Zeszyty Naukowe WWSI, No 25, Vol. 15, 2021, s. 7-31
DOI: 10.26348/znwwsi.25.7

System automatycznej detekcji tablic rejestracyjnych i identyfikacji numerów pojazdów wykorzystujący sieci neuronowe

Julia Błaszczyk[*], Robert Janowski[**]

Warszawska Wyższa Szkoła Informatyki

Abstrakt

W publikacji opisano kolejne etapy tworzenia aplikacji do automatycznego rozpoznawania tablic rejestracyjnych i identyfikacji numerów pojazdów z wykorzystaniem sieci neuronowych i podejścia Deep Learning, począwszy od przygotowania danych do trenowania, a skończywszy na testowaniu skuteczności działania. Do implementacji mechanizmu detekcji tablic rejestracyjnych został wykorzystany model sieci EfficientDet, a do optycznego rozpoznawania numerów identyfikacyjnych na wykrytych tablicach biblioteki Pytesseract oraz OpenCV. Weryfikacja skuteczności działania aplikacji dotyczyła oceny poprawności detekcji tablic rejestracyjnych oraz rozpoznania numerów identyfikacyjnych. Podstawowymi kryteriami oceny były powszechnie używane w takich zastosowaniach miary: dokładność, precyzja, czułość. Przeprowadzone testy pozwoliły ustalić optymalne wartości parametrów dla procesu detekcji tablic (wartości hiperparametrów, czyli rozmiar sieci, liczba kroków, liczba epok, wagi, współczynnik uczenia, rozmiar paczki danych podczas trenowania oraz wartość progu podczas weryfikacji działania), a także dla procesu rozpoznawania numerów (zachowanie lub usuwanie pikseli stykających się z granicą wyciętego obrazu, wartości parametrów filtra bilateralnego, operacji progowania).

Słowa kluczowe **–** Sieci neuronowe, komputerowe rozpoznawanie obrazów, detekcja tablic rejestracyjnych, EfficientDet, Pytesseract, ALPR

[*] E-mail: julia.blaszczyk.97@gmail.com
[**] E-mail: rjanowski@poczta.wwsi.edu.pl

Zgłoszono do druku 6 listopada 2021 r.

1. Wstęp

Uczenie maszynowe to w ostatnich latach prężnie rozwijająca się dziedzina informatyki. Dostarcza ona odpowiednich narzędzi do automatyzacji zadań powiązanych z analizą danych. Pewne czynności z naszego codziennego życia, które dotychczas były wykonywane przez ludzi, ze względu na wcześniej nieosiągalny poziom precyzji przez systemy komputerowe, stają się obiektem zainteresowań i wyzwaniem dla programistów. Jednym z takich obszarów jest system monitoringu. Zamiast pracowników monitorujących obraz wyświetlany przez kamery, wystarczy oprogramowanie analizujące strumień wideo pod kątem interesujących użytkownika informacji.

System taki mógłby znaleźć zastosowanie przy monitorowaniu ruchu samochodowego w obrębie chronionego obiektu lub – w połączeniu z systemem kontroli dostępu – udzielaniu dostępu do nadzorowanego obszaru. Elementem koniecznym do realizacji jest jednak automatyczna detekcja tablic rejestracyjnych i rozpoznawanie na nich numerów identyfikacyjnych pojazdów (*Automatic License Plate Recognition*; ALPR). Zadanie to typowo dzielone jest na cztery części, tworzące razem całościowe oprogramowanie: detekcja pojazdu, detekcja tablicy rejestracyjnej, segmentacja znaków i rozpoznawanie znaków. Dla uproszczenia dwa ostatnie zadania mogą zostać określone jako optyczne rozpoznawanie znaków (*Optical Character Recognition*; OCR).

Dostępnych jest wiele rozwiązań korzystających z konwolucyjnych sieci neuronowych [1] do detekcji tablic rejestracyjnych. W publikacji [2] autorzy do wykrycia pojazdu używają sieci YOLOv2 [3]. Następnie dla każdego obszaru detekcji proponowana sieć wykrywania obiektów płaskich wypaczonych (WPOD-NET [2]) wyszukuje tablicę rejestracyjną i cofa jedną z transformacji współrzędnych, umożliwiając wyprostowanie obszaru tablicy do prostokąta przypominającego widok od frontu. Te pozytywne i skorygowane detekcje są przesyłane do sieci OCR w celu ostatecznego rozpoznania znaków. Na 5 różnych zbiorach danych osiągnęli oni dokładność na poziomie 89,33-82,98% (zależnie od wykorzystania danych wygenerowanych sztucznie i pominięcia etapu przekształcenia tablicy) [2].

Jednym z komercyjnie dostępnych na rynku rozwiązań jest narzędzie Amazon Rekognition [4]. Jest to oprogramowanie do analizy zdjęć i wideo wykorzystujące uczenie maszynowe. Umożliwia ono detekcję obiektów, ludzi, czynności czy tekstu.

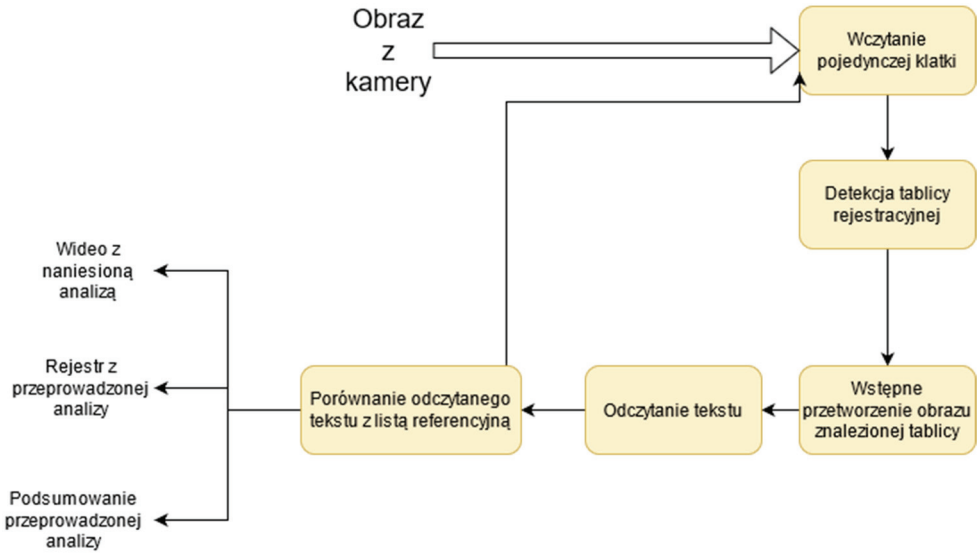
Jego działanie zapewnia, ale nie ogranicza się do detekcji i rozpoznawania tablic rejestracyjnych. W testach na 5 różnych zbiorach danych tablic rejestracyjnych oprogramowanie osiągnęło średnią dokładność wynoszącą 56,60%, mając problemy z wykrywaniem i odczytywaniem tablic z dalekiej odległości [2].

Innowacyjnym podejściem wykazali się autorzy [5], którzy stworzyli oprogramowanie będące w stanie rozpoznać i odczytać tablice rejestracyjne na podstawie ich tła, w czasie rzeczywistym. Podzielili oni zadanie ALPR na kolejne kroki: detekcję samochodu, detekcję tablicy i klasyfikację jej tła oraz symultaniczne wykrycie i rozpoznanie znaków tablicy, przez wczytanie znalezionego obszaru tablicy do sieci, dzięki czemu mogli pominąć zadanie segmentacji znaków.

Autorzy wykorzystali modele inspirowane konwolucyjną siecią neuronową YOLO (*You Only Look Once*), takie jak YOLOv2 [3], Fast-YOLOv2 [6] oraz CR-NET [7] do rozpoznawania znaków. Dla każdego kroku zadania ALPR wytrenowali oni pojedynczą sieć na zdjęciach z ośmiu różnych, publicznie dostępnych zbiorów danych, by uzyskać jak największy przekrój zróżnicowanych tablic (z różnych obszarów świata). Dzięki dodaniu do kroku detekcji tablicy klasyfikację jej tła jako jedną z predefiniowanych klas (Ameryka, Brazylia, Chiny, Europa, Taiwan) mogli oni bez znacznego zwiększenia kosztów zastosować do zadania rozpoznania znaków specyficzne podejście dla każdego z regionów, biorąc pod uwagę wzorce tablic obowiązujące w danym obszarze. W testach na ośmiu zbiorach danych uzyskali oni średnią dokładność na poziomie 96,9%. Komercyjne systemy, z którymi autorzy porównywali swoje oprogramowanie osiągnęły następujący średni poziom dokładności: Sightound – 87,8% i OpenAPLR – 90,7%.

2. Architektura systemu

Architektura oprogramowania ALPR została przedstawiona przy użyciu modelu koncepcyjnego, który definiuje jego strukturę i zachowanie. Rysunek 1 przedstawia projekt architektury programu automatycznej detekcji i rozpoznawania tablic rejestracyjnych. Program ALPR jest aplikacją, która odczytuje i przetwarza obraz wideo z widocznym wjazdem do obszaru monitorowanego oraz zapisuje przetworzony film, rejestr i obszerne podsumowanie przeprowadzonej analizy.



Rysunek 1. Projekt architektury oprogramowania

Głównym celem programu do detekcji i rozpoznawania tablic rejestracyjnych ALPR jest uzyskanie informacji o pojazdach w odniesieniu do ich numerów rejestracyjnych. Dzięki tym informacjom możemy zarządzać udzielaniem dostępu do stref chronionych. Kamera, której obraz wideo podlega analizie, jest umieszczona tak, aby uzyskać wyraźny widok przedniej tablicy rejestracyjnej. Filmy są przechwytywane i przechowywane w repozytorium. Po rozpoznaniu tablicy i identyfikacji numerów rejestracyjnych są one nanoszone na wejściowy film. Kolejne kroki działania programu zostały pokazane na rysunku 1:

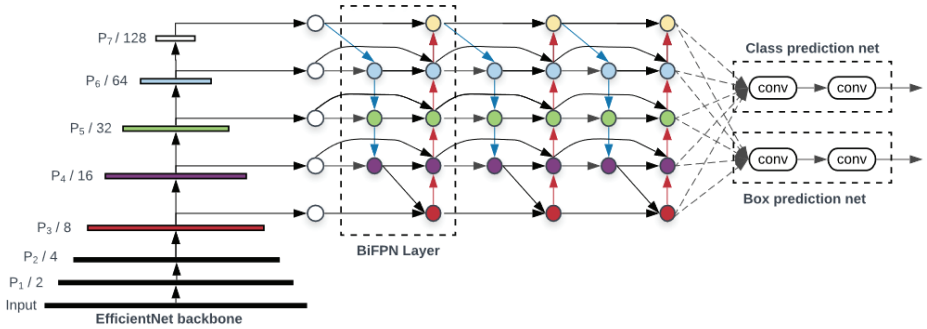
i) Film jest wczytywany do systemu.

ii) Oprogramowanie znajduje tablicę rejestracyjną, przetwarza wykryty obraz, odczytuje numer rejestracyjny pojazdu i sprawdza czy jest on na liście referencyjnej. Operacje powtarzane są dla każdej kolejnej klatki wideo.

iii) W momencie braku kolejnej klatki do przetworzenia, program kończy swoje działanie. Udostępniane są: film z naniesionymi, uprzednio odczytanymi numerami rejestracyjnymi, rejestr oraz podsumowanie przeprowadzonej analizy.

Stworzone do detekcji tablic oprogramowanie ALPR wykorzystuje sieci neuronowe i podejście znane jako Deep Learning [8]. Istnieje wiele modeli sieci neuronowych wykorzystujących technikę Deep Lerning, tu został wybrany model EfficientDet [9], który cechuje się skalowalną architekturą do detekcji. W porównaniu do innych modeli do detekcji zapewnia on zarówno wyższą dokładność, jak i lepszą wydajność, jednocześnie biorąc pod uwagę szerokie spektrum ograniczeń zasobów.

Model EfficientDet składa się z trzech głównych elementów: sieci szkieletowej EfficientNet [10], warstwy fuzji tzw. BiFPN (*Weighted Bi-directional Feature Pyramid Network*) oraz głowy – rysunek 2.



Rysunek 2. Architektura EfficientDet [9]

Projektując architekturę EfficientDet autorzy wykorzystali jako sieć szkieletową EfficientNet wytrenowaną na zbiorze danych ImageNet [11]. Sieć ta osiąga lepsze wyniki na wzorcowym zbiorze danych klasyfikacyjnym ImageNet względem innych szkieletów o tej samej lub większej liczbie parametrów i operacji zmiennoprzecinkowych na sekundę (*floating point operations per second*, FLOPS). Kolejną częścią architektury jest warstwa fuzji, czyli w tym przypadku BiFPN, który jest zoptymalizowaną wersją klasycznego tzw. FPN (*Feature Pyramid Network*) odpowiedzialnego za wieloskalową fuzję cech. Istotnym usprawnieniem względem innych sieci do detekcji jest nowa metoda skalowania. Parametr ϕ jest głównym współczynnikiem identyfikującym rodzaj modelu z rodziny EfficientDet i jest używany do równomiernego skalowania szerokości, głębokości (szkieletu), rozdzielczości (wejścia) oraz liczby warstw i filtrów wyjściowych BiFPN. Wraz ze wzrostem parametru

ϕ rośnie liczba parametrów modelu i operacji zmiennoprzecinkowych na sekundę, przy jednoczesnym zwiększaniu się współczynnika precyzji.

3. Implementacja

3.1. Wybór narzędzi i środowiska wykonawczego

Do uruchamiania, implementacji i testowania oprogramowania został wybrany język Python i środowisko Jupyter [12] dostarczane i obsługiwane przez Google – Google Colaboratory (Colab). Jest to środowisko względnie proste w obsłudze, niezależne od konkretnej platformy (dostęp do swoich notebooków uzyskujemy przez przeglądarkę, na każdym systemie będzie działał tak samo) i bezpłatne. Umożliwia ono pracę z procesorami CPU (*Central Processing Unit*), układami graficznymi GPU (*Graphical Processing Unit*) oraz procesorami TPU (*Tensor Processing Unit*). Jednak ze względu na ograniczone zasoby w środowisku wykonawczym Colab, model sieci neuronowej użytej w oprogramowaniu ALPR był trenowany na zewnętrznym serwerze, na którym dostępne były dwa układy GPU GeForce RTX 2080 Ti.

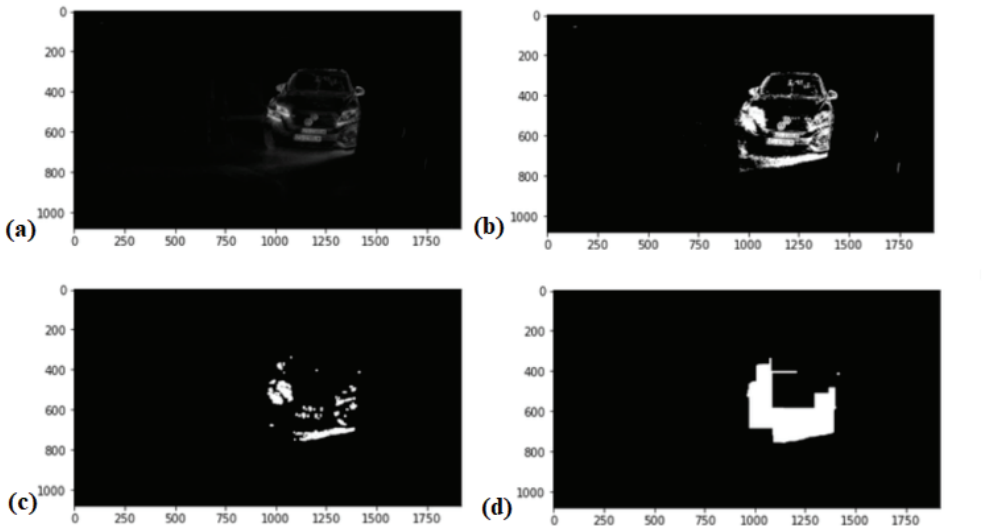
Ponadto do wizualizacji procesu trenowania sieci neuronowych zostało użyte narzędzie Tensorboard [13], które daje możliwość śledzenia na bieżąco takich wskaźników jak dokładność i strata, przeglądania histogramów wag, błędów systematycznych czy wizualizacji wykresu modelu.

3.2. Przygotowanie danych treningowych

Ze względu na czasochłonność procesu przygotowania danych, proces ten został zautomatyzowany, dzięki czemu w każdej chwili łatwo powiększyć zbiór danych. Automatyzacja przygotowania danych treningowych polegała na wykorzystaniu dodatkowego oprogramowania, które pozwoliło na otrzymanie zdjęć z wyodrębnionymi pojazdami i zaznaczonymi na nich współrzędnymi tablic rejestracyjnych. Pierwotny zbiór danych składał się z 884 zdjęć. Na każdym ze zdjęć widoczny jest co najmniej jeden samochód z przednią tablicą rejestracyjną. Z uwagi na fakt, że dostępne algorytmy do wykrywania tablic rejestracyjnych nie były w stanie wykryć tablicy na pełnym zdjęciu, zdecydowano się na zastosowanie algorytmu do detekcji samochodu

by wygenerować zdjęcia, na których wykrycie tablicy będzie już możliwe. Danymi wejściowymi do algorytmu detekcji samochodu było 884 zdjęć, które tworzyły 442 pary poklatkowych zdjęć (rysunek 3).

Rysunek 3. Para poklatkowych zdjęć wczytanych do algorytmu detekcji pojazdu



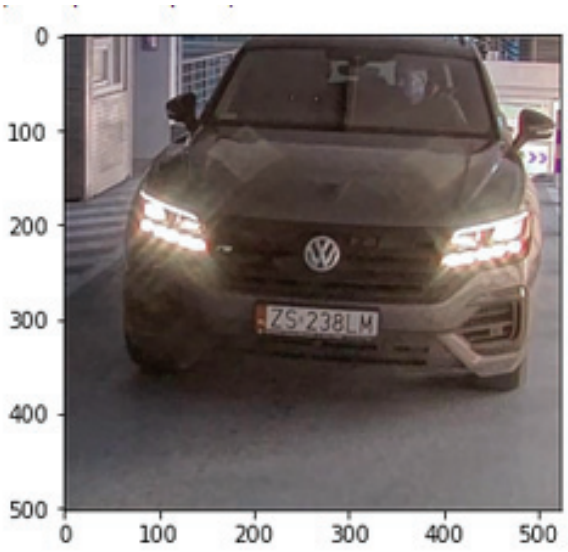
Rysunek 4. Wizualizacja operacji przeprowadzonych na zdjęciach w procesie przygotowawczym: (a) – bezwzględna różnica pikseli pomiędzy macierzami wczytanych zdjęć, (b) – operacja progowania, (c) – otwarcie morfologiczne, (d) – zamknięcie morfologiczne

Algorytm wczytywał dwa następujące po sobie zdjęcia, konwertował przestrzeń kolorów obu zdjęć z RGB na czarno-białą i wyświetlał bezwzględną różnicę pomiędzy pikselami dwóch macierzy zdjęć (rysunek 4 (a)).

W następnym krokiem było progowanie (*thresholding*). Dla każdego piksela zastosowana została ta sama wartość progowa – jeżeli wartość piksela była mniejsza od progu, to została ona ustawiona na 0, w przeciwnym razie wartość została zmieniona na maksymalną (rysunek 4(b)).

Następnie algorytm wykonał przekształcenia morfologiczne: otwarcie, by pozbyć się szumu ze zdjęcia (rysunek 4 (c)), oraz zamknięcie by wypełnić niewielkie puste przestrzenie wewnątrz obiektów na pierwszym planie (rysunek 4 (d)).

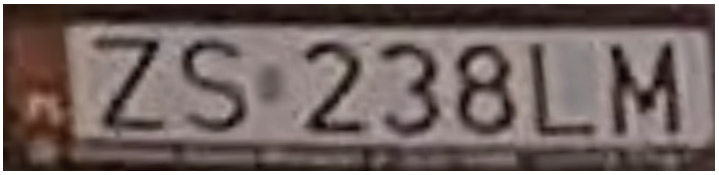
Na tak przygotowanym zdjęciu algorytm nakreślił krzywą łącząca wszystkie ciągłe punkty mające ten sam kolor, czyli kontur naszego samochodu. Po uzyskaniu konturu wokół niego narysowany został prostokąt o współrzędnych dobranych tak, by prostokąt całkowicie obejmował kontur. W celu uzyskania pewności, że cały samochód (a przynajmniej cała tablica rejestracyjna) znalazł się na docelowym zdjęciu, wcześniej uzyskany prostokąt został rozszerzony o stałą wartość i tak uzyskane zdjęcie (rysunek 5) było zapisywane do pliku z opracowanymi samochodami.



Rysunek 5. Zdjęcie wyjściowe z algorytmu do detekcji pojazdów

Wyżej opisany algorytm ze zdjęć 442 samochodów utworzył 433 pliki wyjściowe, co oznacza, że 9 samochodów nie zostało rozpoznanych. Po weryfikacji i usunięciu zdjęć błędnie oznaczonych, pozostało 428 zdjęć poprawnie wykrytych samochodów.

Tak przygotowane zdjęcia zostały podane jako dane wejściowe do wstępnie wytrenowanego modelu WPOD-NET [2] do detekcji tablic rejestracyjnych, który jako dane wyjściowe zwrócił zdjęcia znalezionych tablic (rysunek 6) i ich koordynaty. Z 428 zdjęć z rejestracjami, model zwrócił 392 zdjęcia. Po weryfikacji i usunięciu błędnie wykrytych tablic, pozostały 362 zdjęcia z poprawnie wykrytymi tablicami rejestracyjnymi.



Rysunek 6. Tablica rejestracyjna wykryta przez model WPOD-NET

Ostatnim krokiem było odczytanie tekstu ze zdjęć tablic rejestracyjnych. Do tego zadania został wykorzystany wstępnie wytrenowany model MobileNets [14]. Model MobileNets to architektura konwolucyjnej sieci neuronowej o niewielkim rozmiarze, dużej szybkości i wysokiej dokładności, zaprojektowana z myślą o aplikacjach mobilnych [14].

Ostatecznie model odczytał 167 rejestracji o długości siedmiu znaków, ale po sprawdzeniu i odrzuceniu błędnie odczytanych znaków zostało 71 tablic odczytanych poprawnie. Do tzw. podstawowej prawdy (*ground truth*) wygenerowanej automatycznie, zostały dodane i oznaczone ręcznie pozostałe dane czyli te, które nie zostały poprawnie sklasyfikowane w opisanym procesie. Tak przygotowane dane posłużyły później do wytrenowania modelu do detekcji tablic i do fazy testów.

Na późniejszym etapie tj. po pierwszych próbach treningu z danymi opisanymi powyżej, ale przed pierwszym treningiem opisanym w tabeli 1, do zbioru danych treningowych zostało dołączonych jeszcze 200 zdjęć bez widocznych tablic rejestracyjnych, które nie wymagały żadnego dodatkowego przygotowania.

3.3. Budowanie modelu sieci neuronowej do detekcji tablic rejestracyjnych

Po przygotowaniu danych treningowych i walidacyjnych, opcjonalnym (w przypadku braku przyjmowane są wartości domyślne) skonfigurowaniu hiperparametrów: rozmiar sieci, liczba kroków, liczba epok, wagi, współczynnik uczenia oraz rozmiar paczki danych i pobraniu wag wytrenowanych wcześniej modeli (wagi sieci EfficientNet wytrenowanej na zbiorze danych ImageNet [11] oraz wagi sieci EfficientDet wytrenowanej na zbiorze danych COCO [15]) został rozpoczęty trening poprzez uruchomienie skryptu treningowego train.py [16]. Do skryptu przekazane zostały następujące parametry: ϕ=0, liczba kroków=1000, a liczba epok=50. Zmiennymi parametrami przy dostrajaniu modelu (*fine-tuning*) były wagi, współczynnik uczenia oraz rozmiar paczki danych.

Podczas treningu punkty kontrolne modelu po przejściu kolejnej epoki i logi z treningu były automatycznie zapisywane. Trening zależnie od ustawionego rozmiaru paczki danych kończył się po średnio 8,5 godziny (dla rozmiaru paczki danych ustawionego na 8) lub po 4,5 godziny (dla rozmiaru paczki danych ustawionego na 4). Dane treningowe dla pierwszego treningu to 397 zdjęć z widoczną przednią tablicą rejestracyjną i dwie klasy detekcji: 0: „brak_tablicy”, 1: „tablica”. Dane walidacyjne to: 36 zdjęć z widoczną przednią tablicą rejestracyjną i dwie klasy detekcji: 0: „brak_tablicy”, 1: „tablica”. Zmiany w danych zostały wyszczególnione w tabeli 1. Po zmianach w danych każdy kolejny trening dziedziczył zmiany z treningu poprzedniego, kolejne zmiany sumują się ze zmianami z poprzedniego treningu (chyba, że zostało wyszczególnione inaczej).

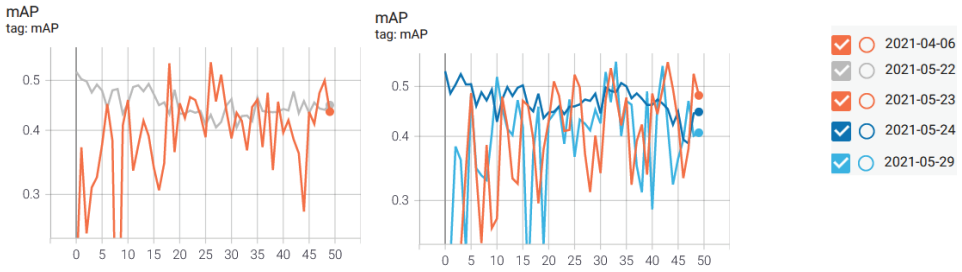
Wykresy przedstawione na rysunkach 7-11 obrazują zmiany wartości podstawowych parametrów charakteryzujących jakość procesu trenowania.

Rysunek 7 prezentuje wartość parametru średniej ze średnich precyzji (*Mean average precision, mAP*).

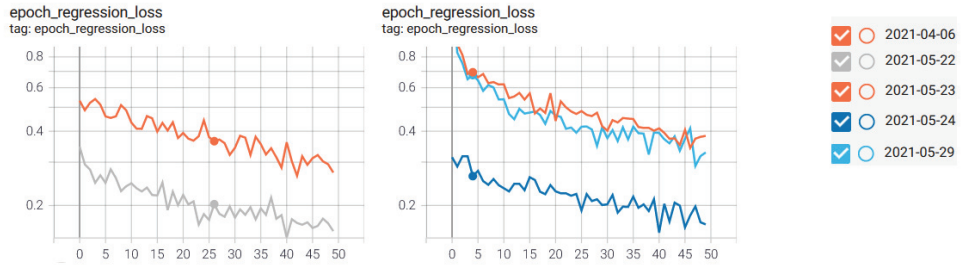
Rysunki 8 i 9 oraz 10 i 11 prezentują wartości funkcji strat (ogólnej i regresji) dla danych treningowych (rysunek 8 i 9) i walidacyjnych (rysunek 10 i 11).

Tabela 1. Podsumowanie przeprowadzonych treningów

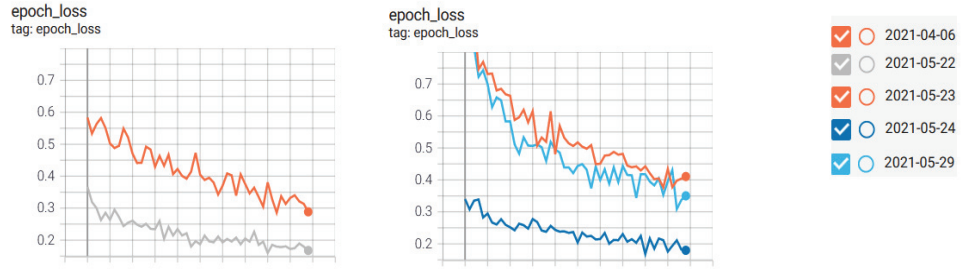
Data treningu	**Początkowy punkt kontrolny (wagi)**	**Współczynnik uczenia**	**Rozmiar paczki danych**	**Zmiany w danych**
06.04.2021	Imagenet	0,001	8	-
08.04.2021	Epoka 26 z 06.04.2021	0,001	8	-
19.05.2021	ImageNet	0,001	8	Dodano 9 zdjęć do zdjęć treningowych bez widocznej tablicy rejestracyjnej. Klasa detekcji zmieniona na: 0: ‘tablica’
20.05.2021	Epoka 46 z 19.05.2021	0,0001	4	-
20.05.2021 v2	Epoka 46 z 19.05.2021	0,0005	4	-
21.05.2021	Epoka 46 z 19.05.2021	0,0001	8	Dodano 90 pustych zdjęć treningowych bez widocznej tablicy rejestracyjnej. Dodano 12 pustych zdjęć walidacyjnych bez widocznej tablicy rejestracyjnej.
22.05.2021	Epoka 47 z 19.05.2021	0,0001	8	-
23.05.2021	ImageNet	0,001	8	-
24.05.2021	Epoka 44 z 23.05.2021	0,0001	8	-
28.05.2021	ImageNet	0,001	8	Usunięto puste zdjęcia ze zbioru zdjęć treningowych.
29.05.2021	ImageNet	0,001	8	Usunięto puste zdjęcia ze zbioru zdjęć walidacyjnych



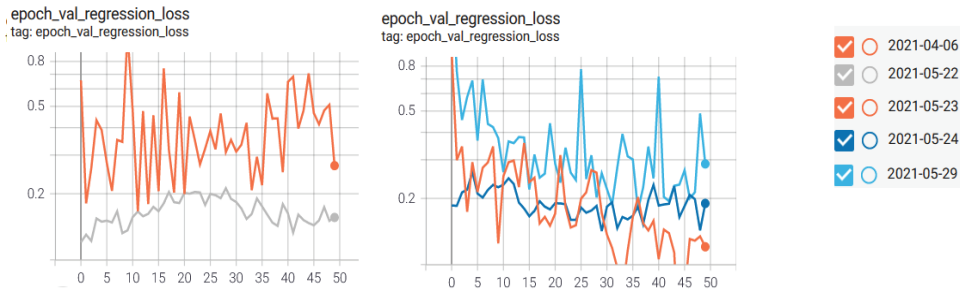
Rysunek 7. Wartości parametru średniej ze średnich precyzji dla wybranych treningów. Najwyższe wartości: **06.04.2021**: 54%, **22.05.2021**: 49%, **23.05.2021**: 56%, **24.05.2021**: 51%, **29.05.2021**: 56%.



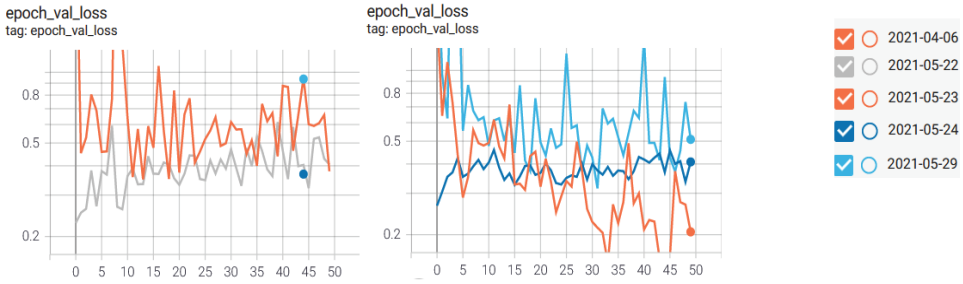
Rysunek 8. Wartości funkcji straty dla zadania regresji dla wybranych treningów. Wartości dla epok z najwyższym mAP: **06.04.2021**: 0.43, **22.05.2021**: 0.22, **23.05.2021**: 0.37, **24.05.2021**: 0.2, **29.05.2021**: 0.42.



Rysunek 9. Wartości funkcji straty dla wybranych treningów. Wartości dla epok z najwyższym mAP: **06.04.2021**: 0.47, **22.05.2021**: 0.24, **23.05.2021**: 0.40, **24.05.2021**: 0.21, **29.05.2021**: 0.44.



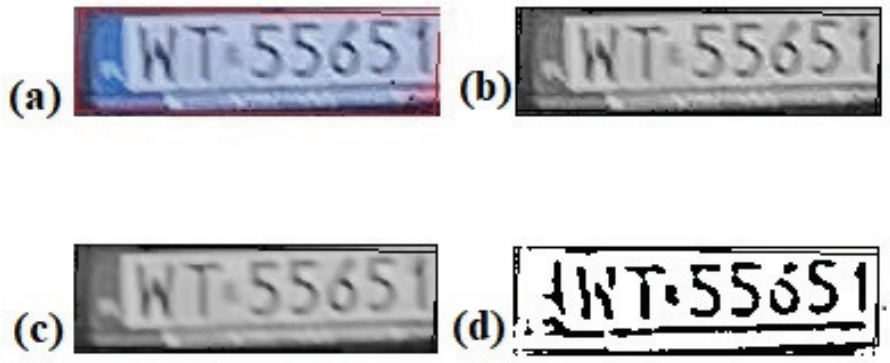
Rysunek 10. Wartości funkcji straty dla zadania regresji, dla zbioru walidacyjnego, dla wybranych treningów. Wartości dla epok z najwyższym mAP: **06.04.2021**: 0.2, **22.05.2021**: 0.17, **23.05.2021**: 0.11, **24.05.2021**: 0.16, **29.05.2021**: 0.39.



Rysunek 11. Wartości funkcji straty dla zbioru walidacyjnego, dla wybranych treningów. Wartości dla epok z najwyższym mAP: **06.04.2021**: 0.35, **22.05.2021**: 0.44, **23.05.2021**: 0.16, **24.05.2021**: 0.36, **29.05.2021**: 0.59.

3.4. Opis wykorzystanego systemu optycznego rozpoznawania znaków (OCR) i zastosowanych usprawnień

Do optycznego rozpoznawania tekstu zostało wykorzystane narzędzie Pytesseract [17] i silnik Tesseract [18]. Danymi wejściowymi algorytmu była tablica rejestracyjna wykryta przez sieć do detekcji – rysunek 12 (a). Przed odczytaniem tekstu z obrazu został zastosowany szereg operacji, mających na celu usprawnienie działania sieci do odczytania znaków.



Rysunek 12. Wizualizacja operacji przeprowadzonych na zdjęciach tablic rejestracyjnych: (a) – zdjęcie wejściowe, (b) – zmieniona przestrzeń kolorów, (c) – nałożenie filtra bilateralnego, (d) – operacja progowania

W pierwszym kroku przestrzeń kolorów zdjęcia została przekonwertowana z barw RGB na barwy czarno-białe i przy pomocy metody skimage.segmentation.clear_border [19] zostały usunięte piksele z pierwszego planu, które dotykały granicy wyciętego obrazu – rysunek 12 (b).

Kolejnym etapem przygotowania tablicy do etapu identyfikacji znaków było zastosowanie filtru bilateralnego – rysunek 12 (c). Zastępuje on intensywność każdego piksela średnią ważoną wartością intensywności z pobliskich pikseli, co ma na celu redukcję szumów przy zachowaniu krawędzi obrazu.

W ostatnim kroku przygotowawczym zostało wykorzystane progowanie. Dla każdego piksela została zastosowana ta sama wartość progowa, jeżeli wartość piksela była mniejsza od progu, to została ona ustawiona na 0, w przeciwnym razie wartość została zmieniona na maksymalną – rysunek 12 (d).

Tak przygotowany obraz był gotowy do próby odczytania tekstu. Po określeniu parametru metody segmentacji stron (*Page Segmentation Method, PSM*) z jakim będzie działał Pytesseract na 7, czyli „traktuj obraz jako pojedynczą linię tekstu", zostały wykonane operacje odczytywania znaków za pomocą metody image_to_string. W celu ujednolicenia wyników, tekst otrzymany na wyjściu został przefiltrowany przez listę alfanumeryczną w celu pozbycia się znaków powszechnie niewystępujących na polskich tablicach rejestracyjnych. W przypadku gdy metoda image_to_string nic nie zwróciła lub zwrócony tekst miał więcej niż 7 znaków, bądź

mniej niż 4, przygotowane wcześniej zdjęcie tablicy było przycinane z każdej strony i ponownie była wykonywana na nim operacja optycznego rozpoznawania znaków. Gdy po kilkukrotnym wykadrowaniu odczytany tekst nadal nie spełniał powyższego warunku, próba odczytania tekstu była kończona.

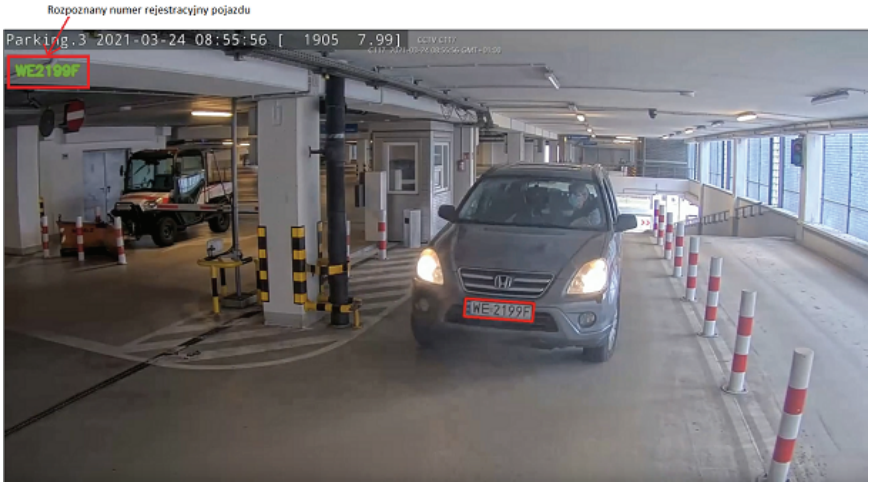
W celu przetwarzania strumienia wideo została użyta biblioteka OpenCV [20], która oferuje różne funkcje do operacji na obrazie i wideo. Dzięki OpenCV można przechwytywać wideo z kamery i wykonywać wymagane operacje na poszczególnych klatkach filmu. Uruchamiając skrypt main.py użytkownik podaje trzy argumenty: ścieżkę do pliku wideo wejściowego, ścieżkę do zapisu filmu wyjściowego oraz ścieżkę do pliku referencyjnego z wypisanymi tablicami, uprawnionymi do znajdowania się w strefie monitorowanej.

Praca na filmie wideo, sprowadza się do pracy na pojedynczych, poklatkowych obrazach według algorytmu zobrazowanego na rysunku 1. Na każdej klatce jest wyszukiwana tablica rejestracyjna, i w przypadku znalezienia jej przez model do detekcji, jest ona oznaczana ramką. Taki wycinek trafia do skryptu do rozpoznawania znaków. Chcąc zapewnić, aby tekst wyświetlany na wideo był jak najbardziej zbliżony do stanu rzeczywistego, ze wszystkich odczytanych sekwencji znaków dla jednego przejeżdżającego samochodu (liczba klatek z wykrytą tablicą równa próbie odczytania tekstu) wyciągana jest mediana przy użyciu metody levenshtein.median [21]. Informacja o tym konsensusowym wyniku zostaje naniesiona na obraz (rysunek 13).

W przypadku odczytania niewystarczającej liczby numerów rejestracyjnych dla jednego samochodu, wynik OCR nie jest wyświetlany na wideo, ze względu na duże prawdopodobieństwo błędnego odczytu.

Przetworzenie całego filmu wejściowego przez skrypt powoduje zapisanie pliku wyjściowego wideo w miejscu wcześniej wskazanym przez użytkownika przy uruchamianiu skryptu. W tym samym miejscu zapisywane są również dwa pliki z danymi wygenerowanymi podczas działania skryptu. W pierwszym (data.json) znajduje się rejestr z datą i godziną przejazdu samochodu oraz medianą z odczytanych tablic dla danego przejazdu. Drugi plik z podsumowaniem (summary.log) dostarcza większej ilości informacji. Po skończonej analizie wideo użytkownik może odczytać: ścieżkę pliku wejściowego, ścieżkę zapisu pliku wyjściowego, datę i godzinę początku analizy, ścieżkę pliku referencyjnego z wypisanymi tablicami rejestracyjnymi i wypisane zestawy odczytanych rejestracji. Jeden zestaw odpowiada jednemu

samochodowi, który przejechał przed kamerą. Zawiera on informacje o wszystkich odczytanych rejestracjach, wyznaczonej z nich medianie i tablicy z listy referencyjnej, do której dopasowano odczytany tekst.



Rysunek 13. Przykładowa klatka z filmu wyjściowego

4. Wyniki

Ze względu na dwuetapowe działanie aplikacji tj. początkowa detekcja tablicy i późniejsze wczytanie znalezionego wycinka do silnika OCR, zaprojektowane i przeprowadzone zostały oddzielne testy do oceny działania modelu do detekcji i do ewaluacji działania silnika Tesseract [18] na przygotowanych wycinkach.

4.1. Testy funkcjonalne procesu detekcji tablic rejestracyjnych

Testy zostały przeprowadzone dla określonych treningów (i ich danego punktu kontrolnego, czyli wag z danej epoki), które zostały wybrane spośród wszystkich treningów na podstawie wysokiego parametru mAP. Do skryptu zostały po kolei wczytane 93 zdjęcia testowe, nie widziane wcześniej przez model podczas treningu. Przy różnych ustawieniach progu (*threshold*) została na nich zastosowana operacja detekcji tablicy. W zbiorze zdjęć testowane były 82 obrazy przedstawiające samochód od

przodu z widoczną tablicą rejestracyjną, znajdujący się w różnej odległości od kamery i 11 zdjęć bez widocznej na nich tablicy rejestracyjnej.

Celem przeprowadzonych testów było wybranie najlepszego modelu do detekcji tablic rejestracyjnych spośród wybranych, wytrenowanych wcześniej modeli. Testowanymi miarami były:

- Dokładność, wyrażająca stosunek prawidłowych wyników do całości zbioru danych. Prawidłowymi wynikami są: detekcja tablicy w przypadku zdjęcia z widoczną tablicą rejestracyjną oraz jej brak, w przypadku zdjęcia bez dostrzegalnej tablicy.
- Precyzja, wyznaczająca udział poprawnie pozytywnych prób w zbiorze wszystkich detekcji. Oznacza ona liczbę poprawnie wykrytych tablic rejestracyjnych do wszystkich prostokątnych obwiedni (*bounding-box*), również tych nieprawidłowych.
- Czułość, określająca stosunek poprawnie oznaczonych tablic do zbioru wszystkich zdjęć, które powinny zostać oznaczone (zdjęć z widoczną tablicą rejestracyjną).

Najlepszy model powinien cechować się wysokimi wartościami wszystkich trzech miar. Dla przewidzianego zastosowania, miara czułości ma najmniejsze znaczenie, ze względu na charakterystykę danych. Dane pochodzące z klatek przedstawiających bardziej oddalone samochody, dostarczają mniej informacji niż klatki, na których widoczny samochód jest bliżej kamery. Do prawidłowego działania programu wystarczy niewiele klatek, ale z dobrze widoczną tablicą rejestracyjną.

Warunki przeprowadzania testów:

- W zbiorze danych znajdują się zarówno zdjęcia z dobrze widoczną, białą tablicą rejestracyjną, jak i zdjęcia bez zauważalnej tablicy.
- Określenie kryteriów według których będą oznaczane wyniki.
- Każdy model do detekcji testowany na tym samym zbiorze danych dla trzech różnych wartości progowych.

W tabeli 2 znajdują się informacje o wszystkich wykrytych obiektach zaznaczonych przez prostokątne obwiednie (*bounding-box*) po przeprowadzonych testach na zbiorze 93 zdjęć dla wybranych modeli. Dla ustawionego progu 0.5 oraz 0.2 zdecydowana większość detekcji była trafna. Podczas testowania wystąpiły przypadki, gdy na zdjęciu z widoczną jedną tablicą rejestracyjną program niepoprawnie wykrywał,

zarówno właściwy obiekt (tablicę rejestracyjną) oraz błędny (inny obszar zdjęcia). To tłumaczy rozbieżności liczby wszystkich wykrytych obiektów w stosunku do całkowitej liczby zdjęć testowych.

Tabela 2. Dane z testów modeli do detekcji przy użyciu skryptu inferencyjnego. Oznaczenia w tabeli: TP - klasa prawdziwie pozytywna, FP – klasa fałszywie pozytywna, FN – klasa fałszywie negatywna, TN – klasa prawdziwie negatywna.

Trening (data treningu_ nr epoki)	**Ustawiony próg**											
	0.5				**0.2**				**0.1**			
	TP	FP	FN	TN	TP	FP	FN	TN	TP	FP	FN	TN
06.04_27	32	1	50	12	43	5	37	12	53	12	27	11
22.05_15	26	-	55	12	36	-	45	12	44	-	37	12
23.05_44	29	-	52	12	39	-	42	12	50	1	31	12
24.05_4	29	-	52	12	39	-	42	12	50	1	31	12
29.05_34	28	-	53	12	46	-	35	12	57	9	22	10

W tabeli 3 zostały zebrane wyniki dotyczące dokładności, precyzji i czułości programu, obliczone dla poszczególnych modeli. Najwyższe wartości dokładności zostały osiągnięte przy progu 0.1. Przy ocenie modelu nie należy kierować się jedynie miarą dokładności, trzeba uwzględnić również precyzję i dokładność. Dla testowanych ustawień progu równego 0.1 rzeczywiście wzrosła liczba poprawnie wykrytych tablic rejestracyjnych, ale równocześnie zwiększyła się liczba błędnie zlokalizowanych prostokątnych obwiedni.

W odniesieniu do precyzji, najlepsze wyniki osiągają modele dla testów na ustawionym progu 0.5 i 0.2, gdzie błędne detekcje stanowiły marginalną liczbę. Wraz ze wzrostem niepoprawnie oznaczonych prostokątnych obwiedni (*bounding-box*) miara precyzji spada.

W odniesieniu do czułości, najlepsze wyniki zostały osiągnięte przy progu ustawionym na 0.1. Dzieje się tak ze względu na rosnącą liczbę poprawnie wykrytych tablic, oraz idącą za tym malejącą liczbę niewykrytych tablic.

Tabela 3. Wyniki dokładności / precyzji / czułości dla wybranych treningów i danych progów. Wyniki podane w procentach.

Trening (data treningu_ nr epoki)	**Ustawiony próg**			**Średnia [%]**
	0.5	**0.2**	**0.1**	
06.04_27	46,3/97/39	56,7/89,6/53,8	62,1/81,5/66,3	55/89,4/53
22.05_15	40,9/100/32,1	51,6/100/44,4	60,2/100/54,3	50,9/100/43,6
23.05_44	44,1/100/35,8	54,8/100/48,1	66/98/61,7	55/99,3/48,5
24.05_4	44,1/100/35,8	54,8/100/48,1	66/98/61,7	55/99,3/48,5
29.05_34	43/100/34,6	62,4/100/56,8	68,4/86,4/70,4	57,9/95,5/53,9

Biorąc pod uwagę średnie ze wszystkich obliczonych miar jakości, najlepsze wyniki osiąga model „29.05_34" z dokładnością na poziomie 57,9%, precyzją wynoszącą 95,5% i czułością równą 53,9% dla przekroju ustawionych wartości progowych. Miary dokładności i precyzji są szczególnie ważne przy przewidywanym zastosowaniu. Zadaniem modelu jest poprawna detekcja, szczególnie ważna jest więc niska liczba błędnych klasyfikacji.

4.2 Testy funkcjonalne procesu identyfikacji numerów na tablicach rejestracyjnych pojazdów

Biorąc pod uwagę połączenie modelu do detekcji i silnika OCR w jedną aplikację, do testów optycznego rozpoznawania znaków zostały wykorzystane wycinki tablic rejestracyjnych oznaczone przez model do detekcji, w celu uniknięcia testów na danych niereprezentatywnych (idealnie proste i czytelne zdjęcia tablic rejestracyjnych). Przy testowaniu silnika Tesseract [19] zostało poddane ocenie jak zmiana różnych parametrów w procesie przygotowywania obrazu wpływa na przebieg operacji odczytywania znaków.

Celem przeprowadzonych testów było wybranie najlepszych parametrów wpływających na przebieg wstępnego przetwarzania obrazu poddawanego optycznemu rozpoznaniu znaków. Dla zadania optycznego rozpoznawania tekstu największe znaczenie mają wartości poprawnie odczytane, dlatego miarą na którą należy zwrócić

szczególną uwagę przy interpretacji wyników testów jest dokładność, wybrane parametry powinny się przekładać na wysoką dokładność. Ocena poprawności wyników bazowała na zgodności odczytanego tekstu z listą referencyjną. Testowanymi miarami były:

- Dokładność, oznaczająca liczbę poprawnie odczytanych tablic w stosunku do całego zbioru danych.
- Precyzja, wyznaczająca udział poprawnie odczytanych tablic w zbiorze wszystkich odczytanych sekwencji.
- Czułość, określająca stosunek poprawnie odczytanych tablic do sumy tablic poprawnie odczytanych i nieodczytanych w ogóle.

Warunki wstępne:

- Na wszystkich zdjęciach testowych znajdują się numery rejestracyjne.
- Ze względu na brak 100% dokładności, zgodność odczytanego tekstu z plikiem referencyjnym określana jest dla 4 kolejnych odczytanych znaków.
- Dla pojedynczej grupy testowej zmieniany jest tylko jeden z etapów przetwarzania obrazu.

Warunki przeprowadzania testu:

- Każda grupa testowa testowana na tym samym zbiorze danych dla włączonego i wyłączonego etapu dynamicznego kadrowania.

Tabela 4. Wpływ użycia funkcji skimage.segmentation.clear_border [20] na wartości miar jakości, zależnie od zastosowania operacji dynamicznego kadrowania. Miary jakości podane w procentach.

Nazwa testu	Parametr „clear_border”	Adaptive rescaling: False			Adaptive rescaling: True		
		Dokładność	Precyzja	Czułość	Dokładność	Precyzja	Czułość
Clear_border_0	False	49,06	78,79	57,78	49,06	78,79	57,78
Clear_border_1	True	54,72	76,32	65,91	62,26	70,21	86,84

Testowanym parametrem w tabeli 4 było zastosowanie lub brak zastosowania funkcji, służącej do usuwania pikseli stykających się z granicą wyciętego obrazu. Najlepsze wyniki uzyskano korzystając, zarówno z funkcji clear_border, jak i procesu dynamicznego kadrowania.

Tabela 5. Wpływ użycia funkcji cv2.bilateral_filter() na wartości miar jakości, zależnie od zastosowania operacji dynamicznego kadrowania. Miary jakości podane w procentach.

Nazwa testu	Parametr „bilateral_ filter”	Adaptive rescaling: False			Adaptive rescaling: True		
		Dokładność	Precyzja	Czułość	Dokładność	Precyzja	Czułość
Bilateral_filter_0	5, 55, 55	47,17	69,44	59,52	52,83	62,22	84,85
Bilateral_filter_1	5, 75, 75	49,06	78,79	57,78	60,38	72,73	84,21
Bilateral_filter_2	1, 65, 65	49,06	68,42	65,0	58,49	68,89	88,57
Bilateral_filter_3	3, 65, 65	49,06	68,42	65,0	58,49	68,89	88,57
Bilateral_filter_4	5, 65, 65	54,72	76,32	65,91	62,26	70,21	86,84
Bilateral_filter_5	10, 65, 65	37,74	58,82	51,28	41,51	61,11	61,11

Wartości przedstawione w tabeli 5 określają jaki wpływ na miary jakości mają ustawienia parametrów filtra bilateralnego, który redukuje zaszumienie obrazu przy zachowaniu jego krawędzi. Najlepszą dokładność otrzymano dla testu „Bilateral_filter_4”, przy jednoczesnym uzyskaniu miar precyzji i czułości na wysokim poziomie.

Tabela 6. Wpływ użycia funkcji cv2.adaptive_threshold() na wartości miar jakości, zależnie od zastosowania operacji dynamicznego kadrowania. Miary jakości podane w procentach.

Nazwa testu	Parametr „adaptive_ threshold”	Adaptive rescaling: False			Adaptive rescaling: True		
		Dokładność	Precyzja	Czułość	Dokładność	Precyzja	Czułość
Thresh-old_0	11, 2	28,3	68,18	32,61	60,38	71,11	91,43
Thresh-old_1	11, 11	54,72	76,32	65,91	62,26	70,21	86,84
Thresh-old_2	11, 21	39,62	61,76	53,85	43,4	57,5	74,19

Wartości przedstawione w tabeli 6 określają jaki wpływ na miary jakości mają ustawienia parametrów operacji progowania, która określonym pikselom przypisuje

skrajne wartości w celu zwiększenia kontrastu. Dla trzech przetestowanych par parametrów, najlepsze wyniki osiągnął zestaw o wartości (11, 11).

W każdym z powyższych testów dokładność i czułość osiągały wyższe wartości przy zastosowaniu dynamicznego kadrowania. Wykorzystanie dynamicznego kadrowania zwiększa liczbę odczytanych sekwencji, zarówno poprawnych, jak i błędnych, stąd spadek miary precyzji. Celem testów było znalezienie zestawu parametrów, który wpływa najlepiej na dokładność przy późniejszym odczytywaniu znaków. Ostateczna implementacja wykorzystuje następujące parametry tj.: zastosowanie funkcji clear_border(), zastosowanie dynamicznego kadrowania, cv2.bilateral_filter z parametrem d=5, sigmaColor=65, sigmaSpace=65, cv2.adaptiveThreshold z parametrami: blockSize=11, C=11. Tak dobrane parametry dla przygotowania obrazu pozwalają silnikowi do odczytywania znaków, uzyskać miary jakości na poziomie: dokładność 62,26%, precyzja 70,21% i czułość 86,84%. Miary te nie są między sobą skorelowane.

5. Podsumowanie

Biorąc pod uwagę wyżej opisane wyniki można stwierdzić, że pomimo zadowalającego poziomu dokładności dla zadania detekcji tablic rejestracyjnych, część programu odpowiedzialna za optyczne rozpoznawanie znaków nie osiąga satysfakcjonujących wyników. Jest to spowodowane wieloma ograniczeniami, w których było realizowane badanie. Ze względu na ograniczoną pamięć i moc obliczeniową dostępnych zasobów nie było można zaprojektować aplikacji opartej na najnowszych rozwiązaniach do odczytywania tekstu, lub w przypadku wykorzystania sieci EfficientDet, nie było można wykorzystać pełnego potencjału wybranego rozwiązania (skalowanie przy użyciu parametru ϕ). Zastosowanie sieci neuronowych do rozpoznawania znaków oraz użycie architektury EfficientDet wyższego poziomu powinno znacząco wpłynąć na poprawę działania aplikacji.

Kolejnym czynnikiem wpływającym na jakość działania wytworzonego oprogramowania był zbiór danych. Rozdzielczość oraz liczba danych treningowych i testowych jest czynnikiem ograniczającym. Polepszenie jakości uzyskanych danych, w związku z postępem technik wykorzystywanych w monitoringu, oraz zwiększenie zbioru

danych mogłoby się przyczynić do wytrenowania lepszego modelu do detekcji, jak i precyzyjniejszego działania narzędzia do OCR.

W obecnej formie wytworzone oprogramowanie spełnia funkcję wstępnego analizatora danych, które później muszą zostać zweryfikowane przez uprawnioną osobę. Po wprowadzeniu proponowanych wyżej usprawnień oprogramowanie powinno być w stanie przeprowadzić samodzielną, precyzyjną analizę.

Literatura

[1] S. Albawi, T.A. Mohammed, S. Al-Zawi, Understanding of a convolutional neural network, 2017, *2017 International Conference on Engineering and Technology (ICET)*, DOI: 10.1109/ICEngTechnol.2017.8308186.

[2] S.M.Silva, C.R. Jung, License Plate Detection and Recognition in Un-constrained Scenarios. [w:] *Computer Vision – ECCV 2018. ECCV 2018. Lecture Notes in Computer Science*, Vol. 11216, (Eds.) Ferrari V., Hebert M., Sminchisescu C., Weiss Y. Springer, Cham, 2018.

[3] J. Redmon, A. Farhadi, *YOLO9000: Better, Faster, Stronger*, arXiv:1612.08242.

[4] *Amazon Rekognition pricing*, https://aws.amazon.com/recognition/ [31.10.2021].

[5] R. Laroca, L.A. Zanlorensi, G.R. Gonçalves, E. Todt, W.R. Schwartz, D. Menotti, An Efficient and Layout-Independent Automatic License Plate Recognition System Based on the YOLO detector. *IET Intelligent Transport Systems* 2021, Vol. 15, No. 4, DOI: 10.1049/itr2.12030.

[6] M.J. Shafiee, B. Chywl, F. Li, A. Wong, *Fast YOLO: A Fast You Only Look Once System for Real-time Embedded Object Detection in Video*, 2017, arXiv:1709.05943.

[7] Li, Y., Wan, J., Miao, Q. et al. CR-Net: A Deep Classification-Regression Network for Multimodal Apparent Personality Analysis. *International Journal of Computer Vision* 2020, Vol. 128. https://doi.org/10.1007/s11263-020-01309-y.

[8] I. Goodfellow, Y. Bengio, A. Courville, *Deep Learning*, 2016, MIT Press, Cambridge.

[9] M. Tan, R. Pang, Q.V. Le, EfficientDet: Scalable and Efficient Object Detection, *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (2020)*, arXiv:1911.09070.

[10] M. Tan, Q.V. Le, EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks, *International Conference on Machine Learning*, 2019, arXiv:1905.11946.

[11] Strona internetowa ImageNet, https://www.image-net.org/ [31.10.2021].

[12] Strona internetowa Project Jupyter, https://jupyter.org/ [31.10.2021].

[13] Strona internetowa TensorBoard, https://www.tensorflow.org/tensorboard/ [31.10.2021].

[14] A.G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, H. Adam: *MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications*. 2017, CoRR abs/1704.04861.

[15] Strona internetowa Common Objects in Context, https://cocodataset.org/ [31.10.2021].

[16] Xuannianz (2020) EfficientDet [kod źródłowy], https://github.com/xuannianz/EfficientDet.

[17] Project description, 2021, PyPI.org, https://pypi.org/project/pytesseract/ [Data uzyskania dostępu: 31.10.2021].

[18] R. Smith, An overview of the Tesseract OCR Engine, 2007, *ICDAR 2007, 9th International Conference on Document Analysis and Recognition*, Curitiba.

[19] Image processing in Python, scikit-image image processing in python, 2020, https://scikit-image.org/ [31.10.2021].

[20] Strona internetowa OpenCV, http://opencv.org [31.10.2021].

[21] M. Hayashida, H.Koyano, Finding Median and Center Strings for a Probability Distribution on a Set of Strings Under Levenshtein Distance Based on Integer Linear Programming. [w:] *Biomedical Engineering Systems and Technologies – BIOSTEC 2016*, (Eds.) A. Fred, H. Gamboa, 2017, Springer, Cham. https://doi.org/10.1007/978-3-319-54717-6_7.

A system for the automatic license plate detection and number identification using neural networks

Abstract

The paper describes the stages of creating an application for automatic license plate recognition and vehicle number identification using neural networks and the Deep Learning approach, starting from data preparation for training and ending with performance testing. The EfficientDet network model was used to implement the license plate detection mechanism and the Pytesseract and OpenCV libraries for the optical recognition of identification numbers on the detected plates. The verification of the effectiveness of the application concerned the assessment of the correctness of the detection of license plates in vehicles and the recognition of identification numbers. The basic evaluation criteria were the measures commonly used in such applications: accuracy, precision and sensitivity. The conducted tests allowed to determine the optimal parameter values for the license plate detection process (hyperparameter values, i.e. network size, number of steps, number of epochs, weights, learning coefficient, size of the data batch during training and the threshold value during operation verification) as well as for the number recognition process (keeping or removing pixels touching the border of the cut image, values of bilateral filter parameters, thresholding operations).

***Keywords**:* Neural networks, computer vision, license plate recognition, EfficientDet, Pytesseract, ALPR.