

WOJSKOWA AKADEMIA TECHNICZNA
im. Jarosława Dąbrowskiego
WYDZIAŁ MECHATRONIKI, UZBROJENIA I LOTNICTWA



PRACA DYPLOMOWA
STUDIA PIERWSZEGO STOPNIA

Piotr Badełek

(stopień, imiona i nazwisko studenta)

***Wykorzystanie sieci neuronowej do analizy obrazu w celu
detekcji pieszego na jezdni***

***Using a neural network for image analysis to detect
pedestrian on the road***

(temat pracy dyplomowej w języku polskim i języku angielskim)

*Mechatronika, Robotyka i automatyka przemysłowa
(kierunek i specjalność studiów)*

dr inż. Waldemar Śmietański

(stopień wojskowy, tytuł/stopień naukowy, imię i nazwisko promotora pracy dyplomowej)

Warszawa – 2022 r.

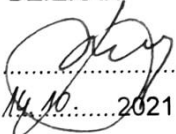
(strona celowo zostawiona pusta)

WOJSKOWA AKADEMIA TECHNICZNA

im. Jarosława Dąbrowskiego

WYDZIAŁ MECHATRONIKI, UZBROJENIA I LOTNICTWA
INSTYTUT TECHNIKI RAKIETOWEJ I MECHATRONIKI**„ZATWIERDZAM”**

DZIEKAN


Dnia 14.10.2021 r.**ZADANIE**
do pracy dyplomowejWydane studentowi: **Piotrowi BADEŁKOWI**

studia pierwszego stopnia

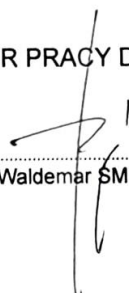
Forma studiów: stacjonarne

I. Temat pracy**Wykorzystanie sieci neuronowej do analizy obrazu
w celu detekcji pieszego na jezdni****Using a neural network for image analysis to detect pedestrians on the road****II. Forma pracy: praca projektowo-aplikacyjna****III. Treść zadania:**


1. Przedstawić przegląd i krytyczną analizę współcześnie dostępnych narzędzi do analizy obrazu z wykorzystaniem sieci neuronowej.
2. Przedyskutować i zaproponować własności funkcjonalne i interfejs użytkownika aplikacji do detekcji pieszego na jezdni.
3. Zgromadzić bazy danych obrazowych do uczenia sieci neuronowej.
4. Zaprojektować i uruchomić proponowaną aplikację do analizy obrazu.
5. Zaproponować metodykę i przebadать skuteczności detekcji pieszego na jezdni przez opracowaną aplikację.

- IV. W rezultacie wykonania pracy dyplomowej należy przedstawić:
- a) notatkę zgodną z „Zasadami redagowania oraz wymaganiami wydawniczymi stawianymi pracy dyplomowej”,
 - b) rysunki:
 - 1. Interfejs użytkownika opracowanej aplikacji.
 - 2. Algorytmy opracowanego oprogramowania.
- V. Termin złożenia pracy dyplomowej w dziekanacie: 2 lutego 2022 r.
- VI. Data wydania zadania do pracy dyplomowej: 15 października 2021 r.

PROMOTOR PRACY DYPLOMOWEJ


.....
dr inż. Waldemar SMIETAŃSKI

DYREKTOR
INSTYTUTU


.....
dr hab. inż. Leszek BARANOWSKI prof. WAT

Zadanie otrzymałem 15.10.2021
Piotr Badetek

Spis treści

WSTĘP.....	7
1. Sztuczne sieci neuronowe	11
1.1 Konwolucyjne sieci neuronowe	13
1.2 Proces uczenia sieci neuronowej.....	14
2. Współczesne narzędzia do analizy obrazów	17
2.1 Biblioteka OpenCV	17
2.2 Biblioteka TensorFlow	18
2.3 Biblioteka PyTorch	18
2.4 Zbiór danych Coco Dataset	19
2.5 Środowisko Google Colab	20
2.6 Algorytm YOLO	21
2.7 Algorytm SSD	22
2.8 Algorytm EfficientDet.....	24
2.9 Algorytm Faster R-CNN	25
3. Własności funkcjonalne aplikacji do detekcji pieszego	27
3.1 Interfejs użytkownika aplikacji	27
3.2 Algorytm działania programu detekcji.....	29
4. Zbiór danych obrazowych.....	31
4.1 Problematyka tworzenia zbioru danych	31
4.2 Zgromadzenie danych obrazowych.....	32
4.3 Etykietowanie danych obrazowych.....	34
4.4 Zbiór danych testowych i zbiór danych treningowych	36
5. Uczenie modelu sieci neuronowej.....	39
5.1 Konwersja danych do plików CSV	39
5.2 Utworzenie plików TfRecord oraz mapy etykiet	40
5.3 Konfiguracja modelu sieci neuronowej.....	41
5.4 Uczenie modelu w środowisku Google Colab	42
6. Uruchomienie opracowanej aplikacji.....	45
7. Testy skuteczności detekcji	49
7.1 Metodologia testów	49
7.2 Wyniki testów	51
7.3 Porównanie uzyskanych wyników z modelem YOLO	56
Podsumowanie	59
Bibliografia.....	61

(strona celowo zostawiona pusta)

WSTĘP

Od początku moich studiów inżynierskich interesowałem się programowaniem. W szczególności zainteresowały mnie zastosowania sztucznej inteligencji w systemach bezpieczeństwa. Systemy te zaliczają obecnie dynamiczny rozwój oraz pozostawiają szerokie pole do dalszych badań. Podczas lektury anglojęzycznych źródeł zauważyłem, że systemy bezpieczeństwa oparte na wykorzystaniu sztucznych sieci neuronowych rewolucjonizują rynek, stając się przy tym coraz bardziej przystępne dla młodych inżynierów oprogramowania. Poza studiami pracuję jako programista w języku Python, dlatego przy wyborze tematu pracy dyplomowej starałem się połączyć moje zainteresowania oraz nabyte umiejętności. Zdecydowałem się więc na opracowanie własnej aplikacji, korzystając przy tym z najnowszych oraz najczęściej wykorzystywanych rozwiązań na moment pisanie tej pracy.

Słowami wstępu, systemy analizy obrazów stają się obecnie bardzo popularne ze względu na szeroką dostępność do dużych mocy obliczeniowych, rosnące pojemności urządzeń pamięciowych oraz szybki rozwój oprogramowań graficznych. Znalazły one zastosowania między innymi w robotyce, autonomicznych systemach jazdy lub nadzorze video. Celem dziedziny analizy obrazów jest wykonywanie pewnych operacji w celu wydobycia z nich pożądaných informacji. W technice analizy obrazów wejście stanowi obraz lub związane z nim cechy. Dziedzina analizy obrazów dzieli się na wiele kategorii, jednak najpopularniejsze z nich to klasyfikacja obrazów oraz detekcja obiektów.

Klasyfikacja obrazów polega na identyfikacji obrazów i kategoryzowaniu ich w jednej z kilku predefiniowanych, odrębnych klas. Oprogramowania i aplikacje do rozpoznawania obrazów mogą definiować to, co jest przedstawione na obrazie, a następnie odróżniać jeden obiekt od drugiego. Klasyfikacja obrazów jest podstawowym zadaniem, które ma na celu zidentyfikowanie obrazu jako całości. Metoda ta jest zazwyczaj używana w przypadku obrazów, na których pojawia się i jest analizowany tylko jeden obiekt. Początkowo klasyfikacja obrazów opierała się na nieprzetworzonych danych pikselowych. Oznaczało to, że komputery rozbiły obrazy na pojedyncze zestawy pikseli. Metody te jednak były zawodne, ponieważ zdjęcia tej samej rzeczy mogą wyglądać zupełnie inaczej. Różnice mogą pojawiać się w: oświetleniu, kolorach tła, kątach widzenia kamery, ostrościach kamery i pozycjach umiejscowienia analizowanych obiektów.

Zastosowana w niniejszej pracy detekcja obiektów jest zadaniem trudniejszym, ponieważ służy do lokalizowania wystąpień obiektów na obrazach oraz filmach wraz z ich przestrzennym położeniem, zaznaczanym za pomocą prostokątnych ramek ograniczających.

Detekcja obiektów jest najczęściej wykorzystywana w zadaniach, w których zależy nam na jednoczesnej detekcji wielu obiektów różnych klas. Aby uzyskiwać wysokie skuteczności poprawnego wykrywania obiektów, wciąż poszukuje się oraz rozwija algorytmy uczenia głębokiego. Wykrywanie obiektów jest używane między innymi w zaawansowanych systemach wspomagania kierowcy. Umożliwia ono wykrywanie pasów ruchu lub obiektów na drodze. Rozwiązania wykorzystujące wykrywanie obiektów znaleźć można także w monitoringu wideo, rozpoznawaniu twarzy w smartfonach lub systemach wyszukiwania obrazów. Klasy obiektów mają swoje specjalne cechy, które są przetwarzane przez konwolucyjne sieci neuronowe. Na przykład, podczas szukania okręgów potrzebne są obiekty znajdujące się w określonej odległości od punktu zdefiniowanego jako środek okręgu, a podczas szukania kwadratów potrzebne są obiekty o rogach prostokątnych oraz o równych długościach boków. Podobne podejście można spotkać w przypadku identyfikacji twarzy, w których wykrywa się położenie oczu, nosa, ust oraz wyróżnia się cechy szczególne, jak kolor skóry i odległość między oczami.

W zadaniach powiązanych z detekcją obiektów przodują głębokie sieci neuronowe, które w ostatnich latach uległy dynamicznemu rozwojowi. Jedną z odmian głębokich sieci neuronowych są użyte w pracy splotowe sieci neuronowe, charakteryzujące się w podstawowej strukturze: warstwą splotową, warstwą aktywacji oraz warstwą redukującą rozmiar. Na wejście, splotowe sieci neuronowe przyjmują obraz, który jest przetwarzany przez kolejne warstwy. Ostatnia warstwa sieci splotowej daje wyjście trafiające na klasyczną sieć neuronową.

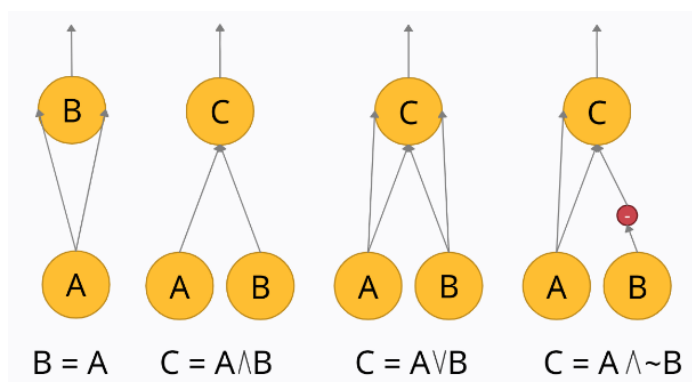
Celem niniejszej pracy było opracowanie aplikacji do detekcji pieszego na jezdni, a następnie zbadanie skuteczności jej działania. Pierwszy rozdział poświęciłem wprowadzeniu w zagadnienie sztucznych sieci neuronowych oraz procesowi ich uczenia. W rozdziale drugim dokonałem przeglądu i analizy współcześnie dostępnych narzędzi do analizy obrazu z wykorzystaniem sieci neuronowej. Kolejny rozdział poświęciłem opisowi własności funkcjonalnych aplikacji, algorytmowi działania programu detekcji oraz algorytmowi działania interfejsu użytkownika. W rozdziale czwartym przedstawiłem opracowanie autorskiego zbioru danych obrazowych, który następnie przygotowałem w celu uczenia modelu sieci neuronowej. Piąty rozdział poświęciłem procesowi uczenia modeli sieci neuronowych. W pracy użyłem metody transferowego uczenia sieci neuronowej, charakteryzującej się zastosowaniem sieci wstępnie przeszkolonej na dużym zestawie danych, a następnie dostosowaniem jej do określonego zadania. Uczenie transferowe przydatne jest w momencie, gdy nie istnieje konieczność nauki złożonego modelu sieci neuronowej na

dużym zestawie danych od podstaw. Zakres uczenia sieci neuronowej obejmował pracę na trzech różnych, konkurencyjnych modelach o otwartym kodzie źródłowym, opracowanych przez zespół TensorFlow. W następnym rozdziale opisano uruchomienie aplikacji oraz jej działanie. Ostatni z rozdziałów poświęciłem na testy skuteczności detekcji. Podczas przeprowadzonych badań porównałem ze sobą wartości prawdopodobieństw detekcji pieszego z wykorzystaniem różnych modeli sieci neuronowych. Otrzymane wyniki zebrałem oraz dokonałem wyboru optymalnego modelu spośród przedstawionych rozwiązań.

(strona celowo zostawiona pusta)

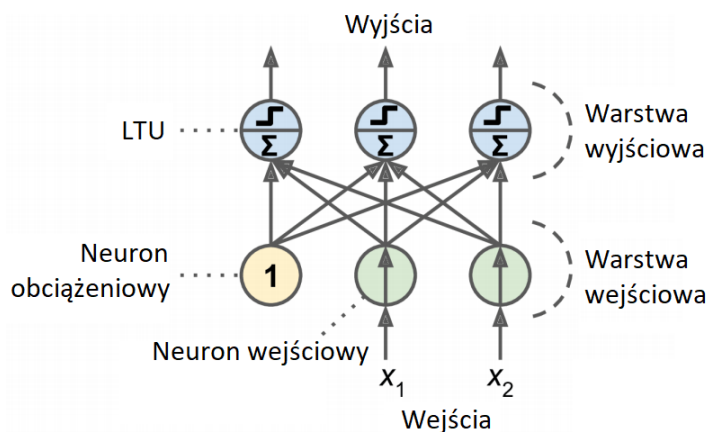
1. Sztuczne sieci neuronowe

Badacze Warren McCulloch i Walter Pitts w swoim artykule z 1943 roku zaprezentowali prosty model biologicznego neuronu, nazywany obecnie sztucznym neuronem. Postać pojedynczego neuronu ma co najmniej jedno wejście i jedno binarne wyjście. Wyjście jest aktywne tylko wtedy, kiedy aktywna jest określona liczba wejść. W ich pracy udowodnione zostało, iż przy połączeniu tak uproszczonych modeli sztucznych neuronów w sieć neuronową, możliwe jest rozwiązanie dowolnego zadania logicznego [1]. Proste operacje, które mogą być następnie łączone ze sobą w celu przeprowadzania bardziej złożonych operacji logicznych, przedstawiono na rysunku nr 1.



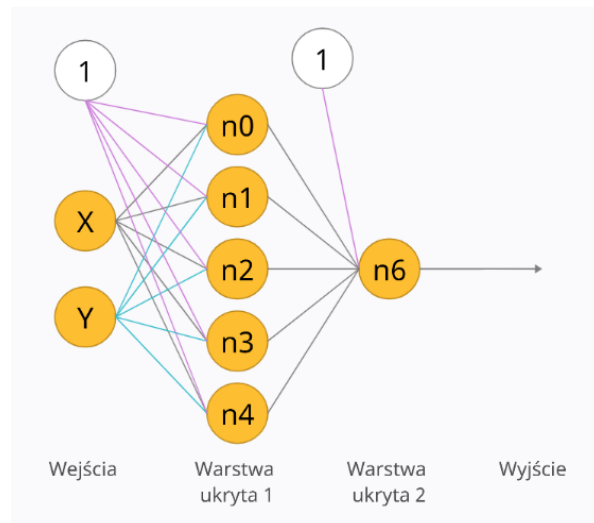
Rysunek 1 Proste operacje logiczne przeprowadzane przez sieci neuronowe. Źródło: Opracowanie własne

W połowie lat 90 rosnąca społeczność zajmująca się pracami nad sztuczną inteligencją kontynuowała rozwijanie architektur sztucznych sieci neuronowych. W roku 1957 został zaprezentowany przez Franka Rosenblatta model perceptronu. Podstawą jego działania jest zmodyfikowany sztuczny neuron, zwany również jednostką liniową z progiem. Za wejścia oraz wyjścia odpowiadają liczby, natomiast każde z połączeń posiada zdefiniowaną wagę. Zadaniem jednostki liniowej z progiem jest wyliczanie ważonej sumy sygnałów, tak aby możliwe było użycie funkcji skokowej, która daje ostateczny wynik. W perceptronie każdy neuron jest połączony ze wszystkimi wejściami. W jego budowie wyróżniamy również neuron obciążeniowy, który wysyła na wyjście wartość równą 1. Model perceptronu zaprezentowany został na rysunku nr 2.



Rysunek 2 Diagram perceptronu. Źródło: [2]

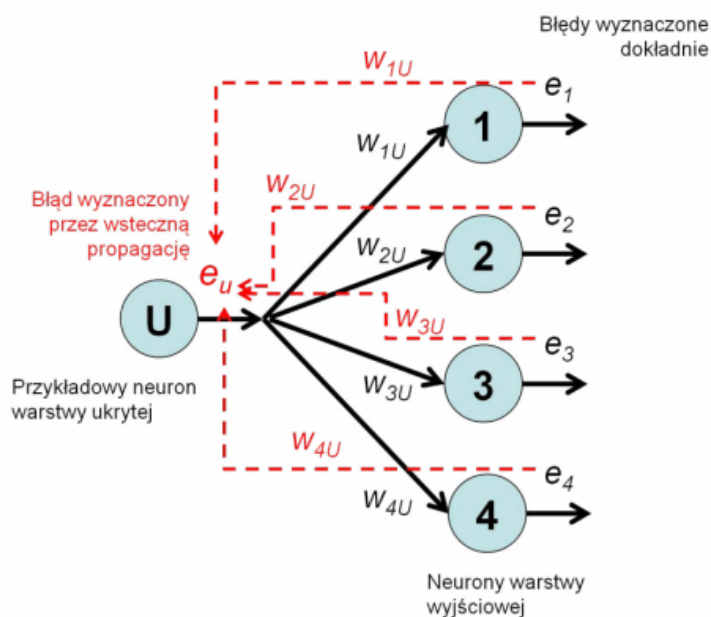
W rzeczywistości jednak pozornie proste zadania mają charakter nieliniowy, których nie jest w stanie rozwiązać zaprezentowany powyżej diagram perceptronu [2]. W związku z tym stworzono perceptrony wielowarstwowe, które składają się z warstwy wejściowej, co najmniej jednej warstwy ukrytej oraz warstwy wyjściowej. Każda z warstw posiada neuron obciążający oraz połączona jest z kolejną warstwą. Głęboka sieć neuronowa występuje, kiedy posiada przynajmniej dwie warstwy ukryte [3]. Diagram głębokiej sieci neuronowej został przedstawiony na rysunku nr 3.



Rysunek 3 Diagram głębokiej sieci neuronowej. Źródło: Opracowanie własne.

Możliwość uczenia perceptronów wielowarstwowych nastąpiła wraz z pojawieniem się algorytmu wstecznej propagacji. Algorytm ten poprawia wartości wag podczas uczenia sieci, oceniając błąd popełniany przez każdy neuron warstw ukrytych. Wartość błędu rozważanego neuronu jest obliczana na podstawie wartości błędów wszystkich tych neuronów, do których wysłał on wartość swojego sygnału wyjściowego jako składnika ich danych wejściowych. Procedurę tą powtarza się, przyjmując wyliczone wartości błędów jako znane, a następnie

przeprowadzając propagację do kolejnej warstwy ukrytej znajdującej się bliżej wejścia. Tak skonstruowany odwrotny przebieg jest w stanie mierzyć gradient błędu we wszystkich wagach połączeń poprzez propagację tego gradientu w kierunku początku sieci. Błąd popełniany w neuronach warstwy wyjściowej wyznacza się natomiast na podstawie danych wyjściowych i odpowiedzi wzorcowych zawartych w zbiorze uczącym. Jako ostatni etap działania algorytmu wstecznej propagacji przyjmuje się mechanizm działania gradientu prostego wobec wszystkich wag połączeń w sieci, używając przy tym obliczonego wcześniej gradientu błędu. W celu poprawnego działania zaprezentowanego algorytmu wstecznej propagacji, należy zrezygnować z używania funkcji skokowej, gdyż jej konstrukcja składająca się z płaskich segmentów, uniemożliwia korzystanie z gradientu [3]. Wśród popularnych funkcji aktywacji używanych z opisywanym algorytmem, wyróżnić można funkcję logistyczną, funkcję tangensa hiperbolicznego lub rektyfikowaną jednostkę liniową. Diagram algorytmu wstecznej propagacji został przedstawiony na rysunku nr 4.



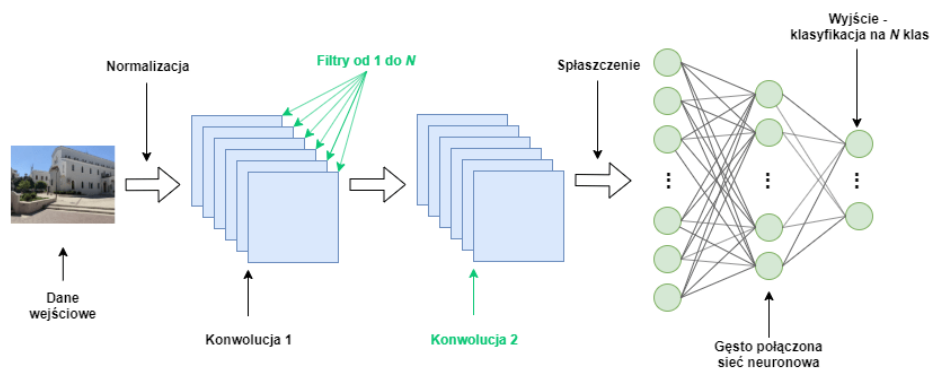
Rysunek 4 Diagram działania algorytmu wstecznej propagacji. Źródło: [4]

1.1 Konwolucyjne sieci neuronowe

Konwolucyjne sieci neuronowe powstały w wyniku inspiracji budową kory mózgowej, która odpowiedzialna jest za odbiór bodźców wzrokowych. Sieci konwolucyjne stoją u podstaw obecnej dynamiki rozwoju dziedziny sztucznej inteligencji, dlatego w tym podrozdziale krótko opisano ich topologię.

Architektura konwolutyjnych sieci neuronowych, zwanych inaczej splotowymi, składa się z wielu warstw o coraz mniejszej liczbie węzłów, gdzie wyjście jednej warstwy połączone jest z wejściem kolejnej. Jednak to, co odróżnia ją od standardowej sieci neuronowej, to fakt, iż wszystkie neurony tworzące sieć są identyczne. Ponadto, mają one takie same parametry oraz wartości wag. Objawia się to w wydajności działania sieci, poprzez zmniejszenie liczby parametrów przez nią kontrolowanych. Zaletą jest również uproszczenie procesu uczenia takich sieci, poprzez zmniejszenie liczby wolnych parametrów, dla których istnieje potrzeba ciągłych obliczeń.

Podsieć konwolutyjna, występująca po procesie normalizacji danych wejściowych, wykorzystuje zazwyczaj dane trójwymiarowe, które są nieprzetworzone. Do działania sieci neuronowej potrzeba natomiast danych spłaszczonych do jednego wymiaru. Diagram konwolutyjnej sieci neuronowej przedstawiono na rysunku nr 5.



Rysunek 5 Diagram konwolutyjnej sieci neuronowej. Źródło: [5]

Podsieć neuronowa, tak samo, jak podsieć konwolutyjna może być wielowarstwowa. Dzięki temu możliwe jest znajdowanie kolejnych własności z obrazów. Każda warstwa konwolucji jest wielowymiarowa, ponieważ definiuje się dla niej N filtrów. Algorytm propagacji wstecznej, którego diagram przedstawiony został na Rys. 4, będzie obniżał istotność filtrów nieskutecznych, a promował filtry wspomagające prawidłową klasyfikację. Dzięki temu po wielu iteracjach możliwe jest otrzymanie skutecznie działających filtrów.

1.2 Proces uczenia sieci neuronowej

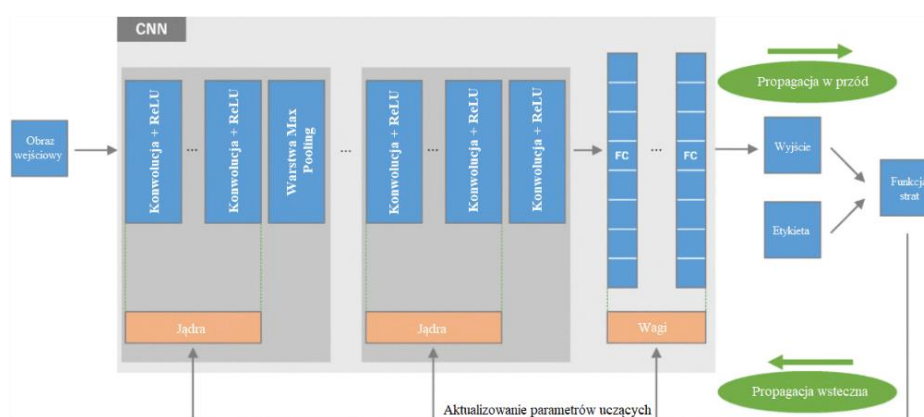
Sieci neuronowe zaliczają się do systemów inteligentnych, ponieważ posiadają zdolności do tworzenia reguł wiążących ze sobą oddzielne elementy systemu oraz zdolności

do rozpoznawania obiektów na podstawie niepełnych informacji. Tematyka uczenia sieci neuronowych to zagadnienie złożone, ponieważ nie przebiega ono według jednej, określonej strategii. Zagadnieniu temu bliżej do postępowania metodą prób i błędów. Wyróżnić można jednak schemat, który występuje wśród najpopularniejszych modeli.

Modyfikacja parametrów sieci, zwanych inaczej wagami, umożliwia przeprowadzanie procesu uczenia sieci neuronowych. Podczas nauki sieci neuronowych nie projektuje się ponownie algorytmu przetwarzania danych wejściowych, lecz stawia się sieci przykładowe zadanie. Proces uczenia konwolucyjnych sieci neuronowych jest prostszy niż proces uczenia standardowej sieci neuronowej, ponieważ zawiera ona mniej parametrów z dokładnością do liczby warstw konwolucyjnych oraz ich rozmiaru [6].

W sieciach konwolucyjnych najczęściej stosuje się warstwę MaxPool, która przesuwa filtry do następnej mapy. Warstwa ta zmniejsza obraz, zmniejszając tym samym złożoność obliczeniową sieci. Proces ten służy uchronieniu sieci przed przeuczeniem, czyli zapamiętaniu przez sieć przypadkowych błędnych danych. Przeuczone modele nie potrafią dopasować posiadanej wiedzy do nowych obserwacji [7].

Warstwa połączeń każdy-z-każdym (FC) oblicza wynik klasyfikacji. Połączona ona jest z każdą poprzednią warstwą. Wydajność modelu sieci konwolucyjnej dla poszczególnych jąder oraz wag obliczana jest za pomocą funkcji strat poprzez propagację wprzód na zbiorze danych treningowych, natomiast parametry uczące są aktualizowane poprzez wsteczną propagację z algorytmem optymalizacji gradientu. Opisany schemat uczenia konwolucyjnej sieci neuronowej zaprezentowano na rysunku 6.



Rysunek 6 Schemat proces uczenia konwolucyjnej sieci neuronowej. Źródło: [8]

Proces uczenia powtarza się dla każdej partii szkoleniowej, dopóki nie obejmie on wszystkich próbek w zbiorze danych. Okres czasu, w którym został objęty cały zestaw szkoleniowy nazywa się epoką.

(strona celowo zostawiona pusta)

2. Współczesne narzędzia do analizy obrazów

Wiele obecnie wykonywanych przez aplikacje do analizy obrazów zadań wymusza wyjście poza proste algorytmy sztucznej inteligencji. Chcąc osiągnąć wysoką skuteczność i szybkość wykrywania obiektów na obrazach, potrzeba zbudować modele uczenia głębokiego do przetwarzania obrazów. Aby uprościć prace programistom, stosuje się specjalne platformy oraz biblioteki. Poniżej zostanie przedstawione kilka z nich, które obecnie są jednymi z najczęściej wykorzystywanych.

2.1 Biblioteka OpenCV

Biblioteka OpenCV to bezpłatna biblioteka wykorzystywana do obróbki obrazu. Jej pełna nazwa to Open Source Computer Vision. Została ona utworzona w 1999 roku w firmie Intel przez Gary'ego Bradsky'ego. Jest ona wieloplatformowa i można z niej korzystać na systemach takich jak Mac OS X, Windows, Android oraz Linux. Pomimo że została stworzona w języku C, nie ma żadnych problemów w korzystaniu z niej za pomocą nakładek w językach takich jak C++, C#, Python, Java oraz Node.js. Biblioteka jest w stanie obsługiwać wiele algorytmów związanych z wizją komputerową oraz uczeniem maszynowym. Jest ona również dobrze udokumentowana oraz stale aktualizowana. Obecnie autorzy skupiają się głównie na przetwarzaniu obrazu w czasie rzeczywistym [9].

Istnieje wersja OpenCV-Python, która jest opakowaniem języka Python dla oryginalnej implementacji OpenCV C++. W porównaniu do języków takich jak C oraz C++, język Python jest wolniejszy. Przy wykorzystaniu tej wersji, jest to działający w tle kod języka C++. Daje to korzyść w postaci zachowania tej samej prędkości oraz łatwości programowania w języku Python. Nakładka OpenCV-Python działa na bazie popularnej biblioteki Numpy, która jest wysoce zoptymalizowana do wykonywania operacji numerycznych, posiadając składnię w stylu Matlab'a. Wszystkie struktury tablic są konwertowane do tablic Numpy. Ułatwia to również integrację z bibliotekami takimi jak SciPy oraz Matplotlib.

Jedną z zalet biblioteki OpenCV jest stosowanie jej w systemie Windows w bibliotekach połączonych dynamicznie. Cała zawartość biblioteki jest wówczas ładowana tylko w czasie jej wykonywania, a niezliczone programy mogą używać tego samego pliku biblioteki. Oznacza to, że jeżeli posiadamy dziesięć aplikacji korzystających z biblioteki OpenCV, nie ma potrzeby utworzenia wersji dla każdej z nich.

Biblioteka OpenCV jest powszechnie wykorzystywana w projektach komercyjnych przez firmę Google. Implementacje jej funkcji znajdziemy w mapowaniu ulic lub kalibracji kamery w aplikacjach takich jak Google Maps lub Google Street's View.

2.2 Biblioteka TensorFlow

Biblioteka TensorFlow została utworzona w roku 2011 przez Google Brain jako projekt badawczy, który z biegiem lat zaczął cieszyć się dużą popularnością w grupie Alphabet. Społeczność specjalizująca się w uczeniu maszynowym doceniła fakt wysoce elastycznej architektury mogącej wykorzystywać różne rodzaje jednostek przetwarzających, takich jak centralne jednostki obliczeniowe, procesory graficzne i specjalizowane układy scalone.

Wersja 2.0 wydana we wrześniu roku 2019 stanowi ważny krok w rozwoju biblioteki. Przez ostatnie kilka lat jedną z głównych jej słabości był skomplikowany interfejs programowania aplikacji. Przykładowo, definiowanie głębokich sieci neuronowych wymagało więcej pracy niż w przypadku korzystania z konkurencyjnych bibliotek [10]. Doprowadziło to do powstania kilku interfejsów wysokiego poziomu, takich jak TensorFlow Slim i Keras.

Jednym z najpopularniejszych interfejsów do programowania aplikacji z TensorFlow jest Python. Początkującym zaleca się używanie sekwencyjnego interfejsu programowania aplikacji o nazwie Keras.

Istnieje rozszerzenie biblioteki o nazwie TensorFlow2 Object Detection API, umożliwiające trenowanie najnowocześniejszych modeli w ramach ujednoliconej struktury, w tym modeli firmy Google Brain. Zaletą interfejsu TensorFlow2 Object Detection jest szeroka gama różnych modeli i strategii pomagających osiągnąć wysoką efektywność rozpoznawania obrazów. Zapewnienie szerokiego zestawu modeli umożliwia wdrożenie ich w niestandardowym zestawie danych. W tym celu TensorFlow2 oferuje również rozwiązania wdrożeniowe, udostępniając na stronie projektu skrypty eksportowe [11].

2.3 Biblioteka PyTorch

PyTorch to biblioteka uczenia maszynowego przeznaczona dla języka Python. Została ona opracowana początkowo przez grupę badawczą zajmującą się sztuczną inteligencją firmy Facebook oraz grupę zajmującą się oprogramowaniem Pyro firmy Uber.

Podczas tworzenia biblioteki PyTorch, celem było osiągnięcie jak największego podobieństwa do biblioteki Python Numpy. Argumentem w tych działaniach była między innymi potrzeba tworzenia tensorów, czyli podstawowych bloków konstrukcyjnych występujących w każdej bibliotece uczenia głębokiego.

Tensory to macierzowe struktury danych, które właściwościami oraz funkcjonalnością są podobne do tablic biblioteki Numpy. Najważniejsza różnica polega jednak na implementacji tensorów w nowoczesnych bibliotekach, które mogą działać bardzo szybko na procesorach graficznych. Jest to bardzo istotne, ponieważ przyspiesza obliczenia numeryczne, co w efekcie może zwiększyć prędkość sieci o 50 razy lub więcej [12]. Kolejnymi elementami charakterystycznymi są zmienne węzły w grafie obliczeniowym, które przechowują dane obliczeniowe oraz wartości gradientu. Wyróżniamy również moduły warstwy sieci neuronowej, które przechowują wagi stanu oraz umożliwiają proces uczenia.

Wszystkie te działania pozwoliły na wysoce efektywną interakcję pomiędzy zwykłym kodem Pythona, Numpy i PyTorch, umożliwiając tym samym szybsze i łatwiejsze kodowanie. Jednym z udogodnień tej biblioteki jest definiowanie sieci neuronowych jako klas języka Python. Dzięki temu interfejs biblioteki PyTorch jest przejrzysty i prosty do nauki. Było to jednym z powodów, dla których przyciągał on początkowo wielu programistów, skarżących się na skomplikowanie opisywanej w powyższym podrozdziale biblioteki TensorFlow.

Biblioteka posiada również platformę do generowania dynamicznych grafów obliczeniowych. Dzięki niej programiści mogą mieć bieżący podgląd na ilość pamięci potrzebnej do utworzenia modelu sieci neuronowej. W przeciwieństwie do bibliotek takich jak TensorFlow, w których przed uruchomieniem modelu należy najpierw zdefiniować cały wykres obliczeniowy, biblioteka PyTorch umożliwia dynamiczne definiowanie wykresu [13].

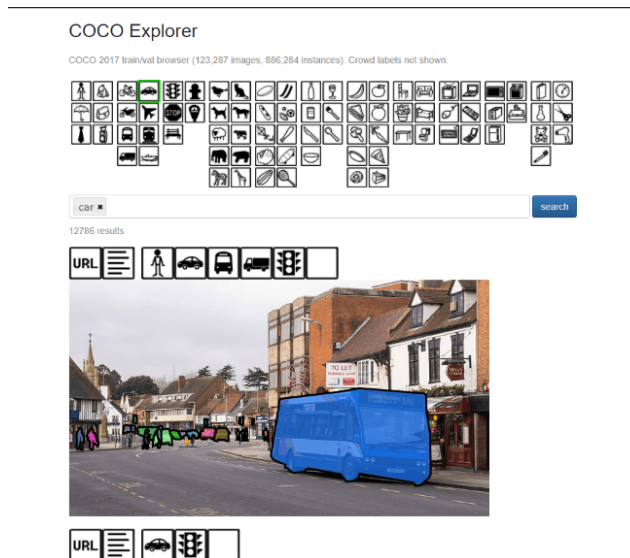
2.4 Zbiór danych Coco Dataset

Coco Dataset to wielkoskalowy zbiór danych firmy Microsoft, którego rozwinięcie brzmi Common Objects in Context. Dotyczy on wykrywania obiektów na dużą skalę, segmentacji obrazów oraz wykrywania kluczowych punktów obrazów. Jego pierwsza wersja została wydana w roku 2014. Zawierała ona wówczas 164 tysiące obrazów podzielonych na zestawy uczące, walidacyjne oraz testowe [15]. Po roku 2015 zostały udostępnione dodatkowe zestawy nowych obrazów. Obecnie, jest on jednym z najpopularniejszych zbiorów danych do detekcji obiektów. W zbiór ten wchodzi ponad 800 tysięcy obrazów oraz ponad 80

kategorii obiektów. Dodatkowo zbiór obrazów określa takie punkty szczegółowe jak lewe oko, nos, prawe biodro lub prawa kostka.

Zestaw danych Coco Dataset operuje na wysokiej jakości danych oraz najnowocześniejszych sieciach neuronowych. Zawiera on takie interfejsy programowania aplikacji jak Matlab, Python oraz Lua, które pomagają w analizowaniu i wizualizowaniu adnotacji. Podstawową strukturą i wspólnym elementem plików używanych przez adnotacje COCO Dataset jest JSON, który ma słowniki posiadające pary klucz-wartość, umieszczone w nawiasach kwadratowych. Może on również posiadać uporządkowane zbiory elementów lub zagnieżdżone słowniki [17].

Zestaw danych COCO umożliwia łatwe wyszukanie pożądanych klas dzięki pomocnej i łatwej w obsłudze stronie projektu. Wyszukiwarka klas została przedstawiona na rysunku nr 7.



Rysunek 7 Widok wyszukiwarki klas na stronie projektu COCO Dataset. Źródło: [16]

2.5 Środowisko Google Colab

Google Colab to bezpłatne środowisko Jupyter Notebook, które działa całkowicie w chmurze. Nie wymaga ono od użytkownika konfiguracji, a notatki, które w nim tworzymy, mogą być równolegle edytowane przez innych członków naszego zespołu. W środowisku Jupyter Notebook, przy próbie wykonania algorytmu uczenia głębokiego na dużym zbiorze danych, często można spotkać się z ograniczeniem pamięci. W związku z tym możemy korzystać z usługi Google Colab, która dysponuje dużymi mocami obliczeniowymi.

Do korzystania z usługi potrzebujemy konto Google oraz przeglądarkę, moc obliczeniowa naszego komputera nie ma w tym przypadku znaczenia. Jest to argument pierwszorzędny przy wyborze opisywanej usługi ze względu na wciąż drożące karty graficzne na rynku komponentów komputerowych. Usługa Google Colab pozwala na nieprzerwaną pracę przez 12 godzin. Po tym czasie środowisko zostaje wyczyszczone, a użytkownik musi zacząć pracę od nowa.

Google Colab jest kompatybilny z wieloma popularnymi bibliotekami do uczenia głębokiego, takimi jak Keras, PyTorch, TensorFlow oraz OpenCV. Dodatkowo istnieje możliwość uruchomienia wielu instancji CPU, GPU i TPU, jednakże zasoby są współdzielone między tymi instancjami.

Dla osób potrzebujących jeszcze większych mocy obliczeniowych oraz dłuższego czasu ciągłej pracy w środowisku powstała usługa Google Colab Pro. Jej cena wynosi na moment pisanía pracy 9.99 USD. Po zakupie subskrypcji otrzymujemy dostęp do GPU takich jak Tesla T4 oraz Tesla P100. Czas pracy zostaje ponadto wydłużony z 12 godzin do 24 godzin [18].

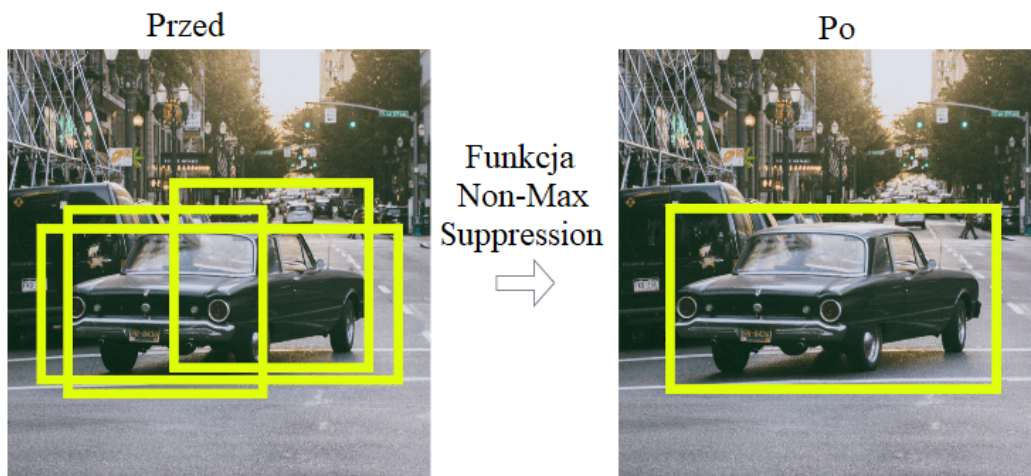
2.6 Algorytm YOLO

Algorytm YOLO przeznaczony jest do rozpoznawania obiektów w czasie rzeczywistym. Został opisany po raz pierwszy w przełomowej pracy Josepha Redmona z 2015 roku. Rozwinięcie jego nazwy brzmi You Only Look Once. Jego nazwa wzięła się z potrzeby jedynie jednego przejścia propagacji do przodu przez sieć neuronową, aby móc przewidywać detekcję. Algorytm uczy się na pełnych obrazach i jest w stanie bezpośrednio optymalizować wydajność wykrywania. To, co najbardziej przemawia za algorytmem YOLO, to jego bardzo duża szybkość działania oraz umiejętność nauki na pełnych obrazach co umożliwia poprawę zdolności kodowania kontekstowych informacji dotyczących klas. Algorytmy oparte na regresji takie jak YOLO, mają skłonności do wymieniania pewnych dokładności w detekcji na znaczną poprawę szybkości działania [19].

Algorytm YOLO dzieli analizowany obraz na komórki, zazwyczaj przy użyciu siatki w proporcjach 19×19 . Każda z komórek jest odpowiedzialna za przewidywanie 5 ramek ograniczających w przypadku, gdy w tej komórce znajduje się więcej niż jeden obiekt. To doprowadza do powstania 1805 ramek ograniczających dla jednego analizowanego obrazu. Oczywiście, większość komórek nie zawiera pożądanego obiektu, dlatego

wyznaczana jest również wartość prawdopodobieństwa, która służy do usuwania ramek o niskiej szansie wystąpienia obiektu.

Pierwotnym problem działania algorytmu You Only Look Once, było zaznaczanie tego samego obiektu wielokrotną liczbą ramek ograniczających [20]. Rozwiązaniem w tym przypadku jest funkcja Non-max suppression, która na początku wybiera ramki z największym prawdopodobieństwem wykrycia obiektu, a następnie pomija te z mniejszymi wartościami prawdopodobieństwa. Działanie tej funkcji zostało przedstawione na rysunku nr 8.



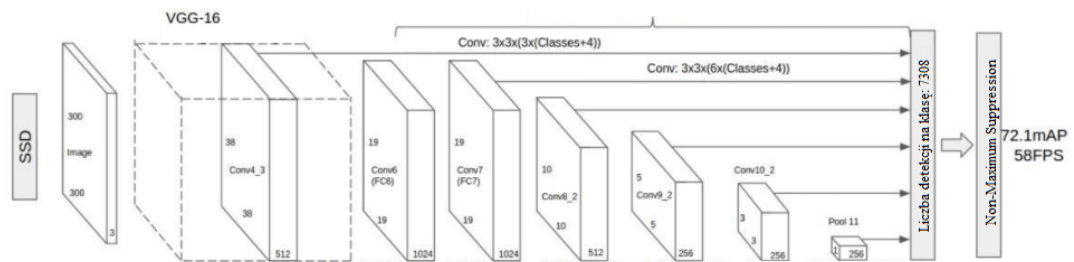
Rysunek 8 Działanie funkcji wybierającej ramki z największym prawdopodobieństwem wykrycia obiektu.

Źródło: [20]

2.7 Algorytm SSD

Algorytm SSD-The Single Shot Detector został opublikowany w listopadzie roku 2016 i osiągnął nowe rekordy pod względem wydajności i precyzji w zadaniach detekcji obiektów w czasie rzeczywistym. Uzyskał on ponad 74% precyzji z prędkością odświeżania 59 klatek na sekundę w standardowych zestawach danych. Człon nazwy Single Shot odnosi się do jednego cyklu przejścia, potrzebnego do wykonania zadań lokalizacji i klasyfikacji obiektów.

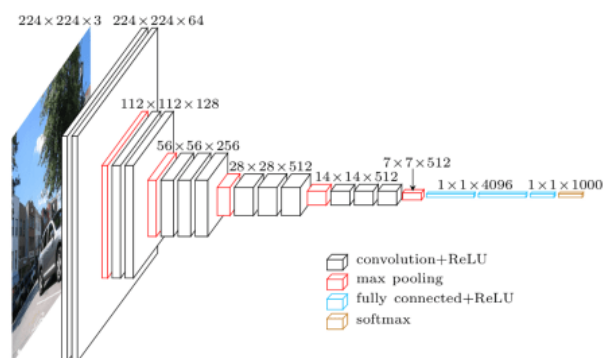
Diagram architektury algorytmu SSD ukazany został na rysunku nr 9. Algorytm odrzuca w pełni połączone ze sobą warstwy. Bazowa sieć używana w algorytmie cechuje się wysoką wydajnością w klasyfikacji obrazów wysokiej jakości. W algorytmie, zamiast w pełni połączonych ze sobą warstw, użyto zestawów pomocniczych warstw splotowych. Umożliwiło to wyodrębnianie cech charakterystycznych oraz zmniejszenie wielkości danych wejściowych do każdej kolejnej warstwy.



Rysunek 9 Diagram strukturalny algorytmu SSD. Źródło: [21]

Algorytm liczy dwie podstawowe wartości przy pomocy filtrów splotu. Jedną z nich jest współczynnik pewności, który mierzy prawdopodobieństwo poprawności obliczonej ramki ograniczającej. Współczynnik ten jest obliczany przy pomocy entropii krzyżowej, która ma charakter logarymiczny. Kolejną wartością jest współczynnik lokalizacji, który mierzy, jak daleko znajdują się obliczone ramki ograniczające od rzeczywistych obszarów ze zbioru uczącego. Po wyodrębnieniu map funkcji, algorytm SSD stosuje filtry splotu 3×3 dla każdej komórki, aby prognozować detekcję obiektów. Każdy filtr generuje 25 wartości wyjściowych, 21 wartości dla każdej z klas oraz jedno pole ograniczające.

W swoim działaniu algorytm SSD wykorzystuje tak naprawdę wieloskalowe warstwy map obiektów do niezależnej detekcji. Algorytm stopniowo zmniejsza wymiar przestrzenny oraz rozdzielczość map obrazów. Następnie, wykorzystuje on warstwy o niższej rozdzielczości do wykrywania obiektów o większej skali. Zostało to przedstawione na rysunku nr 10.



Rysunek 10 Diagram konwolucyjnej sieci neuronowej używanej przez algorytm SSD. Źródło: [21]

W wyniku działania powyższego algorytmu powstaje znacznie więcej prognoz negatywnych niż pozytywnych. Może to doprowadzić do nierównowagi klasowej, która może być szkodliwa dla procesów szkolenia ze względu na zbyt dużą dominację przestrzeni tła.

Jednak algorytm The Single Shot Detector wybiera te wartości, które posiadają największe odstępstwa od wartości pożądaných, a następnie przydziela je do przypadków negatywnych, zapewniając tym samym szybszy i stabilniejszy proces szkolenia.

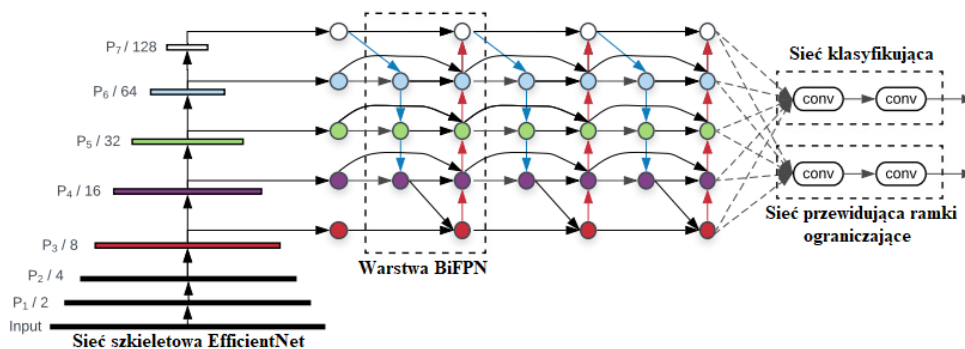
Algorytm SSD zyskuje dodatkowo na szybkości działania dzięki zaokrągłaniu pewnych wartości, takich jak piksele ramki ograniczającej. W wielu zadaniach detekcji obrazów różnica kilku pikseli nie robi znaczących różnic w ostatecznych wynikach [22].

2.8 Algorytm EfficientDet

Algorytm EfficientDet jest przygotowany przez zespół Google Brain, jako skalowalna architektura do wykrywania obiektów przy jednoczesnym osiągnięciu większej dokładności detekcji. Podstawowa koncepcja architektury EfficientDet to złożenie skalowania sieci w wielu wymiarach. Uwzględnia się w tym: rozdzielczość wejściową, głębokość oraz szerokość sieci. Zamiast skalować tylko jeden lub dwa wymiary sieci, EfficientDet rozszerza sieć we wszystkich wymiarach jednocześnie [23]. Dzięki temu nie istnieje potrzeba ręcznego dostrajania współczynników skalowania, który jest procesem żmudnym i często skutkuje uzyskaniem niepełnej wydajności działania modelu.

Zespół Google zaznacza, że architektura sieci EfficientDet jest do 9 razy mniejsza i zużywa znacznie mniej obliczeń w porównaniu z najnowocześniejszymi modelami przeznaczonymi do detekcji obiektów [24].

Celem złożonego skalowania sieci EfficientDet jest skalowanie modelu bazowego o nazwie EfficientDet D0. W schemacie skalowania sieci EfficientDet wyróżniamy sieć szkieletową, posiadającą te same współczynniki skalowania szerokości oraz głębokości algorytmu. Współczynniki te są używane do ponownego wyszkolenia punktów kontrolnych sieci. Diagram złożonego skalownia sieci EfficientDet przedstawiono na rysunku nr 11.



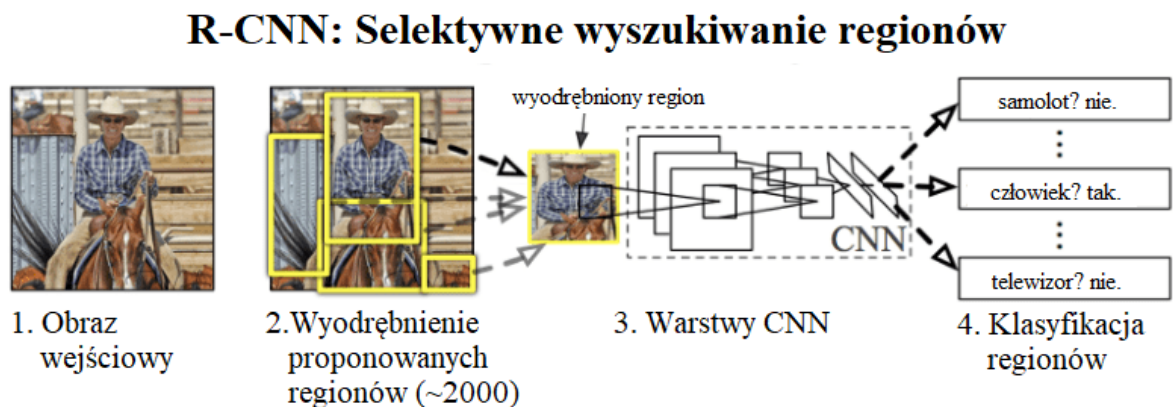
Rysunek 11 Diagram złożonego skalowania sieci EfficientDet. Źródło: [23]

2.9 Algorytm Faster R-CNN

W zadaniach związanych z analizą obrazów liczba interesujących użytkownika obiektów nie jest stała. Zmienne są również położenia przestrzenne obiektów na obrazie oraz ich proporcje. Powoduje to powstawanie dużej ilości analizowanych obszarów na obrazie oraz zwiększenie ilości wymaganych obliczeń.

W celu rozwiązania problemu namnażających się obliczeń oraz analizowanych obszarów na obrazach stworzono algorytm RCNN, który selektywnie wyodrębnia obszary z obrazu. Takie wyodrębnienia określa się jako propozycje regionów.

Algorytm wyszukiwania selektywnego przekształca zaproponowane 2000 regionów w kwadraty, a następnie wprowadza je do splotowej sieci neuronowej. Jako wynik otrzymuje się 4096 wymiarowy wektor cech [25]. Wyodrębniony wektor jest wprowadzany do algorytmu klasyfikującego obecność obiektu. Oprócz przewidywania obecności obiektu w odpowiednich regionach algorytm przewiduje również wartości przesunięć precyzji obwiedni. Oznacza to, iż w przypadku analizy obecności osoby na obrazie, twarz osoby w proponowanym regionie mogłaby zostać ucięta. Dlatego wartość przesunięcia precyzji obwiedni pozwala dostosować odsunięcie proponowanego regionu i zaznaczać obwiednią pełne kształty obiektów. Diagram selektywnego wyszukiwania regionów zaprezentowano na rysunku nr 12.

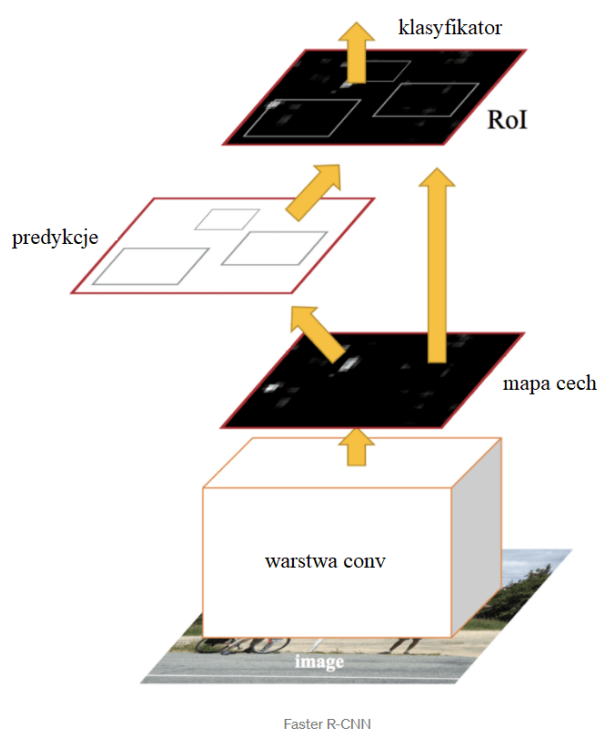


Rysunek 12 Selekttywne wyszukiwanie regionów na obrazie. Źródło: [25]

Problemy w działaniu sieci RCNN są jednak poważne. Wyszukanie sieci zajmuje bardzo dużo czasu, ponieważ należy klasyfikować 2000 propozycji regionów na każdy obraz. Poza tym, sieć RCNN nie jest odpowiednia do zastosowań w czasie rzeczywistym. Pojedyncza klasyfikacja obrazu zajmuje około 47 sekund.

Z tego powodu zrezygnowano z selektywnego znajdowania propozycji regionów i opracowano algorytm Faster RCNN. W algorytmie tym podobnie jak w przypadku sieci RCNN, na wejście dostarczany jest obraz, który zapewnia mapę cech splotowych.

Do wyszukiwania propozycji regionów w modelu sieci Faster RCNN, wykorzystywana jest osobna sieć. Następnie, propozycje regionów są przekształcane oraz przewidywana jest wartość przesunięć dla ramek ograniczających. Na rysunku nr 13 zaprezentowano diagram działania algorytmu sieci Faster RCNN.



Rysunek 13 Diagram działania algorytmu sieci Faster RCNN. Źródło: [25]

3. Własności funkcjonalne aplikacji do detekcji pieszego

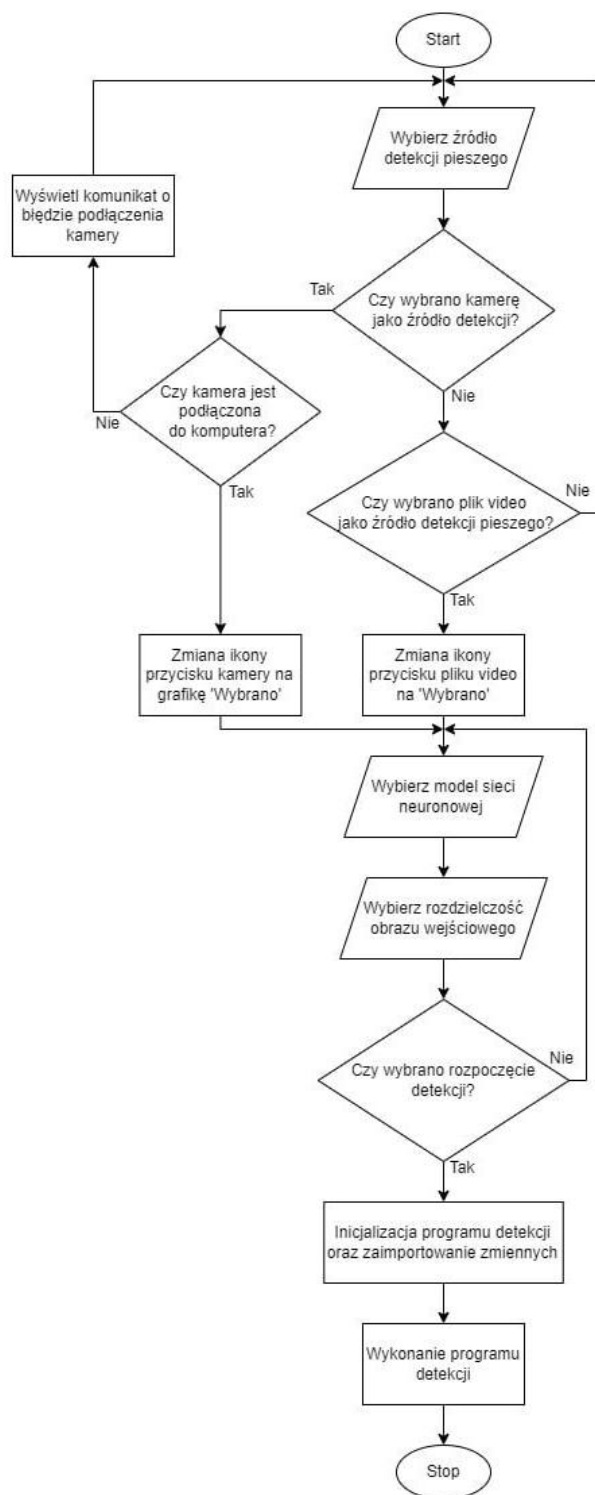
Wykonywana na potrzeby pracy dyplomowej aplikacja ma za zadanie poprawne rozpoznawanie sylwetek pieszego na jezdni. Rozpoznanie sylwetki człowieka jest obrazowane wyświetleniem ramki ograniczającej wokół rozpoznanego obszaru. Powyżej ramki ograniczającej jest wyświetlana również etykieta klasy z nazwą „pieszy”. Oprócz etykiety klasy, ważna jest również informacja zwrotna od sieci neuronowej przeprowadzającej analizę, na temat pewności wykonanych obliczeń. W tym celu, w procesie działania aplikacji obok nazwy etykiety klasy pieszego, będzie umieszczona informacja o prawdopodobieństwie pewności detekcji zwracanej przez sieć.

W systemach bezpieczeństwa odporność systemu na zakłócenia jest priorytetem, bez którego nie można zapewnić poprawnego działania aplikacji. Z przytoczonych w bibliografii źródeł wynika, iż największy wpływ na jakość detekcji obiektu na drodze ma oświetlenie oraz odległość od obiektu, w jakiej realizuje się rejestrowanie obrazu. Z tego względu aplikacja ma w swojej funkcjonalności zakładać rozpoznawanie sylwetki pieszego zarówno porą dzienną, jak i nocną, gdzie występujące różnice w oświetleniu są największe. Opracowana aplikacja ma również za zadanie rozpoznania sylwetki pieszego w różnych odległościach od kamery. Odległości te określono kolejno jako 5 [m], 10 [m] oraz 15 [m].

3.1 Interfejs użytkownika aplikacji

Interfejs użytkownika powinien umożliwiać wprowadzanie zmiennych, tak aby użytkownik mógł modyfikować działanie programu detekcji bez konieczności wprowadzania zmian w kodzie źródłowym. Pozwala to na modyfikację działania programu osobom nie zapoznanym z kodem oraz uniknięcie błędów podczas bezpośrednich zmian kodu. Opracowywana w pracy aplikacja, ma umożliwiać użytkownikowi wybór źródła detekcji, modelu sieci neuronowej oraz rozdzielczości obrazu wejściowego.

Algorytm działania opracowanego na potrzeby niniejszej pracy interfejsu użytkownika aplikacji został zaprezentowany na rysunku nr 14.



Rys. 14. Algorytm działania interfejsu użytkownika. Źródło: Opracowanie własne.

Interfejs użytkownika aplikacji, na samym początku działania programu umożliwia wybór źródła detekcji. Użytkownik może wybrać jako źródło kamerę podłączoną do komputera lub plik video. Algorytm sprawdza w pętli, którą opcję zaznaczył użytkownik. Przy wyborze pliku video jako źródła obrazu, ustawiany jest filtr eksploratora plików na formaty MPG,

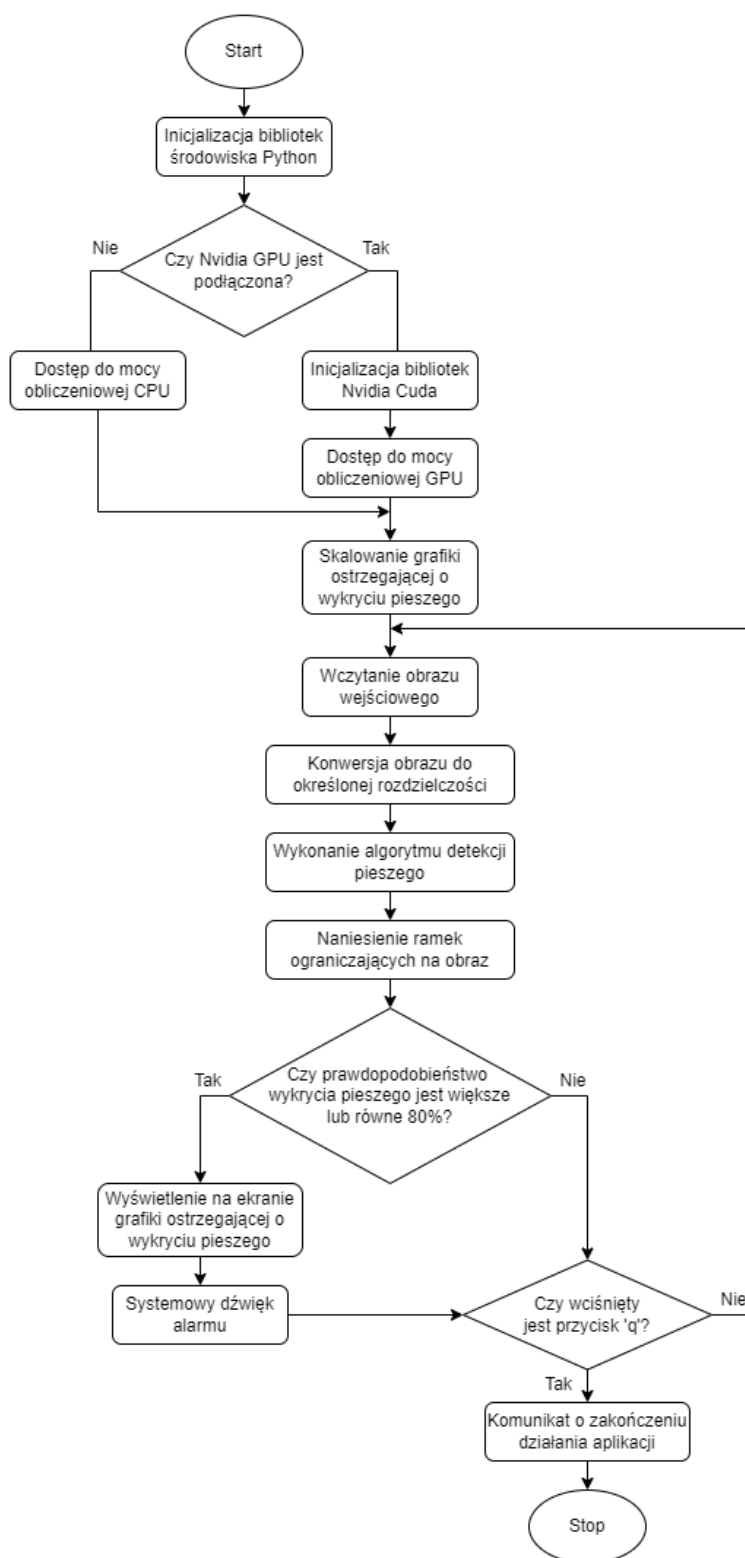
MP4 oraz AVI, a następnie wyświetlane jest okno eksploratora plików. Źródło obrazu powinno zostawać poprawnie importowane do aplikacji niezależnie od miejsca na dysku, w jakim znajduje się plik. W przypadku wyboru kamery jako źródła obrazu aplikacja poprzez funkcję biblioteki OpenCV sprawdza, czy możliwe jest jej użycie. W przypadku braku możliwości wykorzystania kamery jako źródła obrazu użytkownik powinien dostać informację o występującym błędzie. Jeżeli opcja wyboru źródła obrazu przebiegnie pomyślnie, odpowiadający wyborowi źródła przycisk zostaje zamieniony na grafikę z tekstem „Wybrano”.

Interfejs użytkownika aplikacji umożliwia również wybór modelu sieci neuronowej, z której wykorzystaniem użytkownik pragnie przeprowadzić proces detekcji pieszego. Do wyboru w aplikacji powinny być dostępne 3 modele sieci neuronowych, opracowane przez zespół TensorFlow. Następnie, użytkownik wybiera rozdzielczość obrazu wejściowego. Interfejs umożliwia uruchomienie programu detekcji z uwzględnieniem wprowadzonych zmiennych za pomocą przycisku.

3.2 Algorytm działania programu detekcji

Na samym początku działania programu detekcji, następuje inicjalizacja bibliotek środowiska Python. Następnie w programie znajduje się instrukcja warunkowa, sprawdzająca, czy Nvidia GPU jest poprawnie podłączona do komputera oraz, czy możliwe jest jej użycie poprzez zainstalowane narzędzia firmy Nvidia. Jeżeli karta jest poprawnie podłączona wraz z narzędziami, następuje inicjalizacja bibliotek Nvidia CUDA oraz zapewnienie dostępu do mocy obliczeniowej GPU. W przypadku braku podłączenia karty graficznej używane są zasoby obliczeniowe procesora. Następnie, program importuje oraz odpowiednio skaluje grafikę ostrzegającą o wykryciu pieszego. Kolejnym krokiem jest wejście programu do pętli, w której następuje kolejno wczytanie obrazu wejściowego, konwersja obrazu do rozdzielczości określonej przez użytkownika w interfejsie użytkownika aplikacji, naniesienie ramek ograniczających na obraz. W pętli znajduje się instrukcja warunkowa sprawdzająca, czy obliczone prawdopodobieństwo detekcji sylwetki pieszego na obrazie jest większe lub równe 80%. W przypadku spełnienia warunku, na obrazie zostaje wyświetlona grafika ostrzegająca o wykryciu pieszego. Dodatkowo informacja o wykryciu pieszego jest sygnalizowana systemowym dźwiękiem alarmu. W przypadku niespełnienia warunku instrukcji program sprawdza, czy wciśnięty został przycisk q na klawiaturze, dzięki któremu możliwe jest zakończenie wykonywania programu. Jeżeli nie został on wciśnięty, program

wraca na początek pętli, gdzie wczytywany jest ponownie obraz wejściowy. Algorytm działania programu detekcji został przedstawiony na rysunku nr 15.



Rysunek 15 Algorytm działania programu detekcji. Źródło: Opracowanie własne.

4. Zbiór danych obrazowych

Detekcja obiektów jako ważny obszar dziedziny przetwarzania obrazów oraz wizji komputerowej, znacząco poprawiła swoją wydajność poprzez zastosowanie technologii uczenia maszynowego. W jej skład wchodzi techniki programowania komputerów umożliwiające im uczenie się z zestawów danych. Zestawy te zawierają pojedyncze elementy, zwane próbkami uczącymi.

Systemy uczenia maszynowego dzielimy na: nadzorowane, nienadzorowane, półnadzorowane oraz na uczenie przez wzmacnianie.

W niniejszej pracy do procesu uczenia sieci neuronowej wykorzystano metodyki uczenia nadzorowanego. Charakteryzuje się ono przekazywaniem algorytmowi danych uczących wraz z rozwiązaniem problemu, w postaci sklasyfikowanych danych uczących oraz danych testowych. Największą zaletą uczenia nadzorowanego jest możliwość poprawy algorytmu, w momencie, gdy popełni on błąd przy udzielaniu odpowiedzi.

4.1 Problematyka tworzenia zbioru danych

W przypadku nadzorowanego typu uczenia maszynowego za najważniejszy czynnik uważa się dane. To od ich jakości zależą wyniki projektów. Obecnie dużą uwagę przywiązuje się do poprawy jakości danych używanych wraz z coraz bardziej złożonymi algorytmami uczenia maszynowego. Spowodowane jest to ograniczeniami, jakie reprezentują obecnie dostępne modele, które najskuteczniej można przezwyciężyć, używając do treningu większej liczby lepszych jakościowo danych. Jednak nie zawsze większa liczba próbek oznacza bardziej reprezentatywny zbiór danych. Za przykład mogą posłużyć publiczne zbiory danych, które są łatwo dostępne oraz zawierają potężne liczby próbek uczących. Zawarte w nich obrazy są jednak często nieprawidłowo sklasyfikowane. Dodatkowo publiczne zbiory danych często zawierają obrazy z obiektami odstającymi od podanych filtrów wyszukiwania.

Alternatywą dla zbiorów danych publicznych jest stworzenie danych syntetycznych, pochodzących z własnego generatora wykorzystującego oprogramowanie do modelowania 3D. Jest to szybki sposób do generowania olbrzymich ilości danych. Metoda ta pozwala również na automatyczne dodawanie etykiet i tym samym zaoszczędzenie czasu. Obrazy syntetyczne są jednak mniej złożone niż rzeczywiste zdjęcia z ruchu ulicznego.

Mając na uwadze opisane problemy z pozyskiwaniem danych do tworzenia zbioru uczącego oraz testowego, w pracy stworzono własny zestaw bazy zdjęć z ruchu ulicznego, który następnie poddano procesowi etykietowania.

4.2 Zgromadzenie danych obrazowych

Opracowany na potrzeby pracy zbiór danych zawiera obrazy z nagrań przeprowadzonych w terenie zabudowanym. Nagrania rejestrowano z perspektywy kamery umieszczonej na desce rozdzielczej samochodu osobowego.

Nagrania rejestrowano w rozdzielczości 1920 x 1080 pikseli, 30 klatek na sekundę w formacie MP4. W rezultacie, w skład opracowanego zbioru danych wchodzi 500 obrazów.

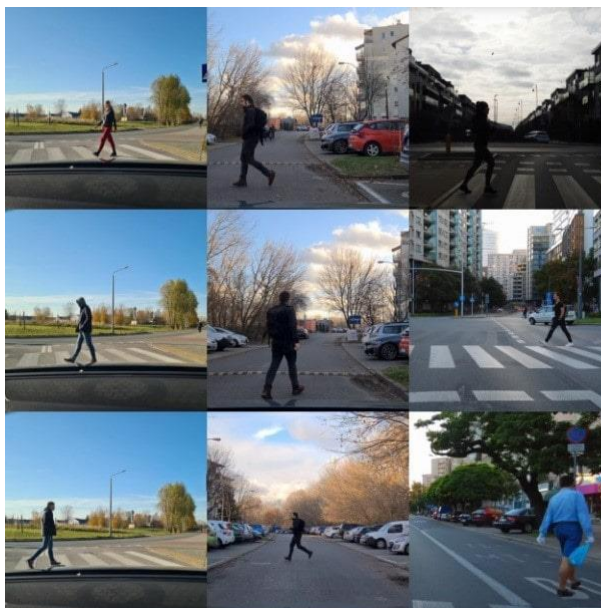
Uzyskane obrazy zostały następnie poddane konwersji stratnej do formatu JPG, w celu przyspieszenia procesu uczenia modelu sieci neuronowej oraz zmniejszenia objętości bazy danych. Całkowity rozmiar zbioru danych wynosi 175 MB. Pierwsza grupa nagrań przeprowadzana była porą dzienną, przy dobrych warunkach oświetleniowych. Na nagraniach pieszy byli w pozycji stojącej, a pojazd wraz z kamerą się nie poruszał. Przykładowe dane obrazowe z pieszymi w pozycjach stojących porą dzienną zaprezentowano na zdjęciu nr 1.



Zdjęcie 1 Przykład danych obrazowych z pieszym w pozycji stojącej porą dzienną. Źródło: Opracowanie własne.

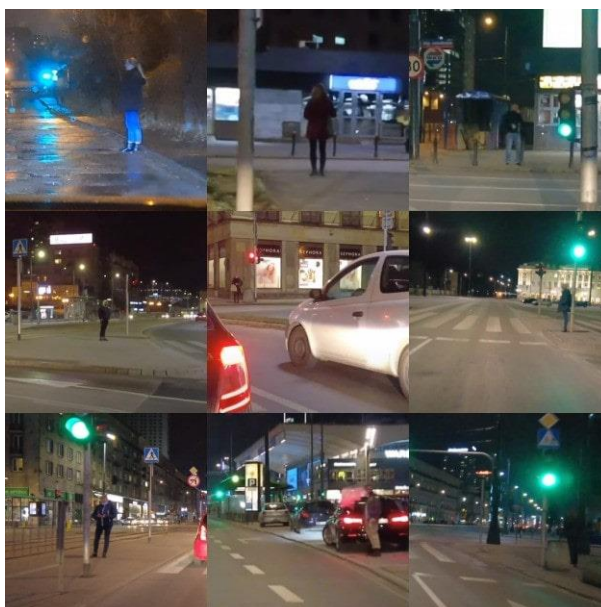
Kolejna grupa nagrań zakładała uwzględnienie pozycji pieszego w chodzie. Nagrania realizowane były w dobrych warunkach oświetleniowych, a pojazd wraz z kamerą

nie poruszał się. Przykładowe dane obrazowe z pieszymi w chodzie porą dzienną zaprezentowano na zdjęciu nr 2.



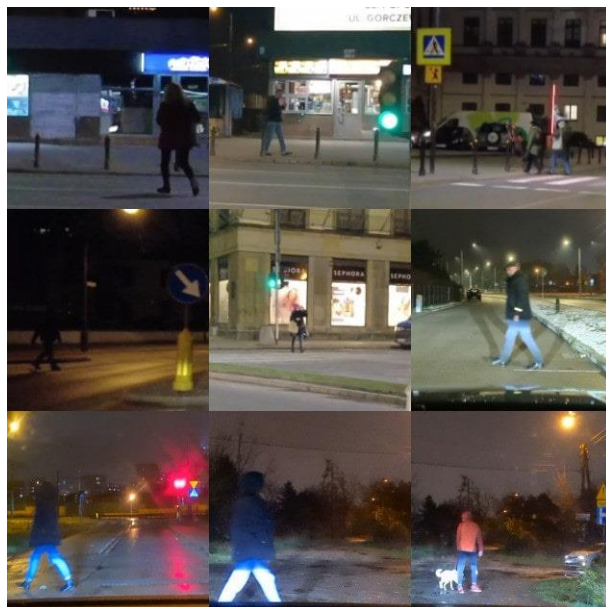
Zdjęcie 2 Przykład danych obrazowych z pieszym w chodzie porą dzienną. Źródło: Opracowanie własne.

Następnie, nagrania przeprowadzono porą nocną przy słabych warunkach oświetleniowych. Przykładowe dane obrazowe z pieszymi w pozycjach stojących porą nocną zaprezentowano na zdjęciu nr 3.



*Zdjęcie 3 Przykład danych obrazowych z pieszym w pozycji stojącej porą nocną.
Źródło: Opracowanie własne.*

Nagrania opracowane w nocy uwzględniały również sylwetki pieszych w chodzie. Przykładowe dane obrazowe z pieszymi w chodzie porą nocną zaprezentowano na zdjęciu nr 4.



Zdjęcie 4 Przykład danych obrazowych z pieszym w chodzie porą nocną. Źródło: Opracowanie własne.

4.3 Etykietowanie danych obrazowych

Zgromadzone w pracy dane obrazowe poddano procesowi etykietowania. Polegał on na nadaniu obrazom informacji o klasach instancji, wykorzystywanym w przypadku uczenia nadzorowanego. W tym celu postacie pieszych występujące w przygotowanych obrazach zostały oznaczone ramkami ograniczającymi w programie LabelImg. Jest to bezpłatne narzędzie o otwartym kodzie źródłowym stworzone do graficznego etykietowania obrazów. Narzędzie to jest napisane w środowisku Python i używa bibliotek QT jako interfejsu graficznego. Program LabelImg obsługuje etykietowanie w formatach takich jak VOC XML lub YOLO. W pracy wykorzystano domyślny format VOC XML z uwagi na łatwość konwertowania tego typu plików. Przed procesem etykietowania należy zmienić plik tekstowy aplikacji zawierający domyślne nazwy klas, zastępując je własnymi nazwami. Zapisanie zdjęcia z utworzonymi ramkami ograniczającymi oraz nazwami etykiet z poziomu programu, powoduje również zapisanie adnotacji do pliku XML wraz ze wszystkimi szczegółowymi danymi wymaganymi przez model uczenia maszynowego. Plik tekstowy zawiera kolejno nazwę folderu, ścieżkę pliku oraz wymiary obrazu. W pliku XML są również

wymienione obiekty, które zostały zidentyfikowane wraz z nazwami etykiet, wymiary narysowanych ramek ograniczających i dodatkowe informacje na temat ich ułożenia.

Zrzut ekranu przedstawiający proces etykietowania danych obrazowych w programie LabellImg został przedstawiony na rysunku nr 16.

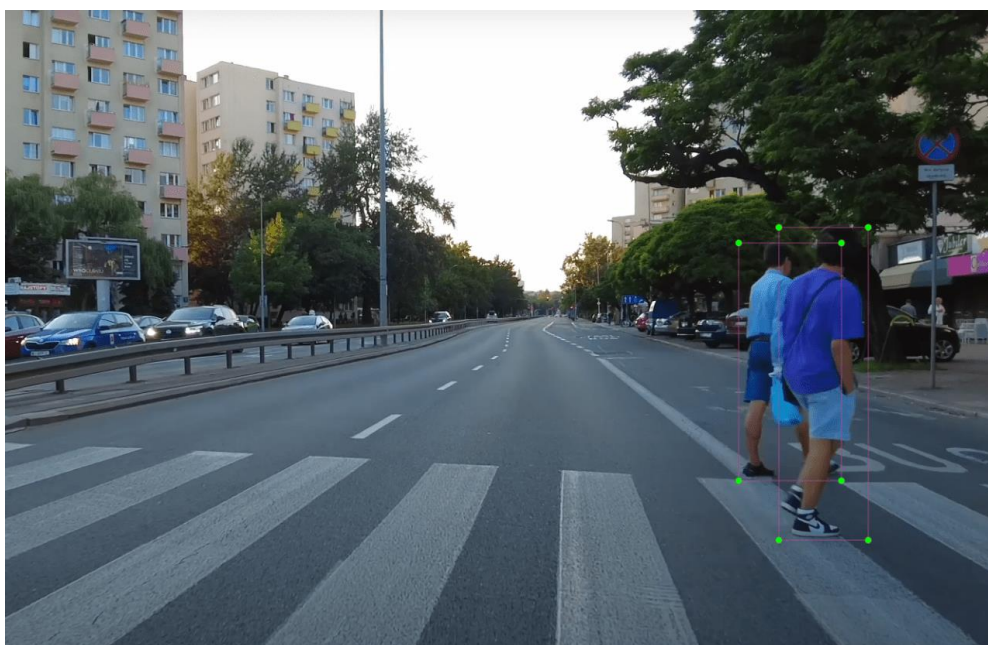


Rysunek 16 Etykietowanie danych obrazowych w programie LabellImg. Źródło: Opracowanie własne.

Podczas procesu etykietowania należy zwracać uwagę na zaznaczanie pełnych kształtów pożądanego obiektu. Zaleca się również dołączanie niewielkich buforów niebędących obiektem zamiast wykluczania części obiektów z uwagi na ich złożony kształt. W ten sposób model jest w stanie nauczyć się rozpoznawania krawędzi etykietowanych obiektów. W przypadku kiedy etykietowane obiekty nachodzą na siebie, należy zaznaczyć oba z nich tak, jakby możliwe było rozpoznanie ich całkowitego kształtu. Dzięki tej metodzie uczony model będzie w stanie rozpoznawać prawdziwe granice obiektów. Obiekty niepełne, których całkowite kształty wychodzą poza obszar obrazów, również powinny być etykietowane. Przypadek z obiektem niepełnym oraz obiektami nachodzącymi na siebie wraz z opisanymi ramkami ograniczającymi został przedstawiony na rysunkach odpowiednio nr 17 i 18.



Rysunek 17 Przykład poprawnego etykietowania danych obrazowych z obiektem wychodzącym poza obszar obrazu.. Źródło: Opracowanie własne.



Rysunek 18 Przykład poprawnego etykietowania danych obrazowych z obiektami nachodzącymi na siebie. Źródło: Opracowanie własne.

4.4 Zbiór danych testowych i zbiór danych treningowych

Użyta w pracy metoda testowania modelu uczenia maszynowego zakłada podział zbioru danych na zbiór danych uczących oraz zbiór danych testowych.

Najczęściej zbiór danych uczących stanowi 80% wszystkich danych, a pozostałe dane przechowuje się jako zbiór testowy. Dzięki takiemu podziałowi możliwe staje się wyznaczenie błędu uogólniania. Zbiór danych testowych umożliwia również w pewnym stopniu przybliżenie wydajności działania modelu w przypadku nieznanych danych.

Niewielka wartość błędu uczenia, czyli rzadkość w błędnych wynikach wobec zbioru uczącego, przy dużym błędzie uogólnienia oznacza przetrenowanie modelu.

Zbiór testowy został stworzony w celu oceny wydajności modelu po zakończeniu procesu uczenia. Zaleca się, aby już na wczesnym etapie zapoznania się ze zbiorem danych, sporządzić zbiór testowy. Dłuższa analiza struktury danych objawia się ryzykiem przetrenowania, czyli rozpoznaniem nadmiernej liczby nieistotnych wzorców. Ludzki mózg w szybkim tempie jest w stanie podczas analizy danych dostrzec pozornie interesujące wzorce, a następnie na ich podstawie wybrać określony model uczenia maszynowego. Zjawisko to nosi nazwę obciążenia związanego z podglądaniem danych. Ponadto, zbiór testowy oraz treningowy muszą być odseparowane. Oznacza to, że żadnych danych pochodzących ze zbioru treningowego nie można użyć w zbiorze testowym. Dzięki odseparowaniu danych algorytm uczenia maszynowego nie będzie w stanie nauczyć się całego zbioru danych przy kolejnych próbach uruchomienia programu. W przypadku niedomiaru danych nie należy również ograniczać wielkości zbioru testowego, ponieważ uniemożliwia to wiarygodną ocenę modelu.

W pracy do utworzenia zbioru testowego oraz zbioru uczącego użyto skryptu napisanego w środowisku Python. Skrypt wybrał 80% losowych próbek ze wszystkich danych dostępnych w utworzonym zbiorze, a następnie przydzielił wybrane próbki do odpowiedniego folderu o nazwie train. Pozostałe dane umieszczone zostały w folderze o nazwie test. Do utworzenia skryptu wykorzystano framework Sklearn, który zawiera wbudowaną funkcję o nazwie `train_test_split`, przeznaczoną do separacji zbiorów danych. W skrypcie istnieje również zmienna `test_size`, która służy do określenia wielkości zbioru testowego względem całego zbioru danych. Dodatkowo utworzony skrypt wraz z losowymi próbkami danych przenosi do folderów odpowiadające konkretnym próbkom pliki w formacie XML, opisane w podrozdziale 4.3.

(strona celowo zostawiona pusta)

5. Uczenie modelu sieci neuronowej

Zastosowana w pracy metoda uczenia transferowego, polega na wykorzystaniu sieci wstępnie przeszkolonej do rozwiązywania nowego zadania. Główną zaletą użytej metody jest możliwość trenowania sieci przy stosunkowo niewielkiej ilości danych. Zmniejsza to również czas potrzebny do wyszkolenia sieci, a także rozwiązuje problem ręcznego dopasowywania parametrów sieci. W pracy zastosowano wstępnie przeszkolone modele sieci neuronowych ze strony projektu TensorFlow Zoo. Spośród wszystkich modeli wybrano trzy, które prezentują różne architektury oraz wykazują dużą wydajność w testach opracowanych przez twórców TensorFlow. Udostępnione modele są darmowe, a twórcy publikują wiele przydatnych poradników pomagających początkowym programistom w ich użyciu.

Do procesu uczenia sieci wykorzystano środowisko Google Colab, które w prosty sposób udostępnia zdalny dostęp do dużych mocy obliczeniowych za pośrednictwem akceleratorów graficznych. Poniżej zaprezentowano przygotowanie danych obrazowych do procesu uczenia sieci neuronowej oraz sam proces z wytłumaczeniem użytej funkcjonalności środowiska Google Colab.

5.1 Konwersja danych do plików CSV

Wykorzystanie danych uczących poprzez framework TensorFlow wymaga przekonwertowania stworzonych plików tekstowych o rozszerzeniu XML na pliki wartości rozdzielonych przecinkami, o rozszerzeniu CSV. W tym celu użyto skryptu napisanego w środowisku Python. W skrypcie funkcja o nazwie `xml_to_csv` importuje oraz wyodrębnia dane z pliku o rozszerzeniu XML z wykorzystaniem wbudowanej biblioteki środowiska Python o nazwie `xml.etree.ElementTree`. Wyodrębnione dane zostają zainicjalizowane jako podstawowa struktura danych biblioteki Pandas, która ma szerokie zastosowania w dziedzinie analizy danych. Wykorzystanie biblioteki Pandas pozwala w prosty sposób zachować poprawną kolejność oraz nazwy kolumn, a także modyfikować całą strukturę danych jako pojedynczy obiekt. Następnie, utworzona struktura danych zostaje zwrócona do głównego wątku programu i zapisana do pliku wartości rozdzielonego przecinkami. W wyniku uruchomienia skryptu zostają utworzone w nadrzędnym folderze przekonwertowane pliki o nazwach `test.csv` oraz `train.csv`.

5.2 Utworzenie plików TfRecord oraz mapy etykiet

W celu zwiększenia wydajności procesu uczenia sieci neuronowej, w pracy użyto rekomendowanego przez twórców narzędzia TensorFlow formatu plików TfRecord. Dzięki plikom o tym rozszerzeniu, dane uczące są przechowywane w układzie binarnych rekordów sekwencyjnych co pozwala na ich szybkie przesyłanie ze względu na krótszy czas dostępu oraz mniejszą ilość miejsca jakie zajmują na dysku. Ma to znaczący wpływ na wydajność importowania danych, a tym samym na czas uczenia modelu sieci neuronowej.

W plikach Rekord, każda pojedyncza próbka danych jest słownikiem przechowującym mapowanie pomiędzy kluczem i danymi rzeczywistymi. Pliki formatu TfRecord mogą być odczytywane jedynie sekwencyjnie.

Przed zapisem danych binarnych do pliku TfRecord, należy na początku określić strukturę danych. W celu określenia struktury pojedynczych próbek danych, wykorzystano udostępnioną w tym celu przez framework TensorFlow komendę `tf.train.Example`. Komenda ta nie jest wywołaniem klasycznej klasy środowiska Python, a buforem danych opracowanym przez firmę Google do serializacji uporządkowanych danych. Strukturyzacja danych utworzonych w programie LabelImg, a następnie przekonwertowanych na plik CSV, wymaga utworzenia list, gdzie każda lista zawiera dane tego samego formatu. W tym celu należało zbadać, jakiego formatu są poszczególne kolumny danych zapisanych w plikach CSV. W celu wyodrębnienia poszczególnych typów danych, dokonano konwersji pliku CSV z wykorzystaniem wbudowanych funkcji biblioteki Pandas. Przykładowe dane zostały przedstawione w tabeli nr 1.

Tabela 1 Przykładowe dane pochodzące z procesu etykietowania danych obrazowych. Źródło: Opracowanie własne.

filename	width	height	class	xmin	ymin	xmax	ymax
107-min.jpg	1608	762	pieszy	1378	439	1459	625
108-min.jpg	1576	760	pieszy	1046	445	1101	575
114-min.jpg	1574	766	pieszy	1257	438	1305	572
122-min.jpg	1364	788	pieszy	991	457	1040	590
123-min.jpg	1893	739	pieszy	518	353	716	690
129-min.jpg	1462	808	pieszy	1070	434	1187	740
129-min.jpg	1462	808	pieszy	1299	458	1403	727
129-min.jpg	1462	808	pieszy	838	472	884	596
130-min.jpg	1384	756	pieszy	1221	407	1357	692
136-min.jpg	1292	762	pieszy	942	435	1005	592
138-min.jpg	1402	667	pieszy	400	346	443	471
140-min.jpg	1476	776	pieszy	1100	397	84	669

Dla kolumn zawierających nazwy plików oraz nazwy klas zastosowano utworzenie list za pomocą komendy `bytes_feature`. Natomiast dla kolumn zawierających rozmiar obrazu oraz parametry ramki ograniczającej zastosowano komendę `float_list_feature`.

Następnie do zapisu danych na dysku użyto funkcji `tf.python_io.TFRecordWriter`. W wyniku działania skryptu na dysku zostały utworzone pliki o nazwach `test.record` oraz `train.record`.

Używany w pracy format plików typu `TfRecord`, należy do formatu zestawów danych wizyjnych, które używają do swojego działania mapy etykiet. Jest to prosty plik tekstowy, łączący nazwy klas z wartościami całkowitymi. Projektowana w pracy aplikacja używa jedynie jednej klasy adnotacji o nazwie „pieszy”. W pliku tekstowym zawierającym mapę adnotacji, ID używanej klasy oznaczone jest cyfrą 1. Mapa etykiet używana jest zarówno w procesie uczenia, jak i w procesie wykrywania.

5.3 Konfiguracja modelu sieci neuronowej

Udostępniany przez twórców modeli sieci neuronowych plik konfiguracyjny pozwala na dostosowanie zmiennych do tworzonych aplikacji. Zmieniając wartości zmiennych klas zawartych w pliku konfiguracyjnym, możliwe jest optymalizowanie procesu uczenia oraz późniejszego działania sieci neuronowej. Większość parametrów zawartych w pliku konfiguracyjnym jest określona i przetestowana przez zespół TensorFlow.

W przypadku ustawień danych wejściowych do modelu sieci neuronowej zmieniono domyślną liczbę klas zdefiniowanych w zbiorze danych. Opracowany zbiór danych zawiera jedną klasę o nazwie `pieszy`, zawierającą sylwetki ludzi poruszających się pieszo z perspektywy kamery umieszczonej na desce rozdzielczej samochodu osobowego. Następnie zmieniono w pliku konfiguracyjnym ścieżkę dostępu do zmiennych o nazwie `checkpoints`, zawierających dokładne wartości parametrów używanych przez model. Zmienne te nie zawierają żadnych informacji odnoszących się do obliczeń wykonywanych przez model, jednak są przydatne w przypadku dostępu do otwartego kodu źródłowego modelu sieci neuronowej.

W pliku konfiguracyjnym zmieniono również domyślny tryb działania modelu sieci neuronowej. Pobrany model pierwotnie w pliku konfiguracyjnym jest używany do celów klasyfikacji obrazów. Zmiana trybu działania sieci na detekcję obrazów następuje poprzez nadpisanie zmiennej klasy dotyczącej uczenia sieci neuronowej na ciąg znaków `detection`.

Zmienna `batch_size` określa liczbę próbek ze zbioru danych jednocześnie podawanych modelowi sieci w procesie uczenia. Po iteracji określonej wartości próbek, prognozy detekcji obiektów są porównywane z oczekiwanymi zmiennymi wyjściowymi i wyliczany jest błąd detekcji. Poprzez uzyskaną wartość błędu, algorytm aktualizuje model i przesuwa się wzdłuż gradientu błędu. Zmienna `batch_size` jest jednym z hiperparametrów mających znaczący wpływ na optymalizację procesu uczenia modelu sieci neuronowej. W pracy przypisano zmiennej `batch_size` wartość 4, z uwagi na użycie do procesu uczenia modelu platformy Google Colab, dzięki której możliwy jest dostęp do dużej mocy obliczeniowej.

Opisywany plik konfiguracyjny do poprawnego działania potrzebuje określenia ścieżki dostępu do wcześniej utworzonej bazy danych. W tym celu określono ścieżki dostępu do mapy etykiet, pliku `train.record` oraz pliku `test.record`, zawierających strukturyzowane wartości binarne etykietowanych obrazów pieszych.

5.4 Uczenie modelu w środowisku Google Colab

W pracy zdecydowano się na uczenie modelu sieci neuronowej z wykorzystaniem środowiska Google Colab, ponieważ umożliwia on pracę z układami GPU oraz TPU. Funkcjonalność ta znacząco zmniejsza czas potrzebny do wytrenowania modelu sieci.

Skompresowany zbiór danych, pliki typu `TfRecord`, mapy etykiet oraz wybrany model sieci neuronowej udostępniono na prywatnym dysku Google Drive. Następnie, w środowisku Google Colab zamontowano dysk Google z wykorzystaniem funkcji `drive.mount()`. Kolejnym krokiem było rozpakowanie folderu roboczego oraz zainstalowanie za pomocą menedżera `pip` pakietów `tf-models` oraz `tf_slim` zawierających zbiór modeli korzystających z interfejsów programowania aplikacji TensorFlow.

W celu ustawienia akceleratora sprzętowego w ustawieniach notatnika zmieniono pozycję na GPU. Środowisko Google Colab automatycznie zainicjalizowało połączenie z serwerem oraz przydzieliło zasoby obliczeniowe. Napisany w środowisku skrypt został przedstawiony na rysunku nr 19.

```

# Zainicjalizowanie dysku Google Drive
from google.colab import drive
drive.mount('/content/drive')

[ ] # Podgląd zawartości zainicjalizowanego dysku Google Drive
!ls

[ ] # Rozpakowanie skompresowanego folderu ze zbiorem danych oraz modelem sieci neuronowej
!unzip '/content/drive/My Drive/Tf2/object_detection.zip' -d '/content/drive/My Drive/Tf2/'

[ ] # Zainstalowanie pakietu modeli korzystających z API TensorFlow
!pip install tf-models-official

[ ] # Zainstalowanie pakietu służącego do uczenia oraz testowania modelu sieci neuronowej
!pip install tf_slim

[ ] # Przejście do folderu roboczego
!cd '/content/drive/My Drive/Tf2/object_detection/'

[ ] # Uruchomienie procesu uczenia modelu sieci neuronowej
!python model_main_tf2.py --pipeline_config_path=training/ssd_efficientdet_d0_512x512_coco17_tpu-8.config --model_dir=training --alsologtostderr

[ ] # Stworzenie wykresu wnioskowania w celu testowania modelu sieci neuronowej
!python exporter_main_v2.py --trained_checkpoint_dir=training --pipeline_config_path=training/ssd_efficientdet_d0_512x512_coco17_tpu-8.config --output_directory inference_graph

```

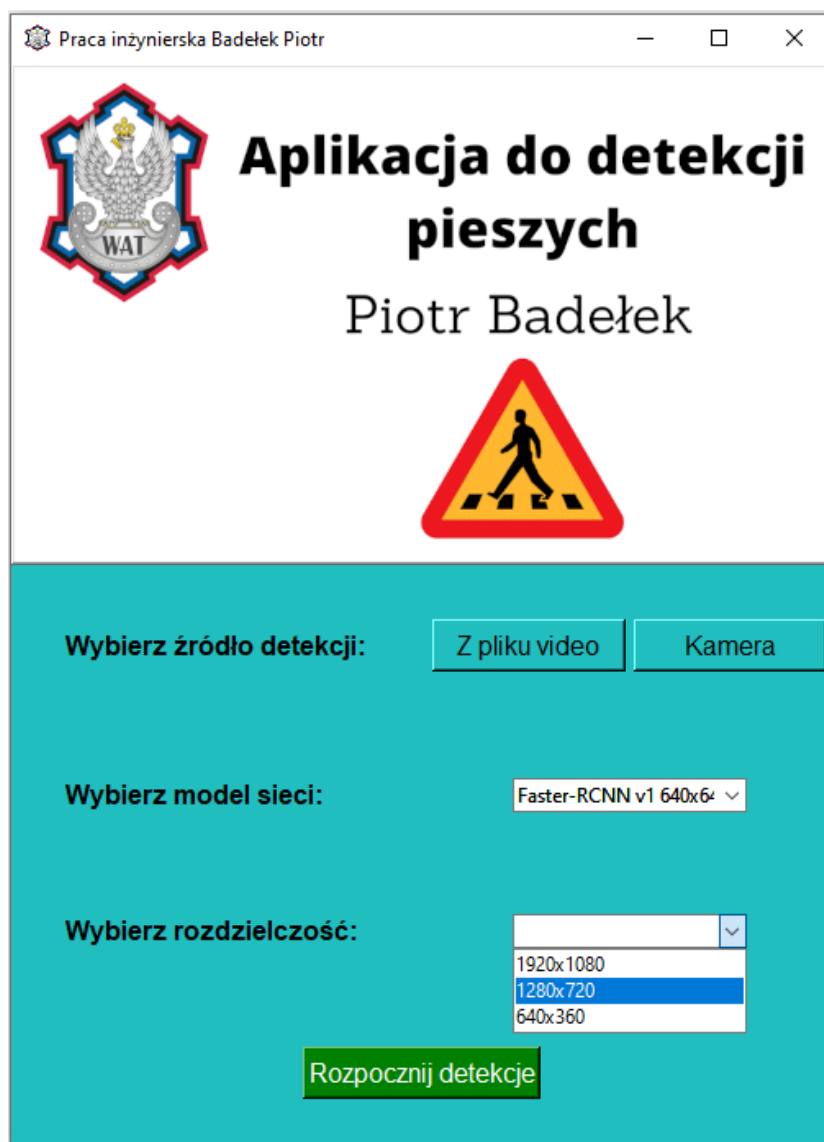
Rysunek 19 Widok użytego w pracy skryptu w środowisku Google Colab do procesu uczenia sieci neuronowej. Źródło: Opracowanie własne.

(strona celowo zostawiona pusta)

6. Uruchomienie opracowanej aplikacji

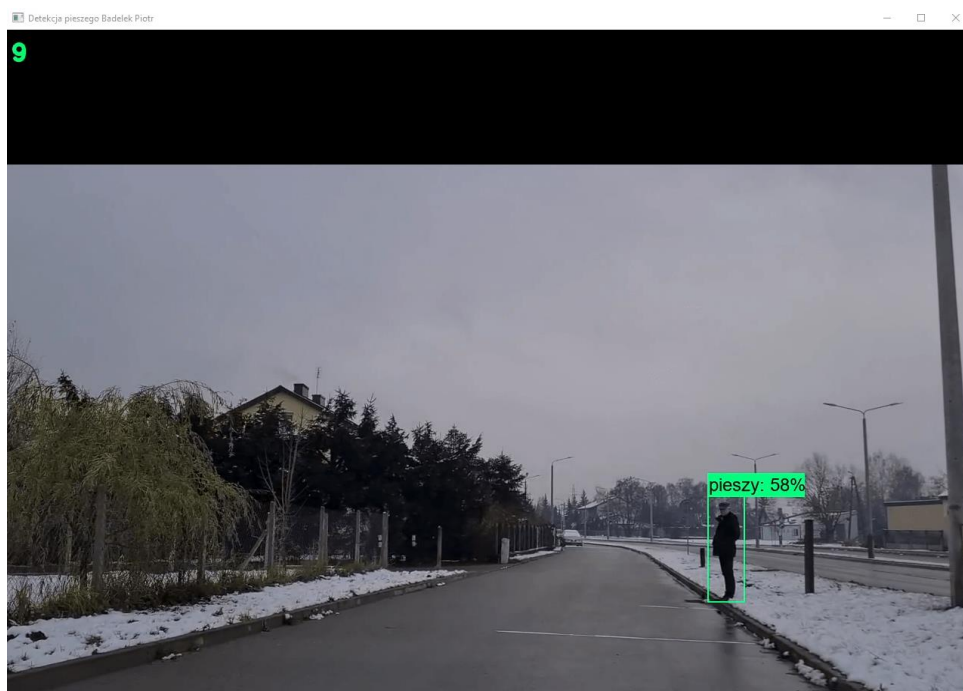
Po uruchomieniu opracowanej aplikacji początkowo wyświetlany jest interfejs użytkownika. Został on stworzony w celu wprowadzenia wymaganych zmiennych do działania programu detekcji. Oprogramowanie interfejsu działa zgodnie z algorytmem przedstawionym na rysunku nr 14. Użytkownikowi po kliknięciu przycisku wyboru źródła detekcji z pliku video wyświetlony zostanie eksplorator plików. Analiza obrazu możliwa jest również ze źródłem ustawionym na kamerkę internetową. Pozostałe dane zostają przez użytkownika określone poprzez rozsuwane suwaki, z dostępnymi w aplikacji opcjami.

Użytkownik do wyboru w aplikacji ma trzy algorytmy detekcji: SSD EfficientDet D0 512x512, Faster-RCNN v1 640x640 oraz SSD MobileNet v1 640x640. Zaimplementowana funkcja wyboru algorytmów sieci neuronowych jest przydatna przy testowaniu działania programu detekcji oraz wyborze optymalnego modelu dla danych warunków środowiskowych. Ostatnim parametrem, jaki użytkownik może określić w interfejsie aplikacji jest rozdzielczość obrazu wyjściowego. W aplikacji udostępniono wybór rozdzielczości takich jak: 1920x1080 pikseli, 1280x720 pikseli oraz 640x360 pikseli. Umożliwia to użytkownikowi dostosowanie rozmiaru wyświetlanego okna do jego potrzeb. Po kliknięciu zielonego przycisku na dole okna interfejsu uruchomiony zostaje program detekcji. Widok interfejsu użytkownika został przedstawiony na rysunku nr 20.



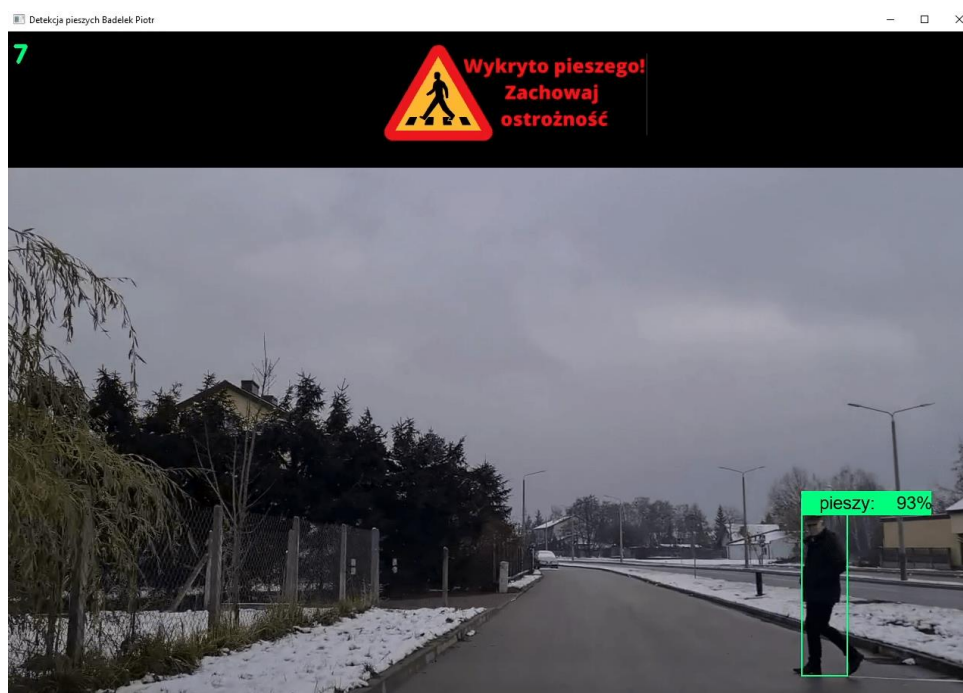
Rysunek 20 Interfejs użytkownika aplikacji do detekcji pieszego na jezdni. Źródło: Opracowanie własne

Po uruchomieniu programu detekcji, użytkownikowi zostaje wyświetlony obraz wyjściowy. W lewym górnym rogu aplikacji wyświetlana jest liczba kadrów na sekundę, jako miara prędkości wyświetlania obrazu wyjściowego. Wykryta przez program sylwetka pieszego jest zakreślana ramką ograniczającą w kolorze zielonym. Nad sylwetką dodatkowo zostaje wyświetlona nazwa klasy detekcji oraz wartość prawdopodobieństwa detekcji, zwracana przez algorytm sieci neuronowej. Widok obrazu wyjściowego w programie detekcji został przedstawiony na rysunku nr 21.



Rysunek 21 Widok obrazu wyjściowego programu do detekcji pieszego na jezdni. Źródło: Opracowanie własne.

Przy zdarzeniu wykrycia pieszego z prawdopodobieństwem wynoszącym równo lub powyżej 80%, na ekranie aplikacji zostaje wyświetlona grafika informująca o wykryciu pieszego. W celu dodatkowej informacji użytkownika o wysokim prawdopodobieństwie wykrycia pieszego, zostaje wywołany systemowy dźwięk alarmu. Opisany przypadek został zaprezentowany na rysunku nr 22.



Rysunek 22 Widok obrazu wyjściowego programu detekcji przy zdarzeniu wykrycia pieszego z prawdopodobieństwem równym lub większym od 80%. Źródło: Opracowanie własne.

(strona celowo zostawiona pusta)

7. Testy skuteczności detekcji

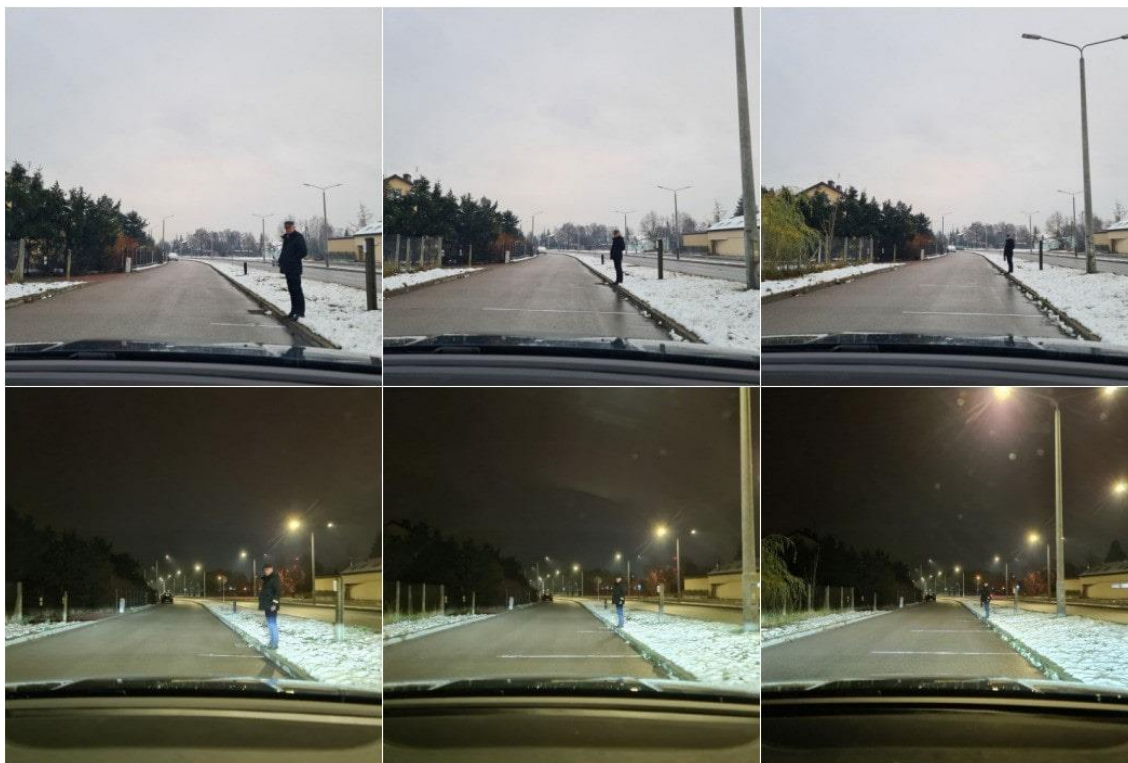
Przeprowadzone testy skuteczności detekcji pieszego na jezdni dla opracowanej aplikacji zostały wykonane na nagraniach, które były realizowane w kontrolowanych warunkach oraz w których występowały określone zmienne środowiskowe. Nagrania były rejestrowane za pomocą telefonu umieszczonego na statywie na desce rozdzielczej samochodu osobowego. Stworzone nagrania nie były wprowadzone do wcześniej utworzonej bazy danych próbek uczących oraz próbek testujących służących procesowi uczenia modelu sieci neuronowej.

7.1 Metodologia testów

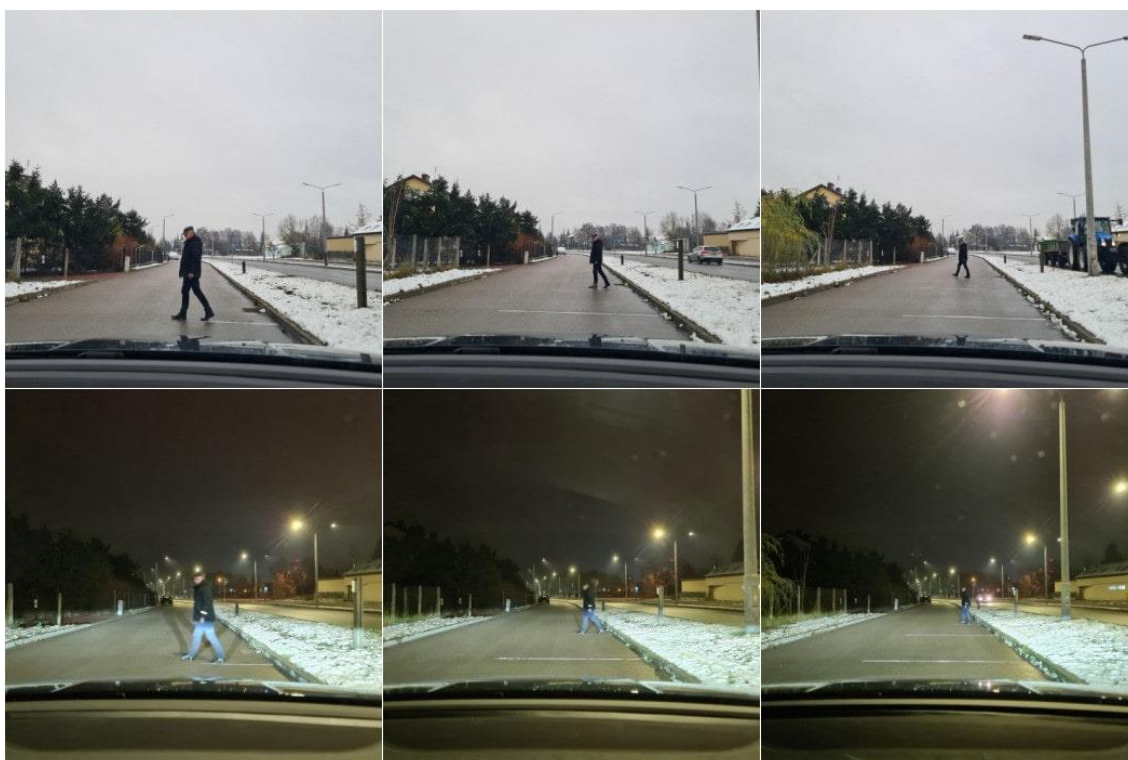
Na potrzeby przeprowadzenia testów skuteczności detekcji pieszego na jezdni poprzez uruchomioną aplikację opracowano nagrania, w których przyjęto następujące założenia:

- Nagrania były prowadzone w tym samym miejscu w terenie zabudowanym.
- W nagraniach zadbano o występowanie jednego pieszego, bez udziału przypadkowych przechodniów.
- Nagrania przeprowadzono w dzień oraz w nocy.
- Zarejestrowano nagrania w rozdzielczości 1920 x 1080 pikseli. Częstotliwość nagrań wynosiła 30 klatek na sekundę.
- Opracowane nagrania symulowały sytuację pieszego w pozycji stojącej oraz pieszego w pozycji kroczącej.
- Odcinek testowy pasa jezdni został oznaczony taśmami w miejscach, w których odległość kamery umieszczonej w samochodzie od pieszego wynosiła kolejno 5, 10 oraz 15 metrów.
- Nagrania przeprowadzono dla prędkości pojazdu równej 20 [km/h]. Wyższe wartości prędkości pojazdu nie były testowane.

Widok pieszego w pozycji stojącej oraz kroczącej, pochodzące z opracowanych testów przedstawiono na zdjęciach odpowiednio nr 5 i 6.



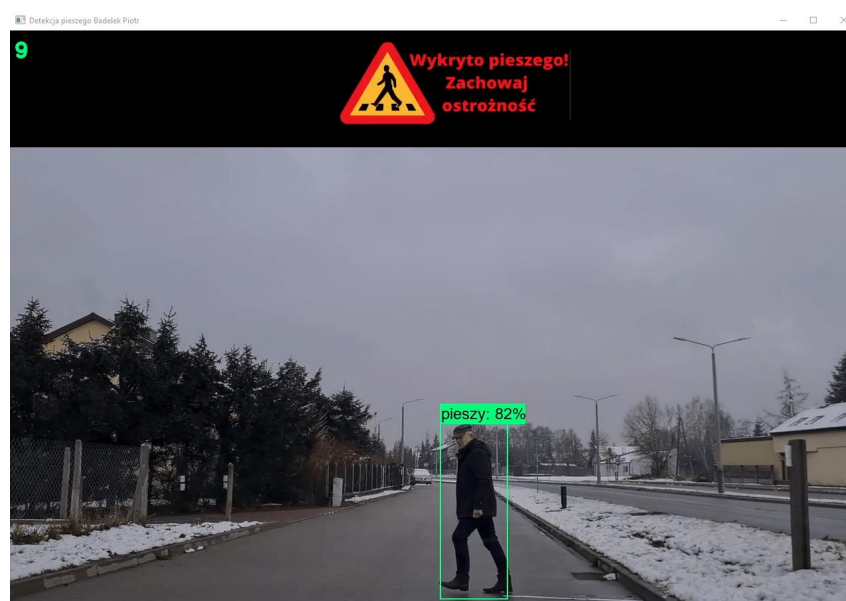
Zdjęcie 5 Widok pieszego w pozycji stojącej porą dzienną oraz nocną. Źródło: Opracowanie własne.



Zdjęcie 6 Widok pieszego w pozycji kroczącej porą dzienną oraz nocną. Źródło: Opracowanie własne.

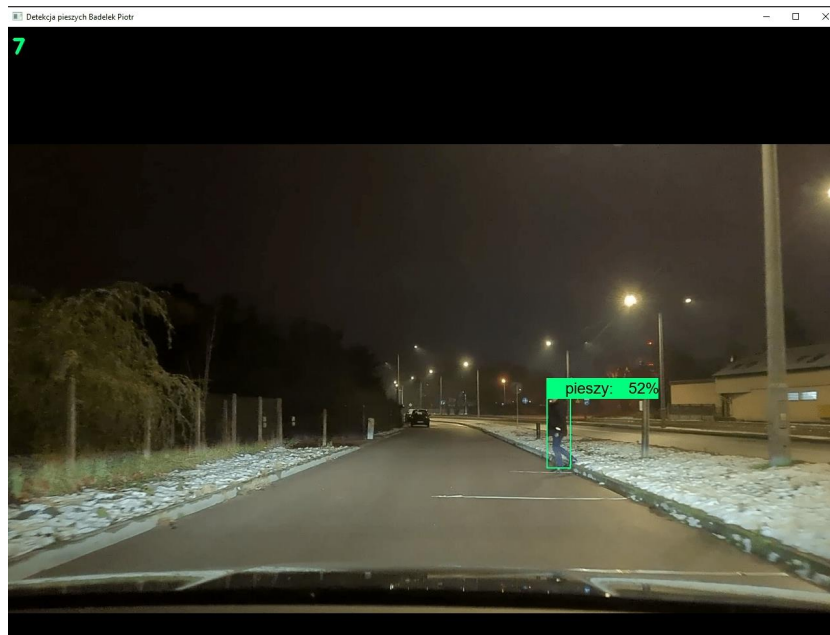
7.2 Wyniki testów

Dla każdego zdarzenia testowego przeprowadzono 3 próby detekcji pieszego. Pierwszym testowanym modelem sieci neuronowej zaimplementowanym w aplikacji oraz wyuczonym poprzez opracowany zbiór danych obrazowych był model SSD MobileNetv1 640x640. Model zwraca na wyjściu parametry ramek ograniczających. Model SSD MobileNet v1 przy zdarzeniu pieszego w pozycji kroczącej porą dzienną w odległości 5 metrów od kamery zwrócił średnią wartość prawdopodobieństwa detekcji wynoszącą 73%. Jest to najwyższa wartość prawdopodobieństwa detekcji wśród wszystkich odległości przy których model był testowany. Na rysunku nr 23 przedstawiono opisane zdarzenie.



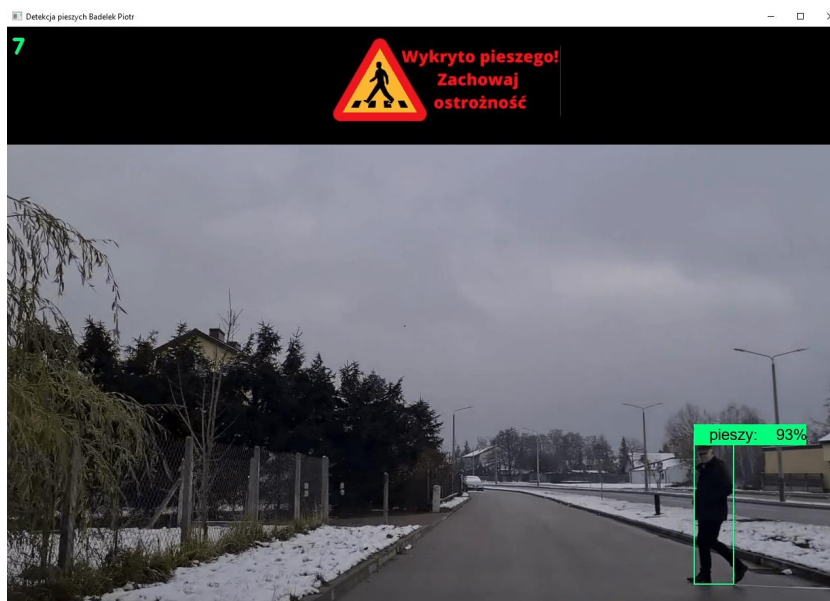
Rysunek 23 Widok obrazu wyjściowego programu detekcji z użyciem modelu SSD MobileNet v1 przy zdarzeniu pieszego w pozycji stojącej w odległości 5 metrów od kamery. Źródło: Opracowanie własne.

Model sieci SSD MobileNet v1 uzyskał najniższą wartość prawdopodobieństwa wykrycia sylwetki pieszego w przypadku testów realizowanych porą nocną w odległości 15 metrów od kamery. Średnie prawdopodobieństwo wykrycia sylwetki pieszego przez model wynosiło dla pozycji kroczącej oraz stojącej 52%. Mimo niskiej wartości pewności detekcji, ramki ograniczające zostały prawidłowo naniesione na sylwetkę pieszego. Przykładowa próba detekcji przez model sieci SSD MobileNet v1 porą nocną w odległości 15 metrów została przedstawiona na rysunku nr 24.



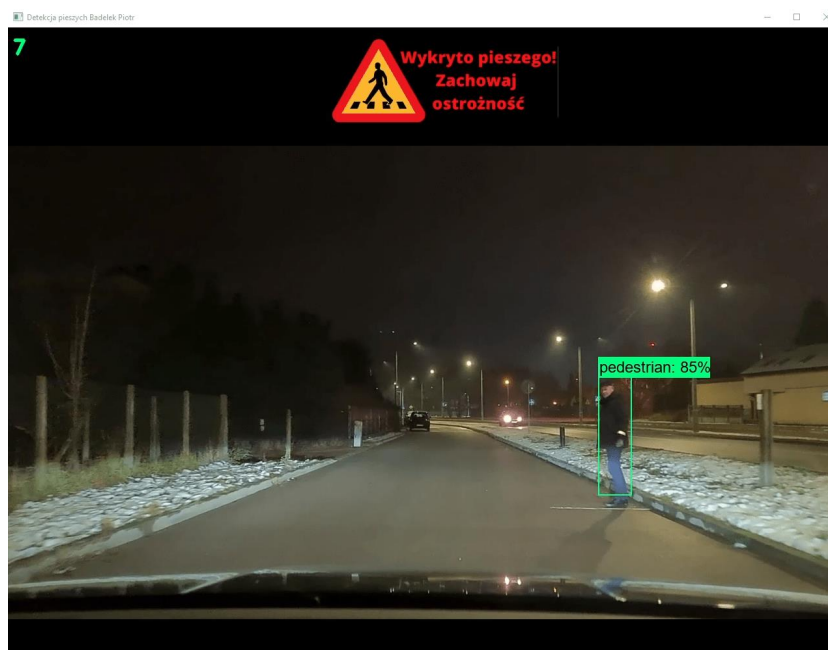
Rysunek 24 Widok obrazu wyjściowego programu detekcji z użyciem modelu SSD MobileNet v1 przy zdarzeniu pieszego w pozycji stojącej w odległości 15 metrów od kamery. Źródło: Opracowanie własne.

Następnym testowanym modelem sieci neuronowej był model SSD EfficientDet D0 512x512. Największą wartość prawdopodobieństwa detekcji uzyskał on w przypadku pieszego w pozycji kroczącej porą dzienną, w odległości 5 metrów od kamery umiejscowionej w poruszającym się pojeździe. Średnia wartość prawdopodobieństwa detekcji pieszego dla opisanego przypadku wyniosła 90%. Przykład opisanego zdarzenia został przedstawiony na rysunku nr 25.



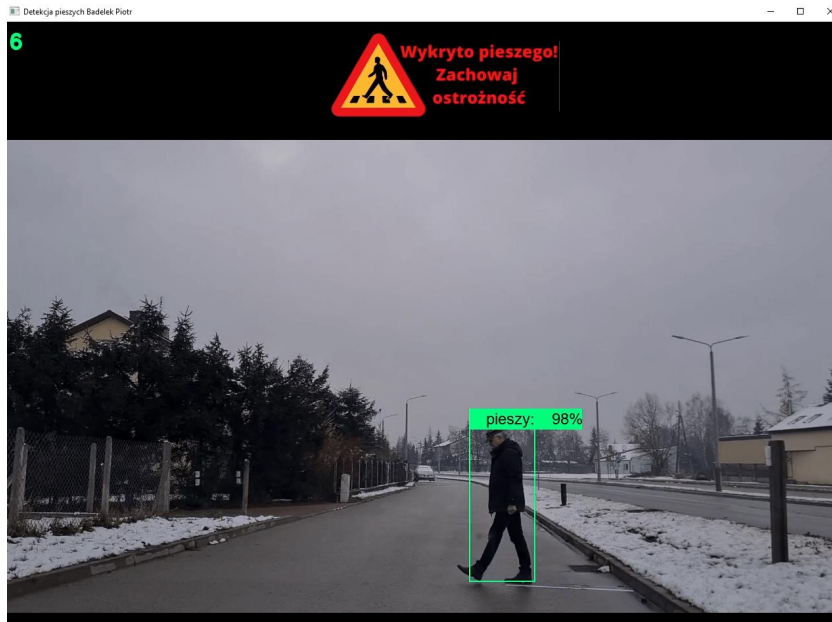
Rysunek 25 Widok obrazu wyjściowego programu detekcji z użyciem modelu SSD EfficientDet przy zdarzeniu pieszego w pozycji kroczącej w odległości 5 metrów od kamery. Źródło: Opracowanie własne.

Pomimo wyższych wartości prawdopodobieństwa detekcji przez model SSD EfficientDet D0 512x512, podczas realizowanych testów zaobserwowano ucinanie kształtów sylwetki pieszego. Na rysunku nr 26 przedstawiono zdarzenie naniesienia ramki ograniczającej na niepełny kształt sylwetki pieszego.



Rysunek 26 Zaobserwowane nanoszenie ramek ograniczających modelu SSD EfficientDet D0 512x512 na niepełny kształt sylwetki pieszego. Źródło: Opracowanie własne.

Ostatnim algorytmem dla którego przeprowadzono testy skuteczności detekcji w opracowanej aplikacji był model Faster-RCNN v1 640x640. Algorytm uzyskał najwyższe wartości prawdopodobieństwa detekcji w przypadku wykrycia pieszego w odległości 5 metrów w pozycji kroczącej porą dzienną. Średnia wartość prawdopodobieństwa detekcji wyniosła wówczas 97%. Na rysunku nr 27 przedstawiono widok programu detekcji dla opisanej próby testowej.



Rysunek 27 Widok obrazu wyjściowego programu detekcji z użyciem modelu Faster-RCNN przy zdarzeniu pieszego w pozycji kroczącej w odległości 5 metrów od kamery. Źródło: Opracowanie własne.

Dla każdego przypadku testowego, model Faster-RCNN v1 640x640 wykazał większą pewność detekcji od pozostałych modeli. Od modelu sieci SSD MobileNet v1 640x640, model Faster RCNN uzyskał średnią wartość prawdopodobieństwa detekcji większą o 23 punkty procentowe. Uzyskane średnie wartości prawdopodobieństw detekcji pieszego w pozycji stojącej oraz kroczącej zestawiono ze sobą kolejno w tabeli nr 2 i 3.

Tabela 2. Zestawienie wyników wartości prawdopodobieństw detekcji pieszego w pozycji stojącej przez użyte modele sieci neuronowych. Źródło: Opracowanie własne.

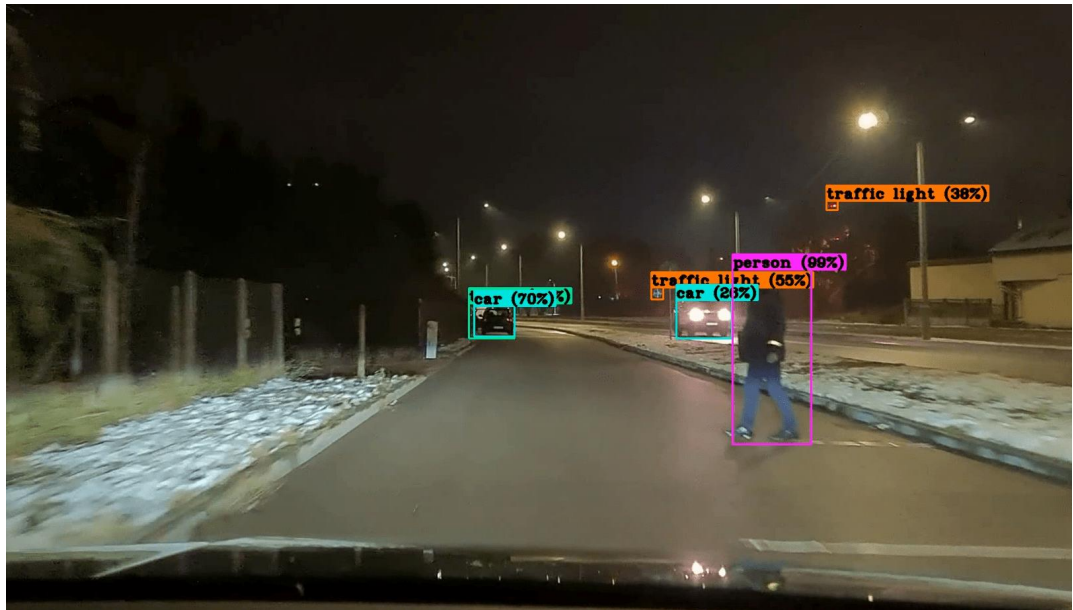
Lp.	SSD EfficientDet D0 512x512	Faster-RCNN v1 640x640	SSD MobileNet v1 640x640
Prawdopodobieństwo wykrycia pieszego w pozycji stojącej porą dzienną w odległości 5 [m]			
1	89%	91%	73%
2	87%	93%	71%
3	89%	92%	75%
Prawdopodobieństwo wykrycia pieszego w pozycji stojącej porą nocną w odległości 5 [m]			
1	85%	89%	63%
2	78%	86%	64%
3	76%	91%	56%
Prawdopodobieństwo wykrycia pieszego w pozycji stojącej porą dzienną w odległości 10 [m]			
1	80%	89%	66%
2	76%	90%	63%
3	77%	88%	62%
Prawdopodobieństwo wykrycia pieszego w pozycji stojącej porą nocną w odległości 10 [m]			
1	75%	81%	55%
2	71%	81%	53%
3	75%	82%	54%
Prawdopodobieństwo wykrycia pieszego w pozycji stojącej porą dzienną w odległości 15 [m]			
1	72%	83%	58%
2	73%	80%	53%
3	72%	84%	55%
Prawdopodobieństwo wykrycia pieszego w pozycji stojącej porą nocną w odległości 15 [m]			
1	70%	71%	51%
2	64%	74%	53%
3	64%	68%	52%

Tabela 3. Zestawienie wyników wartości prawdopodobieństw detekcji pieszego w pozycji kroczącej przez użyte modele sieci neuronowych. Źródło: Opracowanie własne.

Lp.	SSD EfficientDet D0 512x512	Faster-RCNN v1 640x640	SSD MobileNet v1 640x640
Prawdopodobieństwo wykrycia pieszego w pozycji kroczącej porą dzienną w odległości 5 [m]			
1	93%	98%	82%
2	90%	95%	81%
3	88%	97%	82%
Prawdopodobieństwo wykrycia pieszego w pozycji kroczącej porą nocną w odległości 5 [m]			
1	83%	87%	64%
2	83%	85%	62%
3	81%	86%	64%
Prawdopodobieństwo wykrycia pieszego w pozycji kroczącej porą dzienną w odległości 10 [m]			
1	85%	91%	72%
2	84%	86%	67%
3	85%	91%	67%
Prawdopodobieństwo wykrycia pieszego w pozycji kroczącej porą nocną w odległości 10 [m]			
1	78%	85%	60%
2	77%	85%	59%
3	78%	81%	61%
Prawdopodobieństwo wykrycia pieszego w pozycji kroczącej porą dzienną w odległości 15 [m]			
1	78%	85%	57%
2	70%	78%	56%
3	72%	85%	51%
Prawdopodobieństwo wykrycia pieszego w pozycji kroczącej porą nocną w odległości 15 [m]			
1	70%	71%	51%
2	64%	74%	53%
3	64%	68%	52%

7.3 Porównanie uzyskanych wyników z modelem YOLO

Po przeprowadzeniu testów skuteczności algorytmów zaimplementowanych w opracowanej aplikacji, wykonano dodatkowe testy skuteczności detekcji pieszego przez gotowy model sieci neuronowej. Wybrano gotowy program detekcji z siecią You Only Look Once, która uzyskuje według opinii specjalistów bardzo dobre wyniki detekcji. Jego czwarta odsłona została uznana za najszybszy model wykrywania obiektów w czasie rzeczywistym [26]. Użyta w programie sieć YOLOv4, została wytrenowana na 120000 obrazach w zbiorze treningowym, 5000 obrazach w zbiorze walidacyjnym oraz 41000 obrazach w zbiorze testowym [27]. Sieć wytrenowana jest do detekcji 80 różnych klas, dostępnych ze zbioru Coco Dataset. Na rysunku 28 zaprezentowano działanie detekcji obiektów z wykorzystaniem sieci YOLOv4.



Rysunek 28 Widok obrazu wyjściowego programu detekcji obiektów sieci Yolov4.

Źródło: Opracowanie własne.

W tabeli 4 zestawiono wyniki prawdopodobieństwa detekcji modelu sieci Yolov4 oraz wartości prawdopodobieństw detekcji algorytmów zaimplementowanych w aplikacji. Model You only look once osiągnął średnią wartość prawdopodobieństwa detekcji większą o 11 punktów procentowych, niż model z najlepszymi wynikami w aplikacji – Faster-RCNN v1 640x640.

Tabela 4. Zestawienie wyników średnich wartości prawdopodobieństw detekcji pieszego wraz z modelem sieci

Yolov4. Źródło: Opracowanie własne.

Odległość [m]	SSD EfficientDet D0 512x512	Faster-RCNN v1 640x640	SSD MobileNet v1 640x640	Yolov4
Średnia wartość prawdopodobieństwa wykrycia pieszego w pozycji stojącej porą dzienną				
5	88%	92%	73%	99%
10	78%	89%	64%	99%
15	72%	82%	55%	94%
Średnia wartość prawdopodobieństwa wykrycia pieszego w pozycji stojącej porą nocną				
5	80%	89%	61%	96%
10	74%	81%	54%	91%
15	66%	71%	52%	77%
Średnia wartość prawdopodobieństwa wykrycia pieszego w pozycji kroczącej porą dzienną				
5	90%	97%	82%	100%
10	85%	89%	69%	97%
15	73%	83%	55%	99%
Średnia wartość prawdopodobieństwa wykrycia pieszego w pozycji kroczącej porą nocną				
5	82%	86%	63%	99%
10	78%	84%	60%	95%
15	66%	71%	52%	98%

(strona celowo zostawiona pusta)

Podsumowanie

Wynikiem przedstawionej pracy dyplomowej jest projekt aplikacji do detekcji pieszego na jezdni z wykorzystaniem sieci neuronowej. Aplikacja umożliwia wykrywanie pieszego po wcześniejszym zdefiniowaniu przez użytkownika aplikacji: źródła detekcji, algorytmu detekcji oraz rozdzielczości obrazu wyjściowego.

W pracy przedstawiłem współczesne narzędzia używane do analizy obrazu z wykorzystaniem sieci neuronowych. Źródła, z których zaczerpnąłem wykorzystane w pracy rozwiązania, w przeważającej części są anglojęzyczne. Podczas tworzenia pracy zauważyłem, że wśród polskich pozycji literatury oraz wpisów na stronach internetowych, jest wciąż niewiele materiałów opisujących implementację nowych rozwiązań do analizy obrazu z wykorzystaniem sztucznej inteligencji.

Przedstawiona w pracy analiza narzędzi do przetwarzania obrazów z wykorzystaniem sieci neuronowych doprowadziła do wyboru kilku najpopularniejszych rozwiązań. Narzędzia takie jak biblioteki OpenCV lub TensorFlow mają dzięki swojej rosnącej popularności liczne poradniki oraz materiały edukacyjne.

Wykonane na potrzeby pracy testy skuteczności detekcji wykazały, iż optymalnym modelem sieci neuronowej do wykorzystania w opracowanej aplikacji jest model Faster RCNN v1 640x640. Uzyskał on średnią wartość prawdopodobieństwa detekcji na poziomie 84%, o 7 punktów procentowych więcej niż model sieci SSD EfficientDet D0 512x512. Najmniej optymalnym modelem sieci w opracowanej aplikacji był model SSD MobileNet v1 640x640. Sieć ta uzyskiwała w testach średnią wartość prawdopodobieństwa detekcji równą 62%. Wykonane nagrania testowe wykorzystałem również do sprawdzenia skuteczności gotowego programu detekcji. Postanowiłem wybrać algorytm sieci YOLOv4, ponieważ według dostępnych w sieci testów prezentuje on bardzo dobre wyniki skuteczności detekcji. W przypadku testowych nagrań średnie prawdopodobieństwo detekcji pieszego wyniosło 95%. W celu poprawienia skuteczności detekcji należałoby użyć większego zestawu danych obrazowych, który uwzględniałby więcej pozycji pieszego oraz różne warunki środowiskowe.

Podczas wykonywania testów zauważyłem, że każdy z algorytmów zaimplementowanych w aplikacji radzi sobie lepiej z detekcją pieszego w pozycji kroczącej, niż kiedy pieszy stoi. Średnia wartość prawdopodobieństwa detekcji pieszego w pozycji kroczącej była o 4 punkty procentowe wyższa niż w przypadku detekcji pieszego w pozycji stojącej. Może to wynikać z mniejszego pola, które pieszy zajmuje na analizowanych obrazach w pozycji stojącej,

lub z charakterystycznej pozy w momencie wykonywania kroków, które są dla sieci neuronowej łatwiejsze do rozpoznania. Testy nocne charakteryzowały się mniejszą średnią wartością prawdopodobieństwa detekcji o 7 punktów procentowych niż testy dzienne. Spowodowane jest to przede wszystkim gorszymi warunkami oświetleniowymi podczas wykonywania testów. Oświetlenie pochodzące jedynie od lamp latarni ulicznych oraz świateł przejeżdżających samochodów osobowych powoduje powstawanie refleksów na szybie samochodu, oraz obiektywie kamery.

Przy wykonywaniu projektu aplikacji napotkałem na kilka problemów, które znacząco wydłużyły mój czas pracy. Jednym z nich była implementacja bibliotek Nvidia Cuda na komputerze z systemem Windows. Przedstawione biblioteki firmy Nvidia nie wykrywały karty graficznej wraz z aktualnymi sterownikami. Jednakże, na komputerze z systemem Linux udało mi się bez problemu zaimplementować opisywane narzędzie. Ostatecznie, zrezygnowałem z instalatora firmy Nvidia, a biblioteki Nvidia Cuda przeniosłem manualnie, śledząc poradniki użytkowników w sieci. Manualnie przenoszenie pojedynczych plików zadziałało, jednak nie jest to rozwiązanie optymalne do użycia przy rozbudowie aplikacji.

Istnieje wiele różnych rozwiązań optymalizujących oraz rozbudowujących działanie opracowanej aplikacji. Przykładowo, wymagane jest dodanie większej liczby równoległe działających wątków w programie detekcji, które poprawiłyby jego szybkość działania. Oprócz wywoływania wątkiem systemowego dźwięku alarmu planowane jest wywoływanie równoległym wątkiem obrazu ostrzegającego o wykryciu pieszego. Praca nad optymalizacją oprogramowania zakłada również poprawę działania aplikacji z wykorzystaniem samego procesora. Rozwiązanie to pomogłoby w przyszłości zaimplementować aplikację w urządzeniach przenośnych. Planowane są również prace nad rozwojem testów aplikacji oraz sprawdzenie skuteczności detekcji przy występowaniu zdarzeń takich jak detekcja kilku pieszych jednocześnie, oraz detekcja dzieci. Dodatkowo należy sprawdzić skuteczność detekcji przy większych prędkościach pojazdu niż 20 km/h.

Bibliografia

1. McCulloch W., Pitts W., *A Logical Calculus of Ideas Immanent in Nervous Activity*, "The bulletin of mathematical biophysics", 1943, 115-133.
2. Géron A., *Introduction to artificial neural networks, Hands-On Machine Learning with Scikit-Learn and TensorFlow. Concepts, Tools and Techniques to Build Intelligent Systems*, O'Reilly Media, 2018, 255-261.
3. Ridder D., Duin R., Vliet L., *Nonlinear Image Processing Using Artificial Neural Networks*, „Advances in Imaging and Electron Physics”, 2003, 351-450.
4. Tadeusiewicz R., Szaleniec M., *Leksykon sieci neuronowych.*, Wydawnictwo Fundacji „Projekt Nauka”, 2015, 124-125.
5. <https://aigeekprogrammer.com/pl/konwolucyjne-sieci-neuronowe-klasyfikacja-obrazow-czesc-2/>
6. <https://home.agh.edu.pl/~horzyk/lectures/ai/SztucznaInteligencja-UczenieG%c5%82%c4%99bokichSieciNeuronowych.pdf>
7. Szeliga M., *Proces uczenia, Praktyczne uczenie maszynowe*, Wydawnictwo PWN, 2019, 78-84.
8. Convolutional neural networks: an overview and application in radiology | Insights into Imaging | Full Text (springeropen.com)
9. <https://docs.opencv.org/master/index.html>
10. <https://towardsdatascience.com/introduction-on-tensorflow-2-0-bd99eebcdad5>
11. <https://www.tensorflow.org/overview>
12. https://www.tutorialspoint.com/pytorch/pytorch_introduction.html
13. <https://pytorch.org/>
14. <https://medium.com/@ODSC/overview-of-the-yolo-object-detection-algorithm-7b52a745d3e0>
15. <https://paperswithcode.com/dataset/coco>
16. <https://cocodataset.org/#explore>
17. <https://towardsdatascience.com/how-to-work-with-object-detection-datasets-in-coco-format-9bf4fb5848a4>
18. <https://www.analyticsvidhya.com/blog/2020/03/google-colab-machine-learning-deep-learning/>
19. <https://medium.com/@ODSC/overview-of-the-yolo-object-detection-algorithm-7b52a745d3e0>

20. <https://appsilon.com/object-detection-yolo-algorithm/>
21. <https://towardsdatascience.com/understanding-ssd-multibox-real-time-object-detection-in-deep-learning-495ef744fab>
22. <https://jonathan-hui.medium.com/ssd-object-detection-single-shot-multibox-detector-for-real-time-processing-9bd8deac0e06>
23. <https://towardsdatascience.com/efficientdet-scalable-and-efficient-object-detection-review-4472ffc34fd9>
24. Google AI Blog: EfficientDet: Towards Scalable and Efficient Object Detection (googleblog.com)
25. R-CNN, Fast R-CNN, Faster R-CNN, YOLO — Object Detection Algorithms | by Rohith Gandhi | Towards Data Science
26. <https://ichi.pro/pl/yolov4-kontra-yolov5-240922076354091>
27. <https://alexeyab84.medium.com/yolov4-the-most-accurate-real-time-neural-network-on-ms-coco-dataset-73adfd3602fe>

Załącznik nr 6
do „Szczegółowych zasad oraz harmonogramu
wykonywania prac dyplomowych na
Wydziale Mechatroniki, Uzbrojenia i Lotnictwa WAT”

Warszawa, 27.01.2022r.

Wojskowa Akademia Techniczna

Wydział Mechatroniki, Uzbrojenia i Lotnictwa

Imię i nazwisko studenta: Piotr Badetek

Numer albumu: 42318

OŚWIADCZENIE

Świadomy(a) odpowiedzialności karnej z tytułu naruszenia przepisów ustawy z dnia 1 lutego 1994 r. O prawie autorskim i prawach pokrewnych (Dz. U. z 2018 r. poz. 1191 z późn. zm.) oraz konsekwencji określonych w ustawie z dnia 20 lipca 2018 r. Prawo o szkolnictwie wyższym i nauce (Dz.U. z 2018 r. poz. 1668 z późn. zm.)¹⁾, a także odpowiedzialności cywilnoprawnej oświadczam, że przedkładana praca dyplomowa pt.

„Wykorzystanie sieci neuronowej do analizy obrazu w celu
detekcji pęknięcia na jezdnii”

została napisana przeze mnie samodzielnie i nie była wcześniej podstawą żadnej innej urzędowej procedury związanej z nadaniem dyplomu uczelni lub tytułów zawodowych. Jednocześnie oświadczam, że wyżej wymieniona praca dyplomowa nie narusza praw autorskich w rozumieniu ustawy o prawie autorskim i prawach pokrewnych innych osób oraz dóbr osobistych chronionych prawem cywilnym. Wszystkie informacje umieszczone w pracy, uzyskane ze źródeł pisanych i elektronicznych oraz inne informacje, zostały udokumentowane w wykazie literatury odpowiednimi odnośnikami.

27.01.2022 Badetek
/data, podpis studenta/

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

Art. 77 ust. 5. „W przypadku gdy w pracy dyplomowej stanowiącej podstawę nadania tytułu zawodowego osoba ubiegająca się o ten tytuł przypisała sobie autorstwo istotnego fragmentu lub innych elementów cudzego utworu lub ustalenia naukowego, rektor, w drodze decyzji administracyjnej, stwierdza nieważność dyplomu”.

Wyrażam zgodę na udostępnienie mojej pracy dyplomowej przez Archiwum WAT w czytelni i w ramach wypożyczeń międzybibliotecznych.

27.01.2022 Badetek
/data, podpis studenta/