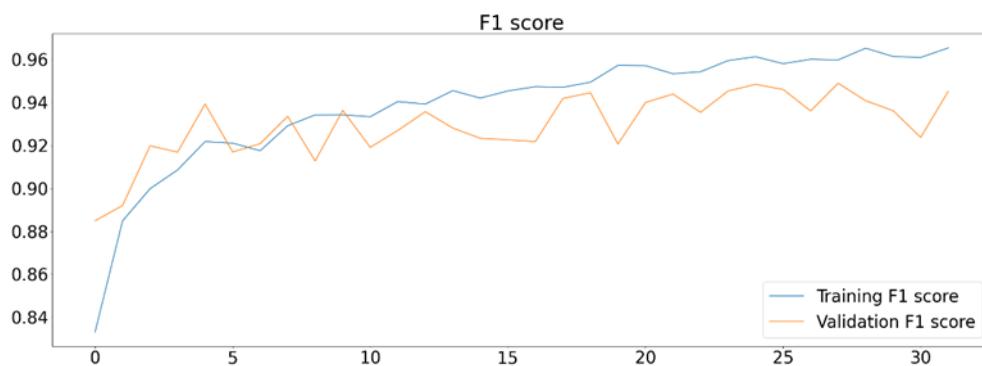
Wyniki tego modelu okazały się wręcz rewelacyjne względem oczekiwania. Mimo niezwykle prostej struktury modelu, był on bardzo bliski spełnienia kryteriów dla systemu przeznaczonego na urządzenia o wysokiej mocy obliczeniowej. Należy przy tym zaznaczyć.

4.3.4. Propozycja II – model oparty o architekturę AlexNet

Kolejnym testowanym modelem był AlexNet. Podobnie jak wcześniej wspomniany EfficientNet jest zazwyczaj używany do klasyfikacji wieloklasowej, natomiast istnieje możliwość jego implementacji jako klasyfikatora binarnego. Model zaimplementowano w języku Python z użyciem gotowych warstw, zgodnie z ogólnie przyjętym schematem tej sieci; składała się ona z 18 warstw, w tym 4 warstw konwolucyjnych. Na potrzeby modelu przeskalowano elementy we wszystkich zbiorach danych do wartości 224 na 224 pikseli. Jedyną zmianą względem standardowego modelu jest zmiana w ostatniej warstwie, w której ograniczono liczbę filtrów i zmieniono funkcję aktywacji na sigmoid. Przeprowadzono szereg testów, z różnymi wartościami szybkości uczenia oraz parametru batch size. Najlepsze rezultaty (osiagnięte przy szybkości uczenia równej 0.00002 oraz wartości batch size równej 32) treningu oraz walidacji modelu prezentują rys. 4.16 oraz 4.17.



Rys. 4.16. Wykresy dokładności i wyniku funkcji straty w zależności od liczby epok

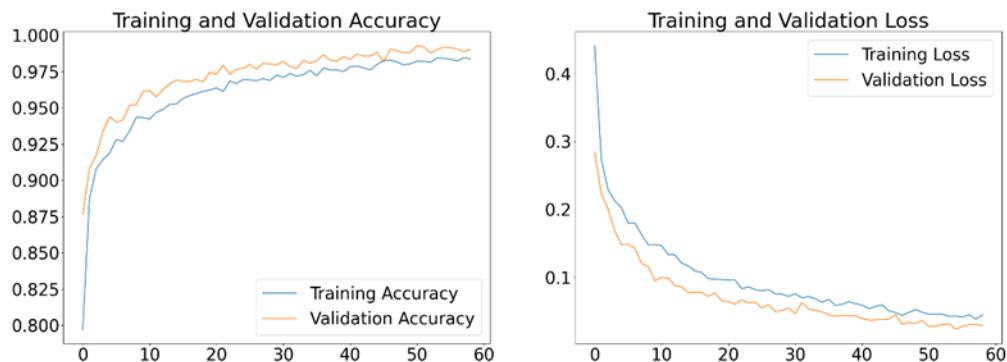


Rys. 4.17. Wykres zależności wyniku F1 od numeru epoki

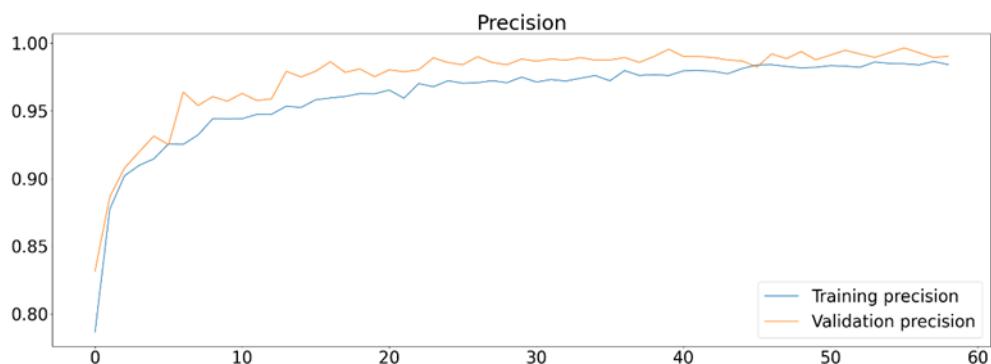
Niestety, ale architektura AlexNet nie sprawdziła się w zadanym problemie. Jej wyniki, nieco wbrew oczekiwaniom, są bardzo zbliżone do tych, osiąganych przez najprostszą sieć CNN. W związku ze słabymi wynikami na zbiorze walidacyjnym zdecydowano się nie sprawdzać wyników dla zbioru testowego.

4.3.5. Propozycja III – model oparty o architekturę EfficientNet

Spośród kilku możliwych implementacji architektur typu EfficientNet w bibliotece Keras zdecydujono, że sprawdzone zostanie działanie modelu EfficientnetV2B0, z uwagi na jego korzystny stosunek ilości parametrów do deklarowanych osiągów [26]. Na wejściu model ten przyjmuje obrazy o wymiarach 224 na 224 piksele, więc w celu treningu i ewaluacji wykorzystano te same zbiory danych co w przypadku sieci AlexNext. W ogólnym przypadku jest to model klasyfikacji multiklasowej, który składa się z Aby dostosować ten model do potrzeb klasyfikacji binarnej, dodano na jego końcu warstwę sieci gęstej z funkcją aktywacji sigmoid. W celu uzyskania możliwie najlepszych wyników przeprowadzono testy z różną wartością parametru szybkości uczenia, który ostatecznie został ustawiony na wartość 0.00005. Z uwagi na odnotowany w trakcie treningu problem przeuczenia zdecydowano, że dane przed wprowadzeniem do właściwego modelu będą augmentowane przez dodatkową warstwę. Ostateczne rezultaty treningu przedstawiają rys. 4.18 - 4.21.

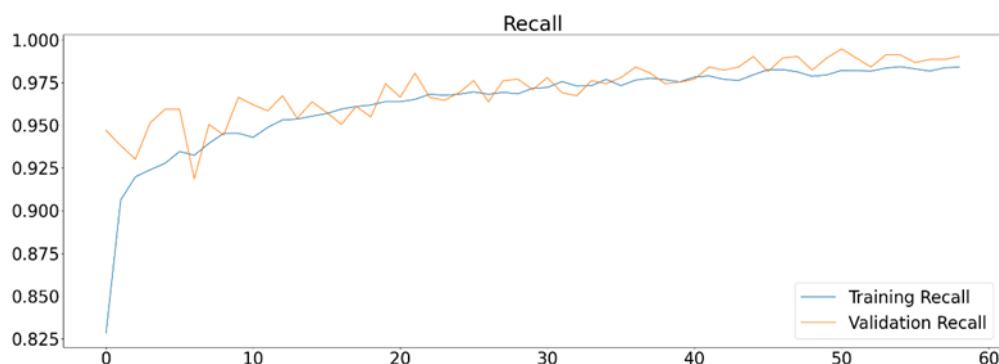


Rys. 4.18. Wykresy dokładności oraz wartości funkcji straty w zależności od epok

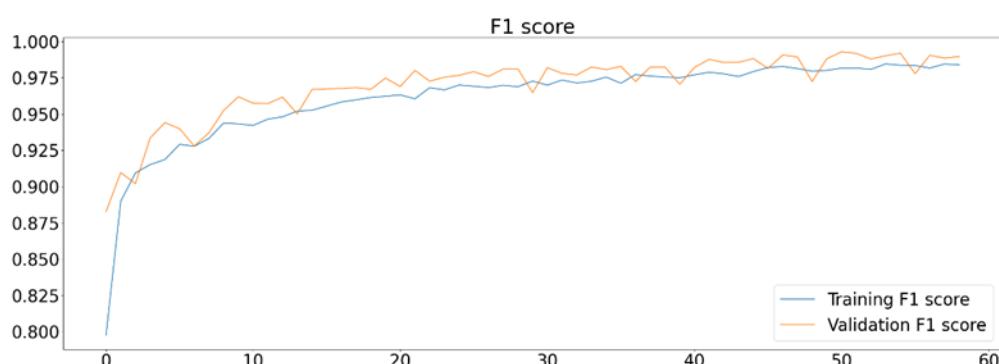


Rys. 4.19. Wykres precyzji w zależności od liczby epok

Wykresy pokazują, że w procesie treningu sieć EfficientNet osiągnęła znacznie lepsze wyniki niż wszystkie wcześniejsze modele. Zdecydowano, że na tym etapie pracy nad modelem można zakończyć. Wszelkie próby poprawy wyników poprzez korekty szybkości uczenia czy augmentacje nie przyniosły



Rys. 4.20. Wykres czułości w zależności od epoki



Rys. 4.21. Wykres wyniku F1 w zależności od epoki

polepszenia rezultatów. W dodatku dalszy progres wymagałby dużej ilości testów, a nic nie sugeruje, aby model był w stanie znacząco przekroczyć próg 99% na zbiorze walidacyjnym. Jedynymi dość nietypowymi, ale niezbyt szkodliwymi wadami modelu są:

- lepsze wartości metryk na zbiorze walidacyjnym- może być to spowodowane augmentacją danych w zbiorze treningowym oraz wykorzystaniem warstw Dropout,
- skoki wartości metryk w zbiorze walidacyjnym- te natomiast wywołuje najpewniej nieidealna generalizacja danych, która najpewniej jest spowodowana niewystarczającą ilością danych w zbiorze treningowym.

Ostateczne wyniki na zbiorze testowym prezentuje tabela 4.7.

Tabela 4.7. Wyniki uzyskane przez model EfficientNetV2B0 na zbiorze testowym

Dokładność	98,9%
Czułość	99,2%
Precyzja	98,14%
Wynik F1	97,22%
Liczba parametrów	5859985
Średni czas przewidywania (GPU)	16 ms
Średni czas przewidywania (CPU)	65 ms

4.3.6. Porównanie wyników

Wyniki poszczególnych użytecznych modeli klasyfikacji znajdują się w tabeli 4.8. Uwzględniono w nich tylko te modele, które uznano za potencjalnie użyteczne.

Tabela 4.8. Wyniki poszczególnych modeli klasyfikacji miejsc

	Prosta sieć CNN 64 na 64 px	Zoptymalizowana sieć CNN, 64 na 64 px	Sieć EfficientNetV2B0
Dokładność	94,5%	97,3 %	98,45%
Precyzja	95,1%	96,29%	98%
Czułość	94%	98,4%	98,9%
Miara F1	94,5%	97,22%	98,44%
Liczba parametrów	617665	5665057	5859985
Czas przetwarzania na GPU	13 ms	16 ms	25 ms
Czas przetwarzania na CPU	16 ms	30 ms	nie dotyczy

4.4. Detekcja pojazdów poza miejscami parkingowymi

Drugim elementem systemu miał być model służący detekcji pojazdów poza miejscami wyznaczonymi. Spośród wielu możliwych implementacji zdecydowano się wykorzystać sieć YOLO. Decyzja ta była podyktowana przede wszystkim faktem, iż sam system oceny zajętości miejsc wymagał znacznych zasobów obliczeniowych, wobec czego model detekcji musiał działać w możliwie jak najkrótszym czasie. Jednocześnie należy zauważyć, iż w badaniach dotyczących detekcji pojazdów model YOLOv5 uzyskał nie tylko rewelacyjne wyniki czasowe, ale także nie ustępował sieci R-CNN czy SSD pod względem efektywności [31]. W przypadku tego zagadnienia zrezygnowano z trenowania własnego modelu.

Z uwagi na duże zainteresowanie modelami detekcji pojazdów w dziedzinie samochodów autonomicznych, dla większości zbiorów istnieją już gotowe, wytrenowane modele detekcji. Samodzielne uczenie modelu byłoby w tej sytuacji czasochłonne, a uzyskane efekty byłyby właściwie powtórzeniem cudzych badań. Zdecydowano się wobec tego na wykorzystanie modelu YOLOv5 wytrenowanego na zbiorze COCO. Został on udostępniony na stronie biblioteki Keras. Aby ograniczyć wykrywanie tylko do określonych obszarów, wykorzystano zmodyfikowany system opisany w rozdziale 4.2, w którym administrator mógł zaznaczyć miejsca wymagające monitorowania. Obszary te może mieć dowolny kształt. Model skanuje obraz, a jeśli jeden z pojazdów w 50% lub więcej pokrywa się ze wskazanym obszarem, zostaje uznany za nieprawidłowo zaparkowany. W celu ewaluacji jakości modelu oraz ustalenia odpowiedniej wartości parametru threshold przeprowadzono proste testy. Na kilku zdjęciach przedstawiających całe parkingi, zebranych na potrzeby zbioru testowego w dziale 4.5 oznaczono 207 pozycji samochodów. Jeśli IOU dla miejsc wskazanych ręcznie oraz tych wyznaczonych przez model wynosił 50% lub więcej, uznawano, że model prawidłowo wskazał pojazd. Spośród 207 pojazdów model wskazał prawidłowo 169, przy średnim czasie detekcji (dla całego, pojedynczego zdjęcia) na poziomie 670 ms z użyciem CPU i 89 ms przy wykorzystaniu GPU. Nie jest to wynik wysoce satysfakcjonujący, natomiast należy zwrócić uwagę na kilka istotnych aspektów:

- wyniki negatywne często występuły w przypadku bardzo niskiej rozdzielczości zdjęć, gdzie przecienny samochód zajmował jedynie obszar kilkuset pikseli,
- w niektórych wypadkach model wykrył pojazdy nie uwzględnione w teście, na przykład w znaczącym stopniu założone- jest to oczywiście zaleta modelu,
- model bardzo słabo sprawdził się w przypadku, gdy zdjęcie parkingu było wykonane z góry.

Z uwagi na brak potencjalnych alternatyw zdecydowano, że model zostanie wykorzystany w systemie.

4.5. Trzeci model – klasyfikacja pojazdów poza miejscami parkingowymi

4.5.1. Przygotowanie zbiorów danych

W przypadku ostatniego z modeli znaleziono tylko jeden zbiór zdjęć przedstawiających pojazdy polskich służb [32]. Zawierał on łącznie 470 obrazów. Niestety, większość z nich została wykonana z poziomu podłoża- a perspektywa typowej kamery ukazuje pojazdy pod pewnym kątem z góry. Wobec braku możliwości pozyskania dobrego zbioru, zdecydowano się dokonać pewnej improwizacji. W pierwszej kolejności zdecydowano się zebrać dodatkowe zdjęcia z grafiki Google. Łącznie uzyskano ich 81. Aby jak najlepiej odzwierciedlić faktyczne warunki podczas testów, większość zdjęć pojazdów uprzejmie wykonywanych wykonanych z góry lub pod kątem włączono od razu do zbioru testowego- łącznie było

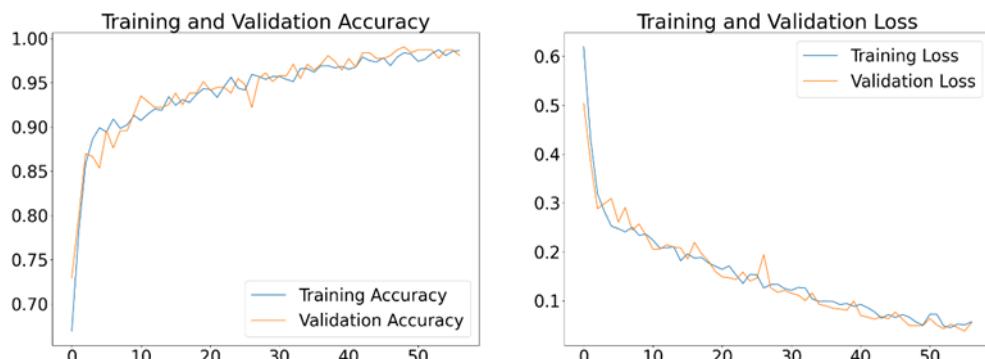
ich 50. Jest to mała próbka, ale najlepiej odzwierciedlała obrazy, które model będzie potencjalnie analizował. Zdjęcia przedstawiające pojazdy cywilne zostały natomiast pozyskane ze zbioru opisanego w podrozdziale 4.3.1. Ostateczny rozkład danych w zbiorach przedstawia tabela 4.9.

Tabela 4.9. Liczba zdjęć poszczególnych klas dla każdego ze zbioru

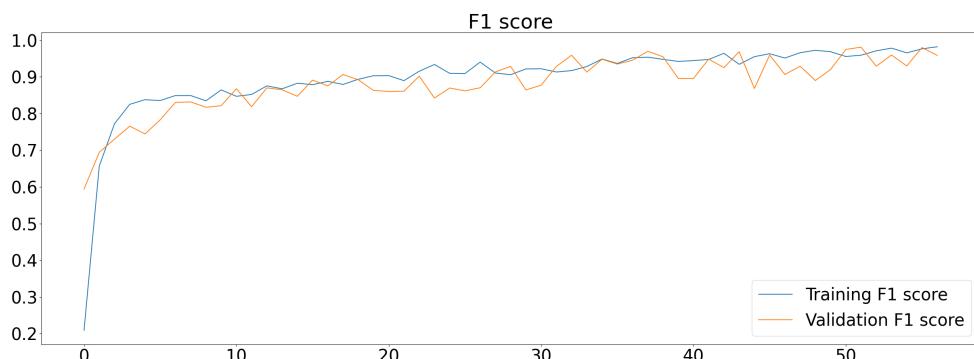
	Zbiór treningowy	Zbiór walidacyjny	Zbiór testowy
Pojazdy cywilne (0)	806	202	102
Pojazdy służb (1)	422	129	51
Suma	1228	331	153

Z samego zbioru zostały stworzone 3 katalogi, każdy o innym rozmiarze zdjęć; odpowiednio: 224 na 224 piksele, 299 na 299 pikseli i 331 na 331 pikseli. Miało to na celu dostosować dane do wymogów narzuconych przez proponowane architektury.

W celu wstępnych testów ponownie stworzono sieć, taką samą jak w rozdziale 4.3.1. Wyniki jej uczenia prezentują rys. 4.22 i 4.23.



Rys. 4.22. Wykres zależności dokładności oraz wyniku funkcji straty od epok



Rys. 4.23. Wykres zależności wartości metryki F1 od numeru epoki

Jej wyniki okazały się (wbrew oczekiwaniom) całkiem zadowalające- ponownie prosta sieć CNN okazała się potencjalnie użyteczna.

Oprócz niej do trenowania wybrano także 4 modele, których implementacje znajdują się w bibliotece Keras:

- EfficientNetV2B0,
- EfficientNetV2B3 (czyli właściwie głębszą wersję powyższej sieci),
- NasNetLarge,
- Xception,
- InceptionResNetV2.

Wszystkie wspomniane modele to sieci głębokie, mające co najmniej kilkadziesiąt warstw. Wobec tego zdecydowano się nie przytaczać architektury każdej z nich w niniejszym dokumencie. Ich szczegółową charakterystykę można znaleźć w źródłach [8][9], [10], [11].

Niestety, sieci NasNetLarge i InceptionResNetV2 zostały niemal natychmiast odrzucone- ich duża liczba parametrów sprawiła, że niemożliwe było ich trenowanie przy dostępnych zasobach. Wyniki pozostałych modeli pokazuje tabela 4.10.

Tabela 4.10. Wyniki poszczególnych modeli klasyfikacji miejsc

	Prosta sieć CNN	EfficientNetV2B0	EfficientNetV2B3	Xception
Batch size	16	16	16	8
Szybkość uczenia	0,00002	0,00002	0,00002	0,00003
Dokładność na zbiorze walidacyjnym	97,65%	96,5 %	98,2%	98,7%
Dokładność na zbiorze testowym	83,7%	81,7%	89,5%	87,6%
Precyzja	95,1%	96,29%	100%	100%
Czułość	56,9%	47%	69%	62,7%
Miara F1	72%	63,1%	81,7%	77,1%
Czas przetwarzania na GPU	13 ms	25 ms	33 ms	24 ms
Czas przetwarzania na CPU	140 ms	75 ms	120 ms	150 ms

Jak pokazuje tabela, nie udało się właściwie wytrenować choćby jednego modelu, który osiągnąłby zadowalającą skuteczność. Chociaż parametry dokładności dla zbioru walidacyjnego są dość optymistyczne, żaden z modeli nie wykrył więcej niż 80% pojazdów uprzewilejowanych. Niestety, ale sytuacja

ta wynika przede wszystkim z braku odpowiedniego zbioru danych- liczba zdjęć jest kilkukrotnie mniejsza niż oczekiwane minimum. Warto jednak zauważyc, że precyza w każdym wypadku była bardzo bliska 100% - a zatem model nie zwalnia z poprawnego parkowania kierowców normalnych pojazdów, a jedynie niepoprawnie klasyfikuje pojazdy służb jako cywilne; model jest wobec tego nadal użyteczny. Jednocześnie ulepszenie zbioru okazało się niemożliwe, wobec czego na tym etapie prace nad tymi modyfikacjami zakończono.

4.6. Pełna implementacja

Sumaryczne zestawienie modeli, które osiągnęły zadowalające wyniki w poszczególnych zadaniach, przedstawia tabela.

Zgodnie z założeniami zdecydowano się stworzyć dwie wersje systemu, dedykowane odpowiednio systemom o małej mocy obliczeniowej (w szczególności posiadającym tylko CPU) oraz o dużym dostępie do zasobów (pracujących na GPU, którego parametry są porównywalne lub lepsze od tych wspomnianych w podrozdziale 3.3.). Należy jednak zaznaczyć, że z uwagi na brak rozwiązań alternatywnych w obydwu systemach zostaną użyte: ten sam system do wskazywania miejsc (opisany podrozdziale 4.2) oraz model do detekcji pojazdów poza miejscami wyznaczonymi (zaprezentowany w podrozdziale 4.5).

4.6.1. Wersja I - bazująca na GPU

W przypadku GPU dopuszczono możliwość zastosowania każdego modelu, bez względu na jego czas przetwarzania- nie był on w tym wypadku krytyczny. Wobec tego kierowano się przede wszystkim wynikami modelu osiągniętymi na zbiorze testowym. W przypadku klasyfikacji miejsc zdecydowano, że użyty zostanie model EfficientNetV2B0 wytrenowany w podrozdziale 4.3.5. Osiągnął on ponad 98% dokładności na zbiorze testowym, dzięki czemu spełnił wymagania jakościowe. Jednocześnie czas predykcji dla jednego miejsca wynosił do 30 ms, co sprawia, że wymagania w kwestii ilości przetwarzanych klatek także zostanie spełnione- przynajmniej w przypadku małych i średnich parkingów (tj. mających do około 20 miejsc, na których nie występuje zbyt pospolite zjawisko niepoprawnego parkowania). W kwestii klasyfikacji pojazdów ze względu na uprzywilejowanie wybrany został model EfficientNetV2B3, gdyż osiągnął najlepsze wyniki, a czas jego pracy nie powinien mieć znaczącego wpływu na tempo działania całego systemu.

4.6.2. Wersja II - działająca na CPU

Wszystkie wytrenowane w ramach projektu modele działają znacznie wolniej, gdy korzystają z CPU zamiast GPU. Wobec tego w przypadku tej wersji systemu ważne było zachowanie jak najlepszego balansu między dostępną jakością a wymogami sprzętowymi. Wobec tego na potrzeby drugiego systemu

wybrano podstawowy model omówiony w podrozdziale 4.3.3 oraz model EfficientNetV2B3 w celu klasyfikacji pojazdów na uprzywilejowane i cywilne. Mimo skrajnej prostoty, osiągnęły one zupełnie akceptowalne wyniki. W kwestii rezultatów czasowych system ten sprawdzi się przy założeniu, że liczba miejsc na parkingu będzie stosunkowo mała (do 10 miejsc). Niestety, ale sam model detekcji pojazdów zużywa ogromne ilości zasobów (jeden skan obrazu zajmuje około 600 ms, czyli 60% czasu możliwego do przeznaczenia na pojedynczą klatkę). Przy wydłużeniu tego czasu do 3 s modele powinny być w stanie obsługiwać nawet bardzo dużą liczbę slotów.

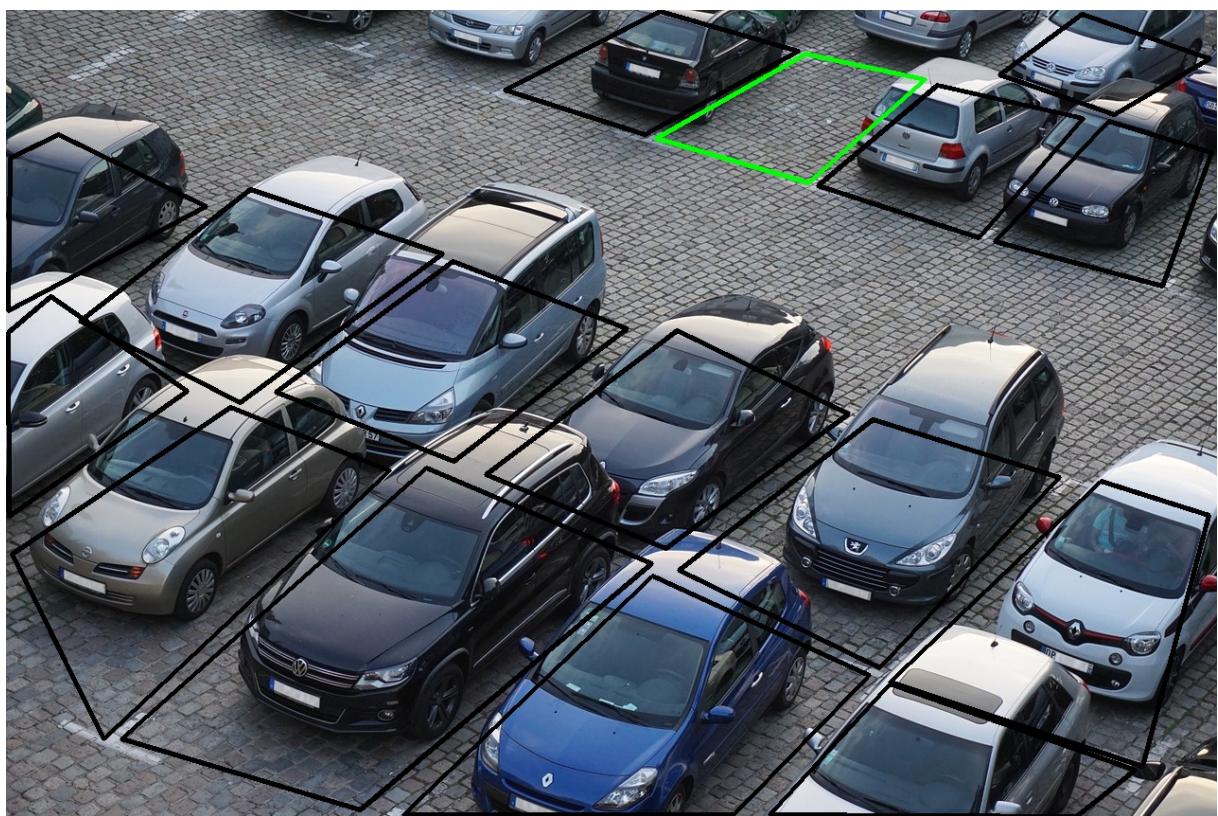
4.7. Pełna wersja demonstracyjna

W celach demonstracyjnych zaimplementowano obydwa systemy z wykorzystaniem zaprezentowanych wcześniej komponentów. Na wejściu przyjmowały one pozycje miejsc przeznaczonych do monitorowania oraz obraz. Następnie dokonywały modyfikacji obrazu, by dobrze zwizualizować obecną sytuację na parkingu. Przykładowe zmodyfikowane obrazy przedstawiają rys. 4.24.

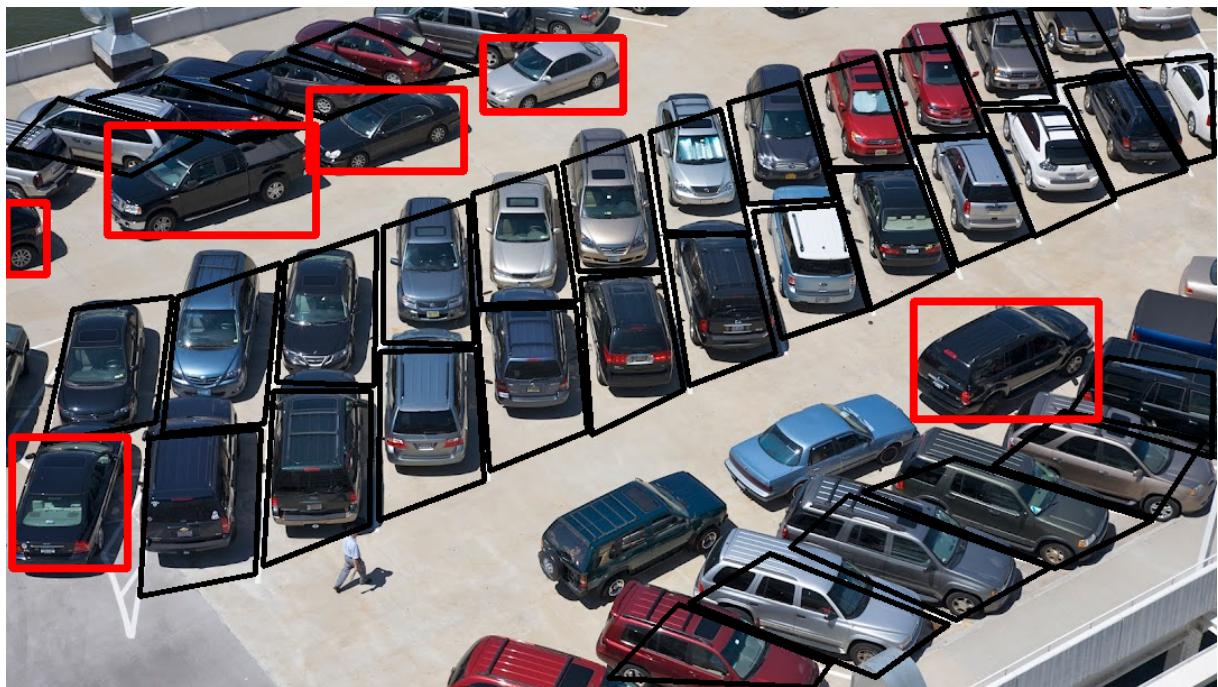
Obszary oznaczone kolorem zielonym reprezentują pojedyncze wolne miejsca. Obszary oznaczone na czarno to natomiast miejsca zajęte. Samochód stojący poza wyznaczonym miejscem jest otoczony czerwonym prostokątem. Gdyby był to pojazd uprzywilejowany, ramka miałaby kolor niebieski.

Ewaluacja systemu wykazała, że przeprowadzone wcześniej dobrze odzwierciedlały działanie prawdziwego systemu. Potwierdziły się m.in.:

- wysoka efektywność modelu klasyfikacji miejsc niezależnie od klasy,
- zmniejszone prawdopodobieństwo wykrycia pojazdów nieprawidłowo zaparkowanych w sytuacji, gdy kamera jest ustawiona blisko pionu,
- brak fałszywych alarmów w przypadku modelu detekcji.



Rys. 4.24. Przykład obrazu generowanego przez system



Rys. 4.25. Przykład obrazu z niewykrytymi pojazdami poza miejscami parkingowymi

5. Podsumowanie

Celem projektu było stworzenie systemu do monitorowania przestrzeni parkingowej. Został on zrealizowany: modele są w stanie (oczywiście z pewnym prawdopodobieństwem) odróżnić miejsca wolne od pustych (co ważne- w różnorodnych warunkach), znaleźć pojazdy poza wyznaczonymi miejscami a nawet określić, czy pojazdy są uprzywilejowane. Mogą dzięki temu spełnić zakładane zadanie, to znaczy wspierać człowieka w analizie obrazu. System posiada jednak także szereg wad. Przede wszystkim działanie systemu zależy w znacznym stopniu od liczby miejsc parkingowych, które muszą być monitorowane. Niezależnie od wybranego modelu, przy pewnej liczbie miejsc czas działania przekroczy zakładane 3 sekundy na pojedynczą klatkę. Ponadto, mimo nakierowania na osiągnięcie jak najwyższej efektywności, modele wciąż popełniają błędy w swoich przewidywaniach (szczególnie model nr.3, odpowiadający za podział pojazdów ze względu na przywilejowanie). Należy także zwrócić uwagę na fakt, iż na obecnym etapie system wymaga dość dokładnego rozumienia działania programu. W dodatku pojęcie przedstawione w projekcie nie ma zbyt dużej szansy na komercjalizację- obsługa każdej kamery wymagałaby oddzielnych podzespołów o stosunkowo dobrych parametrach, a pracujących na niemal pełni swoich możliwości, co za tym idzie- zużywających dużo energii. Niesprzyjający jest także fakt, że model Dalszy rozwój projektu powinien polegać na:

- wprowadzeniu przezroczystego interfejsu graficznego, który umożliwiłby pracę osobom niewykwalifikowanym,
- wprowadzeniu systemu zbierającego dane, takie jak chociażby poziom zajętości parkingu w danej chwili,
- udoskonaleniu modelu klasyfikacji pojazdów na uprzywilejowane i cywilne, przede wszystkim poprzez powiększenie zbioru danych.

Ponadto z pracy wyłania się bardzo ważny wniosek dotyczący sztucznej inteligencji. Jeśli porównamy wyniki z tabeli 4.6 oraz 4.7 zauważymy, że te same modele (a w niektórych przypadkach nawet bardziej rozbudowane) okazują się zupełnie nieskuteczne w sytuacji, gdy nie mają dostępu do odpowiedniego zbioru danych. Jako przykład można tu zestawić najprostszą sieć CNN z rozdziału 4.1 z modelem w architekturze EfficientNetV2B3. Wyszkolony na ponad 11000 próbek prosty model osiągnął na zbiorze testowym ponad 90% dokładności. EfficientNet wyszkolony na zaledwie 1000 próbek, w dodatku niezbyt dobrze dopasowanych do problemu, nie był w stanie przekroczyć progu nawet 70% dokładności.

Bibliografia

- [1] University of Stanford. *The AI index report*. Spraw. tech.
- [2] R. Tadeusiewicz. *Sieci neuronowe*. Akademicka Oficyna Wydawnicza RM, 1993.
- [3] Sasa Sambolek i Marina Ivasic-Kos. „Automatic Person Detection in Search and Rescue Operations Using Deep CNN Detectors”. W: *IEEE Access* 9 (2021), s. 37905–37922. DOI: [10.1109/ACCESS.2021.3063681](https://doi.org/10.1109/ACCESS.2021.3063681).
- [4] Vaibhav Tiwari i in. „Image Classification Using Deep Neural Network”. W: *2020 2nd International Conference on Advances in Computing, Communication Control and Networking (ICACCCN)*. 2020, s. 730–733. DOI: [10.1109/ICACCCN51052.2020.9362804](https://doi.org/10.1109/ICACCCN51052.2020.9362804).
- [5] Mingzhe Chen i in. „Artificial Neural Networks-Based Machine Learning for Wireless Networks: A Tutorial”. W: *IEEE Communications Surveys Tutorials* 21.4 (2019), s. 3039–3071. DOI: [10.1109/COMST.2019.2926625](https://doi.org/10.1109/COMST.2019.2926625).
- [6] Junhang Chen i in. „Sales Forecasting Using Deep Neural Network And SHAP techniques”. W: *2021 IEEE 2nd International Conference on Big Data, Artificial Intelligence and Internet of Things Engineering (ICBAIE)*. 2021, s. 135–138. DOI: [10.1109/ICBAIE52039.2021.9389930](https://doi.org/10.1109/ICBAIE52039.2021.9389930).
- [7] Saad Albawi, Tareq Abed Mohammed i Saad Al-Zawi. „Understanding of a convolutional neural network”. W: *2017 International Conference on Engineering and Technology (ICET)*. 2017, s. 1–6. DOI: [10.1109/ICEngTechnol.2017.8308186](https://doi.org/10.1109/ICEngTechnol.2017.8308186).
- [8] Kaiming He i in. „Deep Residual Learning for Image Recognition”. W: *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2016, s. 770–778. DOI: [10.1109/CVPR.2016.90](https://doi.org/10.1109/CVPR.2016.90).
- [9] Mingxing Tan i Quoc V. Le. „EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks”. W: *CoRR* abs/1905.11946 (2019). arXiv: [1905.11946](https://arxiv.org/abs/1905.11946).
- [10] Shuying Liu i Weihong Deng. „Very deep convolutional neural network based image classification using small training sample size”. W: *2015 3rd IAPR Asian Conference on Pattern Recognition (ACPR)*. 2015, s. 730–734. DOI: [10.1109/ACPR.2015.7486599](https://doi.org/10.1109/ACPR.2015.7486599).
- [11] François Chollet. „Xception: Deep Learning with Depthwise Separable Convolutions”. W: *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2017, s. 1800–1807. DOI: [10.1109/CVPR.2017.195](https://doi.org/10.1109/CVPR.2017.195).

- [12] Jeong-ah Kim, Ju-Yeong Sung i Se-ho Park. „Comparison of Faster-RCNN, YOLO, and SSD for Real-Time Vehicle Type Recognition”. W: *2020 IEEE International Conference on Consumer Electronics - Asia (ICCE-Asia)*. 2020, s. 1–4. DOI: [10.1109/ICCE-Asia49877.2020.9277040](https://doi.org/10.1109/ICCE-Asia49877.2020.9277040).
- [13] Joseph Redmon i in. „You Only Look Once: Unified, Real-Time Object Detection”. W: *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2016, s. 779–788. DOI: [10.1109/CVPR.2016.91](https://doi.org/10.1109/CVPR.2016.91).
- [14] Shaoqing Ren i in. „Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks”. W: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 39.6 (2017), s. 1137–1149. DOI: [10.1109/TPAMI.2016.2577031](https://doi.org/10.1109/TPAMI.2016.2577031).
- [15] Yaoshiang Ho i Samuel Wookey. „The Real-World-Weight Cross-Entropy Loss Function: Modeling the Costs of Mislabeling”. W: *IEEE Access* 8 (2020), s. 4806–4813. DOI: [10.1109/ACCESS.2019.2962617](https://doi.org/10.1109/ACCESS.2019.2962617).
- [16] Xin Zhang i in. „Machine Vision On-line Detection System: Applications and Standardization Requirements”. W: *2020 IEEE International Conference on Artificial Intelligence and Computer Applications (ICAICA)*. 2020, s. 1200–1204. DOI: [10.1109/ICAICA50127.2020.9181902](https://doi.org/10.1109/ICAICA50127.2020.9181902).
- [17] J. Illingworth i J. Kittler. „The Adaptive Hough Transform”. W: *IEEE Transactions on Pattern Analysis and Machine Intelligence PAMI-9.5* (1987), s. 690–698. DOI: [10.1109/TPAMI.1987.4767964](https://doi.org/10.1109/TPAMI.1987.4767964).
- [18] Allam Shehata Hassanein i in. „A Survey on Hough Transform, Theory, Techniques and Applications”. W: *CoRR* abs/1502.02160 (2015). arXiv: [1502.02160](https://arxiv.org/abs/1502.02160).
- [19] Riad Kanan i Houssam Arbess. „An IoT-Based Intelligent System for Real-Time Parking Monitoring and Automatic Billing”. W: *2020 IEEE International Conference on Informatics, IoT, and Enabling Technologies (ICIoT)*. 2020, s. 622–626. DOI: [10.1109/ICIoT48696.2020.9089589](https://doi.org/10.1109/ICIoT48696.2020.9089589).
- [20] Yuxin Song i in. „Vision-Based Parking Space Detection: A Mask R-CNN Approach”. W: *2021 IEEE/CIC International Conference on Communications in China (ICCC)*. 2021, s. 300–305. DOI: [10.1109/ICCC52777.2021.9580236](https://doi.org/10.1109/ICCC52777.2021.9580236).
- [21] P. Choorat, C. Sirikornkarn i T. Pramoun. „License Plate Detection and Integral Intensity Projection for Automatic Finding the Vacant of Car Parking Space”. W: *2019 34th International Technical Conference on Circuits/Systems, Computers and Communications (ITC-CSCC)*. 2019, s. 1–4. DOI: [10.1109/ITC-CSCC.2019.8793297](https://doi.org/10.1109/ITC-CSCC.2019.8793297).
- [22] *Dokumentacja biblioteki Tensorflow*.
- [23] *Dokumentacja biblioteki OpenCV*.
- [24] Martin Marek. *Image-Based Parking Space Occupancy Classification: Dataset and Baseline*. 2021. arXiv: [2107.12207](https://arxiv.org/abs/2107.12207) [cs.CV].
- [25] Google. *Google grafika*.

- [26] *Dokumentacja biblioteki Keras.*
- [27] Merope Manataki, Antonis Vafidis i Apostolos Sarris. „Comparing Adam and SGD optimizers to train AlexNet for classifying GPR C-scans featuring ancient structures”. W: *2021 11th International Workshop on Advanced Ground Penetrating Radar (IWAGPR)*. 2021, s. 1–6. DOI: [10.1109/IWAGPR50767.2021.9843162](https://doi.org/10.1109/IWAGPR50767.2021.9843162).
- [28] Rahul Singh i in. „Impact of Adam, Adadelta, SGD on CNN for White Blood Cell Classification”. W: *2023 5th International Conference on Smart Systems and Inventive Technology (ICSSIT)*. 2023, s. 1702–1709. DOI: [10.1109/ICSSIT55814.2023.10061068](https://doi.org/10.1109/ICSSIT55814.2023.10061068).
- [29] Shiron Thalagala i Chamila Walgampaya. „Application of AlexNet convolutional neural network architecture-based transfer learning for automated recognition of casting surface defects”. W: *2021 International Research Conference on Smart Computing and Systems Engineering (SCSE)*. T. 4. 2021, s. 129–136. DOI: [10.1109/SCSE53661.2021.9568315](https://doi.org/10.1109/SCSE53661.2021.9568315).
- [30] Chao Luo i in. „How Does the Data set Affect CNN-based Image Classification Performance?” W: *2018 5th International Conference on Systems and Informatics (ICSAI)*. 2018, s. 361–366. DOI: [10.1109/ICSAI.2018.8599448](https://doi.org/10.1109/ICSAI.2018.8599448).
- [31] Jeong-ah Kim, Ju-Yeong Sung i Se-ho Park. „Comparison of Faster-RCNN, YOLO, and SSD for Real-Time Vehicle Type Recognition”. W: *2020 IEEE International Conference on Consumer Electronics - Asia (ICCE-Asia)*. 2020, s. 1–4. DOI: [10.1109/ICCE-Asia49877.2020.9277040](https://doi.org/10.1109/ICCE-Asia49877.2020.9277040).
- [32] <https://universe.roboflow.com/piotr-z-xuaxc/poland-special-vehicles>.