

Politechnika Warszawska

W Y D Z I A Ł M E C H A T R O N I K I



Praca dyplomowa magisterska

na kierunku Automatyka, Robotyka i Informatyka Przemysłowa

Ocena jakości detekcji obiektów przez modele sieci neuronowej, uczone
przy pomocy danych pochodzących ze środowiska wirtualnego

numer pracy według wydziałowej ewidencji prac: 114D-MSP-IP/253293/1212213

Mateusz Tadeusz Kawecki

numer albumu 253293

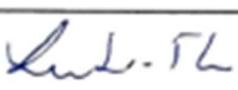
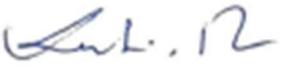
promotor
dr inż. Anna Sztynber

konsultacje

—

WARSZAWA 2021

**PRACA DYPLOMOWA
magisterska**

| | |
|---|--|
| <u>Specjalność:</u> | Robotyka |
| <u>Instytut prowadzący specjalność:</u> | Instytut Automatyki i Robotyki |
| <u>Instytut prowadzący pracę:</u> | Instytut Automatyki i Robotyki |
| <u>Temat pracy:</u> Ocena jakości detekcji obiektów przez modele sieci neuronowej, uczone przy pomocy danych pochodzących ze środowiska wirtualnego. | |
| <u>Temat pracy (w jęz. ang.):</u> Assessment of the quality of object detection by neural network models, trained with data from the virtual environment. | |
| <u>Zakres pracy:</u> | |
| 1. Zgłębienie zagadnienia detekcji obiektów na obrazach, z wykorzystaniem CNN. 2. Analiza metod pozyskiwania danych uczących ze środowisk wirtualnych. 3. Pozyskanie reprezentatywnych danych ze środowiska rzeczywistego i wirtualnego. 4. Zaprojektowanie i zaprogramowanie sieci neuronowej do zadań detekcji. 5. Opracowanie miarodajnej metody porównawczej modeli. 6. Opracowanie wniosków końcowych | |
| <u>Podstawowe wymagania:</u> | |
| 1. Umiejętność programowania w języku Python. 2. Znajomość framework-ów uczenia maszynowego (Keras, Tensorflow). 3. Umiejętność projektowania i optymalizowania sieci neuronowych. | |
| <u>Literatura:</u> | |
| 1. Géron A. „Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow.”, 2019 2. Wu B., Wan A., Yue X. i Keutzer K. „SqueezeSeg: Convolutional Neural Nets with Recurrent CRF for Real-Time Road-Object Segmentation from 3D LiDAR Point Cloud”, IEEE International Conference on Robotics and Automation (ICRA), 2018 | |
| <u>Słowa kluczowe:</u> | |
| Praca dyplomowa jest realizowana we współpracy z przemysłem? | |
| Tak/Nie * | |
| <u>Imię i nazwisko dyplomanta:</u> Mateusz Kawecki | <u>Imię i nazwisko promotora:</u> dr inż. Anna Sztyber |
| <u>Imię i nazwisko konsultanta:</u> | |
| <u>Temat wydano dnia:</u> 03.02.2021r. | <u>Termin ukończenia pracy:</u> |
| <u>Zatwierdzenie tematu</u> | |
|  Opiekun specjalności |  Z-ca Dyrektora Instytutu odpowiedzialny za sprawy dydaktyczne |

Streszczenie

Tytuł pracy: Ocena jakości detekcji obiektów przez modele sieci neuronowej, uczone przy pomocy danych pochodzących ze środowiska wirtualnego.

Słowa kluczowe: uczenie maszynowe, głębokie sieci neuronowe, sieci konwolucyjne, analiza obrazu, detekcja obiektów, rozpoznawanie znaków drogowych.

Przedmiotem niniejszej pracy dyplomowej jest ocena jakości modeli sztucznych sieci neuronowych, trenowanych na danych pochodzących z symulacji rzeczywistości, takich jak gry komputerowe, czy programy modelarskie. Pierwsze rozdziały pracy przybliżają poruszony problem oraz rozwijają uzasadnienie wartości jego rozwiązania. Można się z nich dowiedzieć, że rzeczywistość wirtualna może stanowić łatwo dostępne i tanie źródło danych potrzebnych do trenowania modeli, których zadaniem jest klasyfikacja i detekcja obiektów. W dalszej części przyjrzano się podobnym problemom rozwiązywanym przez zespoły naukowe na świecie, opisano ich cele, motywy i wyniki. Kolejne rozdziały wykładają wiedzę teoretyczną niezbędną do rzetelnego przeprowadzenia eksperymentu. Na początku omówione zostało uczenie maszynowe, w swym obszernym znaczeniu, by następnie skupić się na sieciach neuronowych, z szczególnym wyróżnieniem sieci konwolucyjnych. Zaprezentowano różne architektury sieci, które potencjalnie mogłyby zostać wykorzystane w eksperymencie, mowa o sieciach ResNet oraz YOLO w dwóch wersjach. Ponieważ do przeprowadzenia konstruktywnej oceny wymagane są uniwersalne miary, zostały one opisane w kilku wariantach. Następna część dotyczy opisu eksperymentu. Zawiera informacje o pochodzeniu danych uczących. Obiektem na jakie zdecydowano się, by były przedmiotem rozpoznawania przez model na obrazie, są znaki drogowe. Następnie opisano technologie wykorzystane w procesie trenowania, wraz z jego parametrami.

Ocena wyników pracy została przeprowadzona na drodze porównania modelu uczonego na obrazach zaczerpniętych ze środowiska wirtualnego, jakim była gra komputerowa, z modelem uczonym na obrazach rzeczywistych. Następnie dokonano porównania jakości detekcji tego samego zbioru obrazów testowych przez oba modele. Wyniki tego porównania oraz wnioski ogólne przedstawiono w ostatniej części pracy.

Abstract

Paper title: Assessing the quality of object detection by neural network models learned using data from a virtual environment.

Keywords: machine learning, deep neural networks, convolutional networks, image analysis, object detection, traffic sign recognition.

The object of this thesis is to evaluate the quality of artificial neural network models trained on data coming from reality simulations, such as computer games or modeling programs. The first chapters of the thesis introduce the problem addressed and develop a justification of the value of its solution. One can learn from them that virtual reality can provide a readily available and inexpensive source of data needed to train models tasked with object classification and detection. The following section looks at similar problems solved by research teams around the world and describes their goals, motives, and results. Subsequent chapters lecture the theoretical knowledge necessary for reliable experimentation. First, machine learning, in its broad sense, is discussed, to then focus on neural networks, with special emphasis on convolutional networks. Different network architectures that could potentially be used in the experiment are presented, the ResNet and YOLO (in two versions) networks are mentioned. Since universal measures are required for constructive evaluation, they are described in several variations. The next section deals with the description of the experiment. It contains information about the origin of the learning data. The objects chosen to be the subject of recognition by the model in the image are traffic signs. Then the technologies used in the training process are described, along with its parameters.

The evaluation of the results of the work was carried out by comparing the model learned on images taken from a virtual environment, such as a computer game, with the model learned on real images. A comparison of the detection quality of the same set of test images by both models was then made. The results of this comparison and general conclusions are presented in the last section of the thesis.



Politechnika Warszawska

Warszawa, 14.12.21r.....
miejscowość i data

Mateusz Kawecki
imię i nazwisko studenta

253293.....
numer albumu

Automatyka, Robotyka, Informatyka przemysłowa
kierunek studiów

OŚWIADCZENIE

Świadomy/-a odpowiedzialności karmej za składanie fałszywych zeznań oświadczam, że niniejsza praca dyplomowa została napisana przeze mnie samodzielnie, pod opieką kierującego pracą dyplomową.

Jednocześnie oświadczam, że:

- niniejsza praca dyplomowa nie narusza praw autorskich w rozumieniu ustawy z dnia 4 lutego 1994 roku o prawie autorskim i prawach pokrewnych (Dz.U. z 2021 r., poz. 1062) oraz dóbrosobistych chronionych prawem cywilnym,
- niniejsza praca dyplomowa nie zawiera danych i informacji, które uzyskałem/-am w sposób niedozwolony,
- niniejsza praca dyplomowa nie była wcześniej podstawą żadnej innej urzędowej procedury związanej z nadaniem dyplomów lub tytułów zawodowych,
- wszystkie informacje umieszczone w niniejszej pracy, uzyskane ze źródeł pisanych i elektronicznych, zostały udokumentowane w wykazie literatury odpowiednimi odnośnikami,
- znam regulacje prawne Politechniki Warszawskiej w sprawie zarządzania prawami autorskimi i prawami pokrewnymi, prawami własności przemysłowej oraz zasadami komercjalizacji.

Mateusz Kawecki
czytelny podpis studenta



Politechnika Warszawska

Warszawa, 14.12.21 r.....
miejscowość i data

Mateusz Kawecki.....
imię i nazwisko studenta

253293.....
numer albumu

Wydział Mechatroniki;
Automatyka, Robotyka, Informatyka przemysłowa...
Wydział i kierunek studiów

Oświadczenie studenta w przedmiocie udzielenia licencji
Politechnice Warszawskiej

Oświadczam, że jako autor/współautor* pracy dyplomowej pt. „*Ocena jakości detekcji obiektów przez modele sieci neuronowej, uczone przy pomocy danych pochodzących ze środowiska wirtualnego*” udzielam/nie udzielam* Politechnice Warszawskiej nieodpłatnej licencji na niewyłączne, nieograniczone w czasie, umieszczenie pracy dyplomowej w elektronicznych bazach danych oraz udostępnianie pracy dyplomowej w zamkniętym systemie bibliotecznym Politechniki Warszawskiej osobom zainteresowanym.

Licencja na udostępnienie pracy dyplomowej nie obejmuje wyrażenia zgody na wykorzystywanie pracy dyplomowej na żadnym innym polu eksploatacji, w szczególności kopiowania pracy dyplomowej w całości lub w części, utrwalania w innej formie czy zwielokrotniania.

Mateusz Kawecki
czytelny podpis studenta

* niepotrzebne skreślić

Spis treści:

| | | |
|--------|---|----|
| 1. | Cel i zakres prac | 9 |
| 2. | Wstęp | 11 |
| 3. | Przegląd istniejących rozwiązań..... | 13 |
| 4. | Uczenie maszynowe i jego rodzaje | 16 |
| 4.1. | Kryteria podziału ML..... | 16 |
| 4.2. | Problemy uczenia maszynowego | 19 |
| 4.3. | Klasyfikacja jako narzędzie ML | 21 |
| 5. | Sieci neuronowe | 22 |
| 5.1. | Proces uczenia głębokiej sieci neuronowej | 24 |
| 5.1.1. | Znikające i eksplodujące gradienty | 25 |
| 5.1.2. | Optymalizatory | 28 |
| 5.1.3. | Regularyzacja | 33 |
| 5.2. | Konwolucyjne sieci neuronowe | 36 |
| 5.3. | Architektury sieci neuronowych stosowane do rozpoznawania obiektów..... | 39 |
| 5.3.1. | ResNet | 40 |
| 5.3.2. | YOLO | 41 |
| 5.4. | Miary wydajności..... | 44 |
| 5.4.1 | Sprawdzian krzyżowy..... | 44 |
| 5.4.2. | Macierz pomyłek | 44 |
| 5.4.3. | Precyza a czułość modeli..... | 45 |
| 5.4.4. | Intersection over Union (IoU) | 46 |
| 5.4.5. | Funkcja straty (<i>Loss</i>) | 47 |
| 6. | Dane uczące i testujące - pozyskanie i opracowanie..... | 49 |
| 6.1. | Data mining | 49 |
| 6.2. | Znaki drogowe – obiekt detekcji | 49 |
| 6.3. | Pozyskanie danych uczących ze środowiska wirtualnego | 50 |
| 6.4. | Pozyskanie danych uczących ze środowiska rzeczywistego..... | 54 |
| 6.5. | Dane testujące | 56 |
| 7. | Trenowanie modeli | 57 |
| 7.1. | Serwis trenujący modele | 57 |
| 7.2. | Wykorzystane technologie | 59 |

| | |
|---|----|
| 7.3. Procedura trenowania..... | 59 |
| 8. Ocena modeli oraz porównanie wyników ich uczenia na danych ze środowiska rzeczywistego i wirtualnego | 63 |
| 9. Podsumowanie i wnioski..... | 72 |
| 10. Bibliografia | 73 |
| 11. Spis rysunków..... | 76 |

1. CEL I ZAKRES PRAC

Tematem niniejszej pracy jest porównanie precyzji klasyfikacji oraz detekcji znaków drogowych przez modele konwolucyjnej sieci neuronowej, uczone przy pomocy danych pochodzących ze środowiska rzeczywistego i wirtualnego. Ocena skuteczności obu modeli obrazuje wartość danych uczących model zaczerpniętych z symulatora wirtualnego jako substytutu obrazów pozyskanych w terenie, przy dużym nakładzie czasu i wysiłku. Celem jest zdefiniowanie w jakim stopniu możliwe jest posługiwanie się symulacją rzeczywistości przy fabrykowania obrazów uczących, odpowiadających obiektem czy zjawiskom ze świata rzeczywistego. Taka metoda pozyskiwania obrazów wyklucza konieczność tworzenia zbiorów uczących w oparciu o sfotografowane obiekty w terenie czy w studio, wystarczy praca przy komputerze. Do stworzenia takiej bazy wystarczyłoby szybkie, komputerowe wymodelowanie obiektu osadzonego w przestrzeni również odwzorowującej rzeczywistość. Problem pojawi się w momencie, gdy projekt wymusza dużą zmienność tła i oświetlenia obiektów, wówczas należałoby zaprojektować wirtualny symulator rzeczywistości zapewniający możliwość wprowadzania wielu zmiennych do środowiska wirtualnego. Stworzenie oprogramowania dla takiego symulatora, do wykonania jednego projektu okazałoby się niekiedy mniej opłacalne niż skrupulatna sesja w terenie. Na szczęście okazuje się, że wydajny obliczeniowo sprzęt, to nie jedyne co technologia uczenia maszynowego może zyskać dzięki gwałtownemu rozwojowi wirtualnego świata gier, ponieważ w opisany wyżej przypadku rozwiązaniem stają się gotowe symulatory rzeczywistości jakimi są gry komputerowe, stworzone w zupełnie innym celu, lecz starając się wiernie odwzorować nasz świat (rys. 1).



Rysunek 1 Współczesne gry komputerowe stanowią najwygodniejsze źródło takiej symulacji. Widok ekranu gry Microsoft Flight Simulator [1]

W celu dokonania rzetelnego porównania należy podążyć za przedstawionym poniżej planem eksperymentu, który w całości został opisany w dalszej części pracy:

- 1) Wybór obiektu do klasyfikacji - w ramach tego etapu należy zdecydować, które obiekty i zjawiska są łatwo dostępne do sfotografowania, jak również mogą być uwydawnione w grach i odwzorowane na poziomie najbliższym oryginałowi.
- 2) Wybór środowiska wirtualnego do pobrania danych - gdy został wybrany konkretny obiekt podlegający klasyfikacji, trzeba wybrać grę stanowiącą najlepsze źródło zdjęć tego obiektu. Kryteriami są najlepsze odwzorowanie obiektu i renderowanie na wysokim poziomie z uwzględnieniem zmiennych warunków oświetleniowych, oraz częsta występowanie danego obiektu w wybranym symulatorze,
- 3) Pozyskanie odpowiadających sobie danych ze środowiska rzeczywistego i symulatora - jest to etap najbardziej czasochłonny, wymaga zarówno wielu godzin spędzonych na fotografowaniu obiektów tak, by liczba zdjęć stanowiła obszerny materiał do nauki modelu, jak również czasu potrzebnego na zrobienie ujęć ze świata wirtualnego, co zgodnie z założeniem zajmie znacznie mniej czasu,
- 4) Przeprowadzenie obróbki danych oraz ich etykietowanie - prace nad zebranymi danymi są niekiedy złożone z wielu etapów zebranych w tzw. potoki. Zdjęcia należy zestandardyzować i ujednolicić ich cechy, a następnie zaetykietować. Operacje na zdjęciach odbywają się w sposób automatyczny i dlatego ten proces nie zajmuje dużo czasu, nawet jeśli zdjęcie jest kilka tysięcy, natomiast etykietowanie jest działaniem manualnym człowieka na każde zdjęcie osobno co pochłania dużo więcej czasu. Etykietowanie polega na wskazaniu na zdjęciu umiejscowienia obiektu, który chcemy, żeby w przyszłości podlegał klasyfikacji. Istnieją dedykowane programy do etykietowania przyśpieszające pracę (rys.2). Ich obsługa polega na ujęciu obiektu na zdjęciu w ramkę, a program automatycznie eksportuje etykietę dotyczącą danego zdjęcia. W tym projekcie wykorzystano program LabelImg, którego widok okna zamieszczono na zdjęciu poniżej.



Rysunek 2 Widok okna programu LabelImg, służącego do etykietowania obiektów [Źródło: opracowanie własne]

- 5) Tworzenie modeli ze zróżnicowaniem funkcji aktywacji oraz architektury sieci - zmiany tych parametrów często dają lepsze rezultaty, dlatego uczenie modeli w różnych wariantach jest nieodłącznym elementem rzetelnego poszukiwania najlepszego modelu,
- 6) Porównanie wyników miar jakości klasyfikacji - generowanie współczynników jakości przeprowadzanej predykcji przez najlepszy z modeli i porównanie ich celem podsumowania eksperymentu,
- 7) Podsumowanie i wnioski.

2. WSTĘP

Pojęcia takie jak *sztuczna inteligencja* czy *sięć neuronowa* funkcjonują powszechnie w świadomości dzisiejszego społeczeństwa, za sprawą powracającej fascynacji tym zagadnieniem. Wymienione terminy są znane od dekad, gdy powstawały pierwsze koncepcje sieci neuronowych inspirowane pracą mózgu, nasuwające wizje myślących maszyn. Starano się bezpośrednio przełożyć biologię na elektronikę układów logicznych, co w niedługim czasie obnażyło luźny związek między tymi dwoma światami i nakreśliło nowy kierunek badań - odchodzący od poszukiwania ścisłej analogii do funkcjonowania układu nerwowego [2]. Z

czasem pierwotne zainteresowanie sieciami neuronowymi malało przez granice możliwości technologicznych, by rozbudzać się na nowo, gdy te zostały przesuwane. Pojawia się więc pytanie - czy obecny entuzjazm związany ze sztuczną inteligencją jest tymczasowy, a ona niedługo okaże się mniej perspektywiczna niż może się obecnie wydawać? W dzisiejszych czasach spełnione są wszystkie warunki konieczne do szybkiego rozpowszechniania się zastosowań sztucznej inteligencji. Do najważniejszych należą: coraz większa moc obliczeniowa komputerów, dostęp do ogromnych zbiorów danych i fundamentalny dorobek naukowy w tej dziedzinie. Powszechna dostępność komputerów sprawiła, że skomplikowane zadania obliczeniowe, jak na przykład predykcja, prognozowanie, dotychczas wykonywane przez człowieka, przy wykorzystaniu żmudnych i czasochłonnych narzędzi matematycznych, takich jak rachunek prawdopodobieństwa, statystyka czy metody numeryczne, stały się łatwe i szybkie do rozwiązania, przy tym pozbawione ryzyka popełnienia błędu. Z wykorzystaniem sztucznej inteligencji te same zadania mogą odbywać się poza zupełną kontrolą człowieka, automatyzując procesy obliczeniowe, a nawet decyzyjne. Paliwem napędowym tej technologii jest dostęp do danych - informacji, na podstawie których system mógłby nauczyć się podejmować decyzje. Im więcej danych dostarczymy w procesie trenowania, tym dokładniejsze efekty pracy sztucznej inteligencji osiągniemy. Ważnym czynnikiem rozwijającym dziedzinę sieci neuronowych jest wspomniana moc obliczeniowa. Pierwsze komputery, będące świadkami narodzin koncepcji nie pozwalały na rozwiązywanie złożonych problemów, dlatego rozwój tej technologii dość szybko wyhamował. Obecnie możliwe jest przetwarzanie dużych ilości dostarczonych informacji w relatywnie niedługim czasie, co odbywa się wielowątkowo z wykorzystaniem szerokich zasobów pamięci podręcznej i szybkich procesorów. Przebieg procesu tworzenia modelu klasyfikatora w oparciu o tak rozległą bazę danych wymaga znacznej mocy obliczeniowej, którą dzisiaj dostarczają stworzone w zupełnie innym celu karty graficzne komputerów. Te dedykowane grom systemy, doskonale sprawdzają się pod względem wydajności podczas trenowania modeli sztucznej inteligencji. Środowisko deweloperów gier komputerowych znacząco się rozrasta, co można dostrzec nawet na rynku krajowym. Rosnące zapotrzebowanie na rozrywkę w postaci gier doprowadziło do gwałtownego rozwoju tej branży w ciągu ostatnich dwóch dekad w odniesieniu nie tylko do konkurencyjności, ale przede wszystkim do technologii jakiej te produkcje wymagają. Praca systemów obliczeniowych, obsługująca współczesne gry jest zoptymalizowana do pracy z obrazem wysokiej jakości, dlatego karty graficzne zostały z sukcesem zaadaptowane do rozwiązywania problemu niewystarczającej mocy obliczeniowej podczas prac nad głębokim widzeniem maszynowym. Na podstawie przedstawionych informacji oraz ciągłych inwestycji w rozwój sieci neuronowych i sztucznej inteligencji pojmowanej w ogóle, można spodziewać się, że jej wykorzystanie będzie cały czas rosło, ułatwiając pracę ludzi w wielu gałęziach przemysłu, handlu i in. [3] [4].

Kluczem do prawidłowego wytrenowania modelu, będącego w stanie dokonać predykcji czy klasyfikacji, jest odpowiednia baza danych. Pozyskiwanie danych a następnie ich przygotowanie jest etapem wymagającym największego skupienia i uwagi. Należy dokładnie przemyśleć problem, który jest do rozwiązania i zastanowić się jakie dane i w jakiej formie są

potrzebne, by proces trenowania nie przebiegał w niewłaściwym kierunku. Samo pozyskiwanie danych jest niekiedy bardzo czasochłonne i drogie, szczególnie w odniesieniu do dziedziny widzenia maszynowego, gdzie materiałem do nauki rozpoznawania obiektów są obrazy. Powstały dotychczas obszerne bazy obrazów odpowiednio skategoryzowanych i etykietowanych, do których dostęp jest swobodny. W wielu przypadkach podstawowego zastosowania sieci neuronowych, te dane są wystarczające, niestety, kiedy projekt staje się bardziej specjalizowany i zależy nam na rozpoznawaniu obiektu spoza istniejącej bazy danych, jesteśmy zmuszeni takową przygotować we własnym zakresie. Z pomocą często przychodzą zdjęcia wyszukiwane w intrenecie poprzez przeglądarkę, którą można dodatkowo uzbroić w rozszerzenie wspomagające pobieranie obrazów efektywniej. W sytuacji, gdy obiekty nie są wystarczająco rozpowszechnione w sieci, pozostaje nam samodzielne fotografowanie przedmiotu czy zjawiska, jest to zdecydowanie najbardziej żmudna metoda, szczególnie jeśli obiekty naszego zainteresowania są unikatowe, rozmieszczone w dużych odległościach, co wymaga podróżowania lub rzadko występujące jak zjawiska pogodowe i geologiczne. Jakość trenowanego modelu zależy od ilości dostarczonych próbek do przeanalizowania co sprawia, że chcąc uzyskać rzetelnie pracujący model należy przeznaczyć na jego trenowanie od kilkuset do kilkunastu tysięcy zdjęć, w zależności od ich złożoności.

3. PRZEGŁĄD ISTNIEJĄCYCH ROZWIĄZAŃ

Pierwsze próby zastosowania ML w branży gier opierały się na stworzeniu modeli funkcjonujących wyłącznie w jej obrębie. Najczęściej celem było optymalizowanie wyniku w ramach rozgrywki czy tworzenie botów. Uczone modele pomagały również szukać równowagi w poziomie trudności gry, by ta była wymagająca, jednak możliwa do ukończenia dostosowując się do umiejętności gracza. Uczenie modeli sieci CNN na podstawie obrazów zebranych z gry opisał Anirudh Venkatesh w 2018 r. tworząc system rozpoznawania konkretnych postaci w grze *Super Smash Bros. Melee*. Jest to prosta gra platformowa, umożliwiająca testowanie możliwości jakie daje uczenie maszynowe. Autor wykorzystał konwolucyjną sieć neuronową zaprojektowaną według swojego pomysłu, by nauczyć model rozpoznawać i śledzić cztery postaci w grze oraz stworzył bota samodzielnie poruszającego się w świecie gry. Jest to przykład pokazujący potencjał płynący z gier jako źródła danych aniżeli tylko źródła rozrywki [5].

W artykule naukowym pt. „*Obtain Datasets for Self-driving perception from Video Games automatically*” z 2018 roku chińscy badacze opisali metodę pozyskiwania danych z gry komputerowej o nazwie Grand Theft Auto V (GTAV), celem opracowania metody zdobywania informacji potrzebnych do nauki modeli sieci konwolucyjnych, które miałyby zastosowanie w

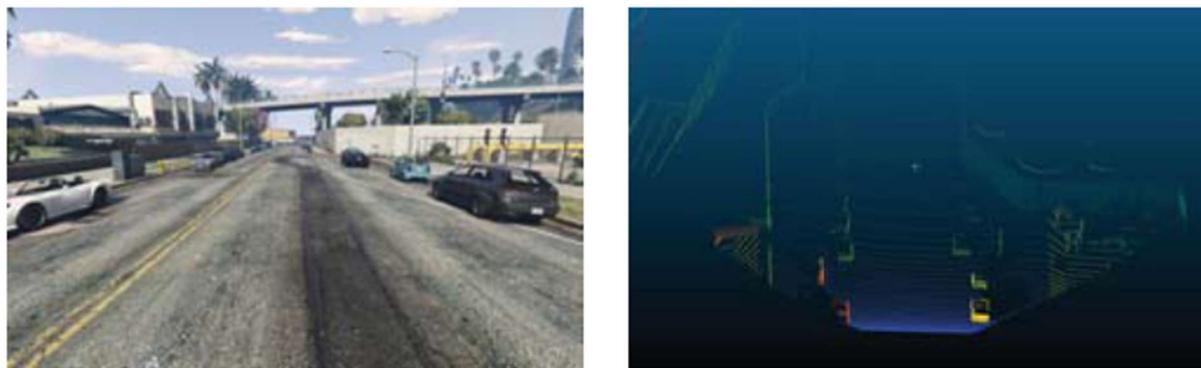
rzeczywistości, bez wysiłku człowieka. Wybrana przez nich gra charakteryzuje się wysokiej jakości odwzorowaniem detali świata rzeczywistego, dodatkowo jest to wirtualny świat zachowujący aspekty rzeczywistej fizyki i co najważniejsze oświetlenia. W artykule podkreślono jak kosztowne jest ręczne przetwarzanie danych – dostęp do nich pochłania środki pieniężne, a proces etykietowania – czas. W opisywanym w artykule eksperymentie, badacze postanowili zautomatyzować poruszanie się kamery po otwartym świecie wirtualnym bez konieczności ingerencji człowieka. Obiekty uzyskane na obrazie były na bieżąco segmentowane, z dokładnością do pojedynczego piksela, natomiast etykietowanie następowało z wykorzystaniem metadanych pozyskanych z pliku przekazywanego przez grę do karty graficznej, efekt został ukazany na Rysunku 3. Największą trudność sprawił naukowcom fakt, iż gra nie jest programem typu open-source, co uniemożliwia łatwe oskryptowanie programu pod swoje potrzeby, a dostęp do danych nie jest bezpośredni lecz wymaga wykorzystania dodatkowych interfejsów.



Rysunek 3 Efekt segmentacji semantycznej [6]

W podsumowaniu przedstawiono potencjał płynący z zaprezentowanej metody data-mining – zebrane dane mogą być przeniesione do zastosowań w świecie rzeczywistym, jako dane trenujące model w całości, lub jeżeli ich jakość nie jest wystarczająca, do trenowania techniką transfer-learning. Metoda okazała się bardzo wydajna, dostarczając około 40 tys. zaetykietowanych obrazów w ciągu doby, przy czym podkreślono, że wynik ten uzyskano na sprzęcie średniej klasy i możliwa jest jego poprawa przy zastosowaniu większej mocy obliczeniowej [6].

W artykule pt. „*SqueezeSeg: Convolutional Neural Nets with Recurrent CRF for Real-Time Road-Object Segmentation from 3D LiDAR Point Cloud*”, również zaprezentowano metodę pozyskiwania danych ze środowiska wirtualnego, jakim jest gra GTAV, jednak różni się ona co do zasady działania od przykładu opisanego powyżej.



Rysunek 4 Zastosowanie technologii LiDAR w grze GTAV [7]

Podobnie jak poprzednio, tak i tu wykorzystano plug-in o nazwie *Script Hook V*, tym razem w celu umieszczenia na dachu samochodu, którym można swobodnie poruszać się po otwartym świecie gry, wirtualnego urządzenia LiDAR zbierającego dane w postaci chmur punktów, co zostało zaprezentowane na fotografii powyżej (rys.4). Następnie chmury te zostają przekształcone na obraz 2D. Odbywa się to przez przejście z układu kartezjańskiego punktów na układ polarny, w efekcie czego powstaje obraz o pięciu kanałach: współrzędne x, y, z oraz intensywność punktu i jego zasięg [8]. Tak powstałe obrazy są etykietowane poprzez dodatkowe interface-y API, wykorzystujące metadane (podobnie jak w przykładzie powyżej). Opisana technika ma na celu rozszerzenie bazy danych uczących model do rozpoznawania samochodów, pieszych i rowerzystów poruszających się po rzeczywistych ulicach. Dane pochodzące z gry stanowią uzupełnienie bazy danych chmur punktów o nazwie KITTI. Okazuje się, że połączenie danych z bazy KITTI, zawierającej skany przestrzeni rzeczywistej, ze skanami LiDAR z gry GTAV, daje lepsze efekty uczenia modelu niż trenowanie go wyłącznie na danych z bazy KITTI o ponad 15% [7].

Zaprezentowane idee łączy wspólny cel – rozszerzanie źródeł danych uczących o nowe obszary w technologii. Są to kroki w ciekawym kierunku do rozwoju. Przedstawione prace pokazują potencjał i realne zapotrzebowanie, by pozyskiwanie danych trenujących było łatwiejsze. Są niewątpliwą inspiracją do dalszego poszukiwania rozwiązań oraz potwierdzeniem słuszności prowadzenia eksperymentu w niniejszym projekcie.

4. UCZENIE MASZYNOWE I JEGO RODZAJE

Rozdział ten poświęcony został zaprezentowaniu istniejących technologii, które rozwijając się przez lata z różną intensywnością, dostarczają obecnie wszechstronne narzędzia do klasyfikacji czy predykcji. Na wstępie omówione zostały techniki uczenia maszynowego (*Machine Learning*), obowiązujące w nim podziały ze względu na najpraktyczniejsze kryteria. Opisane zostały również częste problemy jakie można napotkać pracując z tą technologią, szczególnie w odniesieniu do dostarczonych danych trenujących. Ponieważ celem eksperymentu opisanego w tej pracy jest rozpoznawanie obrazów przez sieci neuronowe, dlatego szczegółowo opisano w tym rozdziale również technikę klasyfikacji przy użyciu ML, jako alternatywne podejście do zagadnienia. Dalszą część poświęcono sieciom neuronowym oraz opisano miary pomocne przy ewaluacji pracy wytrenowanych modeli.

„[Uczenie maszynowe to] dziedzina nauki dająca komputerom możliwość uczenia się bez konieczności ich jawnego programowania.”

~ Arthur Samuel, 1959

4.1. KRYTERIA PODZIAŁU ML

Podchodząc do nowego projektu związanego z uczeniem maszynowym należy sprecyzować na wstępnie rodzaj problemu do rozwiązywania i dobrać odpowiednią metodę jego rozwiązania. Istnieje wiele kategorii algorytmów i strategii, niektóre się pokrywają i współdzielą te same algorytmy. Kiedy dokładnie określi się wszystkie cechy zagadnienia można przystąpić do doboru optymalnej lub niekiedy jedynej słusznej, metody postępowania. Do głównych kryteriów podziału systemu uczenia maszynowego można zaliczyć:

- 1) uczenie nadzorowane lub nienadzorowane,
- 2) uczenie wsadowe lub uczenie przyrostowe,
- 3) uczenie z przykładów lub uczenie z modelu.

Pierwszy z wymienionych podziałów wynika z rodzaju danych jakie są dostarczone do rozwiązywania problemu – jeżeli dane są opatrzone etykietami, mówi się o uczeniu nadzorowanym, w przeciwnym wypadku o uczeniu nienadzorowanym. Jeżeli jest wiadomym, że projekt wymaga przeprowadzania precyzyjnej klasyfikacji a dane nie posiadają etykiet,

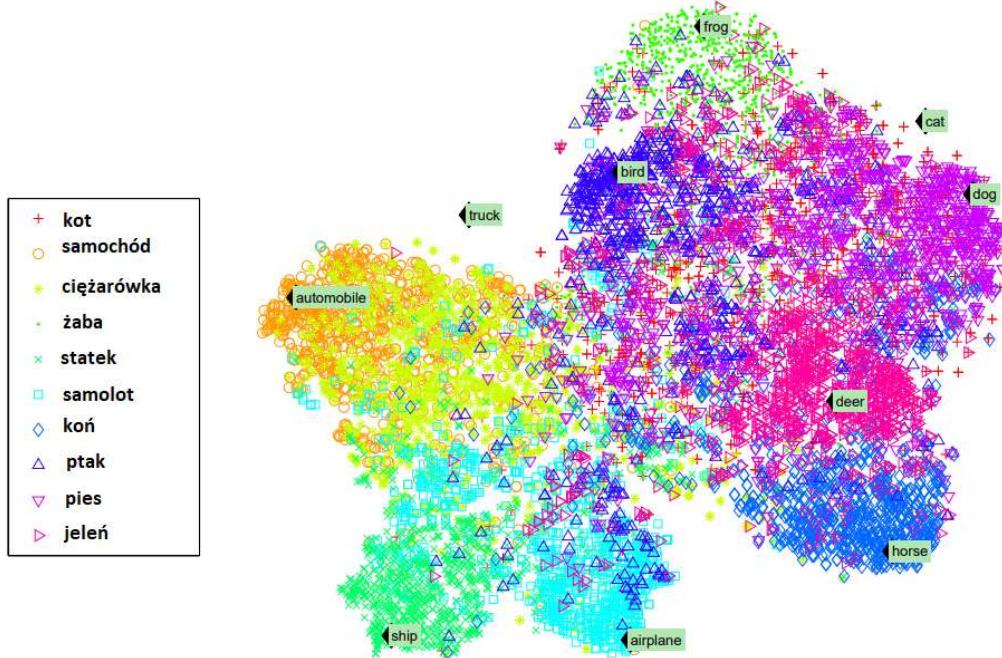
należy takie dołączyć do przykładów w ramach procesu opracowywania danych. Uczenie nadzorowane umożliwia rozwiązywanie problemów związanych z rozpoznawaniem obiektów (klasyfikacja) i przewidywania wartości w oparciu o algorytmy regresji. Regresja jest narzędziem powszechnie wykorzystywana ze względu na prostotę implementacji i logiki działania, niestety bywa również niewystarczająco dokładna, gdy związki pomiędzy atrybutami danych są zbyt skomplikowane [9]. Do algorytmów uczenia maszynowego nadzorowanego najczęściej używanych zalicza się:

- sieci neuronowe,
- metoda k-najbliższych sąsiadów,
- regresja liniowa,
- regresja logistyczna,
- maszyny wektorów nośnych,
- drzewa decyzyjne i losowe lasy.

Uczenie nienadzorowane polega na wykorzystywaniu danych, które nie są opatrzone etykietami, z tego powodu charakter ich pracy polega na uogólnianiu cech obiektów i grupowaniu ich aniżeli na precyzyjnej klasyfikacji (Rys. 5). System pracy bez nauczyciela, w postaci człowieka dodającego etykiety, oszczędza wiele czasu [10].

Ten system uczenia jest stosowany najczęściej do następujących typów zadań:

- analiza skupień,
- wykrywanie anomalii,
- redukcja wymiarowości,
- uczenie z wykorzystaniem reguł asocjacyjnych.



Rysunek 5 Przykład zastosowania uczenia nienadzorowanego do analizy skupień. Widać na wykresie podobieństwo obiektów z klas „kot” i „pies”, oraz znaczącą różnicę pomiędzy klasami „żaba” i „statek” [11]

Drugi podział wynika ze sposobu dostarczania danych do nauki oraz ich ilości. O uczeniu wsadowym mówi się, gdy do nauki modelu stosuje się jeden obszerny pakiet danych, natomiast uczenie przyrostowe opiera się o „douczanie” modelu na podstawie stale napływających informacji w postaci małych pakietów. W sytuacji, gdy model wymaga odświeżania częściej niż raz na dobę, mniej opłacalne jest rozwiązywanie uczenia wsadowego, nazywanego uczeniem offline, ponieważ sama nauka trwa kilka godzin zużywając dużo energii, a nowy model całkowicie zastępuje poprzedni. W takim wypadku najlepiej sprawdza się uczenie przyrostowe. Czas trwania uczenia modelu na podstawie małego pakietu nowych informacji, które regularnie napływają do bazy danych, jest o wiele krótszy i istniejący model jest jedynie rozszerzany. Dodatkową zaletą uczenia przyrostowego jest fakt, że jego algorytmy sprawdzają się w sytuacji, gdy celem jest nauczenie modelu na bazie danych przewyższających objętością dostępną pamięć sprzętową. Można wówczas zastosować ten typ uczenia by model powstał w kilku etapach i po każdym z nich wykorzystane dane mogły być wymieniane na nowe.

Ostatni podział opiera się o metodę uogólniania – sposób przewidywania nowych obiektów na podstawie wyuczonego systemu. Tak dla rozróżnienia istnieje uczenie na przykładach i uczenie z modelu. Pierwsze odnosi się do sytuacji, w której system z podanych danych wyciąga ich cechy, a następnie porównuje je z cechami nowo napotkanego przykładu i przy pomocy ustalonej miary podobieństwa tych cech kategoryzuje nowy obiekt. W kontrze do tej metody funkcjonuje wymienione uczenie z modelu – na podstawie dostarczonych danych odszukiwane są zależności między cechami i trend ich wartości co prowadzi do powstania generalizującej funkcji. W najprostszym przykładzie może to być funkcja liniowa jako wynik regresji liniowej punktów w układzie współrzędnych, w którym osiom przypisane są wartości dwóch cech.

Niestety najczęściej zależności między cechami atrybutów danych są bardziej skomplikowane i prosta regresja liniowa nie daje skutecznych rezultatów a modele muszą zostać rozbudowane (jak się okaże w rozdziale *Problemy uczenia maszynowego*, zbyt skomplikowany model również nie daje oczekiwanych rezultatów) [1] [12].

4.2. PROBLEMY UCZENIA MASZYNOWEGO

Proces tworzenia poprawnie funkcjonującej struktury logicznej jaką jest system uczenia maszynowego, jest skomplikowany i powiązany wieloma wewnętrznymi zależnościami. Niektóre z nich prowadzą do niepowodzeń lub niedostatecznie dobrych wyników podczas testowania i nie jest to efekt niespodziewany. Ponieważ każdy projekt jest inny i wymaga indywidualnego podejścia, nie wszystkie uogólnione zasady są na tyle uniwersalne, żeby nie wymagały dodatkowych regulacji. Normalną praktyką jest testowanie nowoutworzonych modeli, różniących się parametrami nauki, architekturą sieci, czy funkcją aktywacji i wybranie najlepszego.

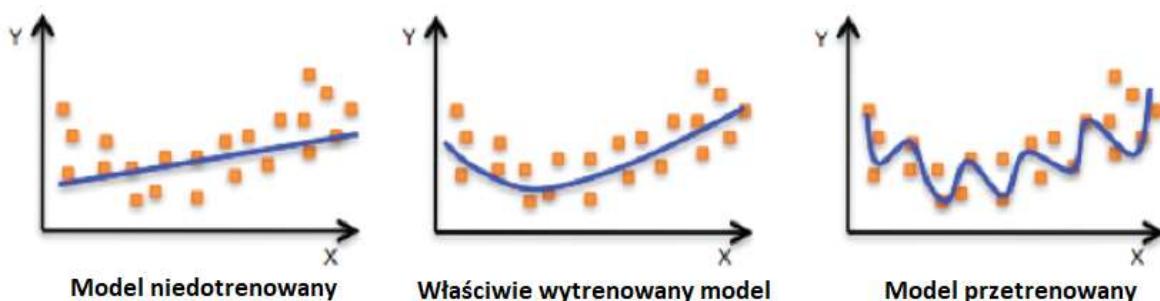
Istnieją dwa główne źródła problemów, z którymi należy się liczyć podejmując działania w dziedzinie uczenia maszynowego. Pierwszym z nich są dane uczące, drugim algorytm. Poniżej wymieniono niepożądane dane cechy danych wraz z opisem:

- Niedobór danych uczących - najczęstszym powodem niepowodzeń i małej precyzji pracy systemów jest zastosowanie zbyt nielicznych zbiorów uczących. Powoduje to nieprawidłowe uformowanie modelu i prowadzi do częstych pomyłek podczas jego pracy.
- Niereprezentatywność danych uczących - nie wyczerpują one różnorodności występowania danego przypadku obiektu czy zjawiska w rzeczywistości i tylko ich okrojona forma jest „prezentowana” modelowi podczas nauki. W rezultacie, w odniesieniu do rozpoznawania obiektów na obrazach, już niewielka zmiana oświetlenia ekspozycji powoduje kłopot z zaklasyfikowaniem jej przez model.
- Zła jakość danych – najczęściej pochodzą one z niedokładnych pomiarów. W rezultacie zbiór, którym dysponujemy może zawierać dane o wartościach zdecydowanie odbiegających od pozostałych. Taka niedokładność powoduje zakłamanie wprowadzające uczony model w błąd. Aby zapowiedz takie sytuacji możliwe jest usunięcie znacznie różniących się przykładów bądź samodzielne ich edytowanie i poprawa wedle własnej oceny, by zbliżyć je do pozostałych np. przez uśrednienie. Kolejnym przykładem zlej jakości danych są te nie zawierające opisu wszystkich wymaganych cech, np. część danych pochodzących z ankiety nie zawiera odpowiedzi na kluczowane pytanie pomijane przez ankietowanych ze względu na jego wrażliwość. Takie informacje wykluczają możliwość rzetelnego wykorzystania danych. Znajomość podstaw metodologii badań i rzetelne pomiary mają największy wpływ na jakość pozyskiwanych zbiorów uczących.

- Nieistotne cechy – niektóre z atrybutów niewiele mówią o obiekcie z punktu widzenia przygotowania modelu do rozpoznawania go, natomiast wnioskowanie na podstawie kilku cech daje możliwość wyłuskania dodatkowych, cennych informacji. Tym zagadnieniem zajmuje się inżynieria cech – szuka ukrytych informacji wynikających z zależności pomiędzy tymi jawnie dostarczonymi [10].

Do wymienionych przyczyn związanych z danymi należy dodać te wynikające z błędów algorytmów:

- Przetrenowanie – jest to zjawisko, gdy wyuczony model doskonale przewiduje wartości próbek, na których się uczył jednak słabo radzi sobie z nowymi przykładami. Powodem tego zjawiska jest zbyt skomplikowany model wytrenowany na zbyt małej liczbie danych. Najprostszą metodą zapobiegania temu jest uproszczenie modelu np. z wielomianowego do liniowego, ponadto można zmniejszyć ilość atrybutów do nauki. Jeżeli zależy nam, aby model pozostał w niezmienionej formie, by umiał odszukać subtelne różnice między atrybutami, to należy rozszerzyć bazę przykładów do nauki.
- Niedotrenowanie danych uczących – zgodnie z intuicją jest to sytuacja odwrotna do przetrenowania i polega na zastosowaniu zbyt prostego algorytmu, który nie jest w stanie odwzorować skomplikowanego rozrzutu danych uczących. Przeciwdziała się wówczas stosując model o wyższym stopniu złożoności, większej liczbie stopni swobody, czyli parametrów podlegających strojeniu w czasie procesu nauki modelu, dla przykładu: model liniowy posiada dwa stopnie swobody – wysokość prostej i nachylenie (Rys. 6) [3].



Rysunek 6 Zobrazowanie różnic pomiędzy modelami wytrenowanymi właściwie oraz nieprawidłowo [40]

4.3. KLASYFIKACJA JAKO NARZĘDZIE ML

Zanim pojawiły się sieci neuronowe, tworząc w swym skomplikowaniu osobny dział uczenia maszynowego zwany uczeniem głębiom (*Deep Learning*), powstawały metody klasyfikacji oparte o prostsze algorytmy. Ze względu na rodzaj zagadnienia stosuje się jeden z trzech typów klasyfikacji w ML: klasyfikację binarną (*Binary Classification*), klasyfikację wieloklasową (*Multi-Class Classification*), klasyfikację wieloetykietową (*Multi-Label Classification*). Pierwszy z wymienionych typów klasyfikacji jest najmniej złożony i stanowi podstawę działania pozostałych. Realizuje się przy jego pomocy proste zadania oceny czy dany obiekt przynależy do konkretnej klasy, czy nie. Przykładowo jest wykorzystywany w podejmowaniu decyzji czy wiadomość e-mail powinna być zaklasyfikowana jako spam. Zastosowanie klasyfikatora binarnego w odniesieniu do zbioru MNIST skutkowałoby jedynie możliwością sprawdzenia czy badaną cyfrą jest np. „3”, czy nie. Algorytmami wykorzystywanymi w technologii klasyfikacji binarnej są m.in. regresja logistyczna, drzewa decyzyjne, czy naiwny klasyfikator Bayes-a.

W przeciwieństwie do klasyfikatora binarnego, który potrafi odróżniać dwie klasy, klasyfikator wieloklasowy radzi sobie z ich większą ilością. Wykorzystuje w tym celu algorytmy takie jak: algorytm losowego lasu (*Random Forest Classifier*) czy stochastycznego spadku wzduż gradientu (*Stochastic Gradient Decent – SGD*). Jak wspomniano działanie omawianego klasyfikatora opiera się na złożonej pracy klasyfikatorów binarnych. Istnieją dwie strategie działania i są one określane jako:

„OvO” (one-versus-one) – polega na tym, że jeżeli zadaniem jest rozróżnianie wielu klas, to uczymy wiele klasyfikatorów binarnych, po jednym na każdą parę klas.

„OvR” (one-versus-rest) – w tym przypadku dysponując wieloma klasami uczymy po jednym klasyfikatorze na klasę, który decyduje czy badany obiekt należy do danej klasy czy nie. Następnie obiekt jest przypisywany klasie, której klasyfikator osiągnął najlepszy wynik dopasowania.

Użycie poszczególnej metod jest podyktyowane ilością danych do nauki i ilością klas. Należy kierować się zasadą, że szybciej przebiega nauka wielu modeli opartych na nielicznych danych niż kilku modeli uczących się na ogromnej ilości informacji. W przypadku popularnej bazy MNIST, która zawiera ręcznie zapisane cyfry od 0 do 9, należałoby utworzyć aż 45 klasyfikatorów binarnych, następnie poddać klasyfikowany znak ocenie każdemu z nich, by wyłonić ostateczną klasę przydziału. Do algorytmów realizujących ten rodzaj klasyfikacji zaliczają się między innymi: K-najbliższych sąsiadów (*k-Nearest Neighbors*), czy drzewa decyzyjne (*Decision Trees*).

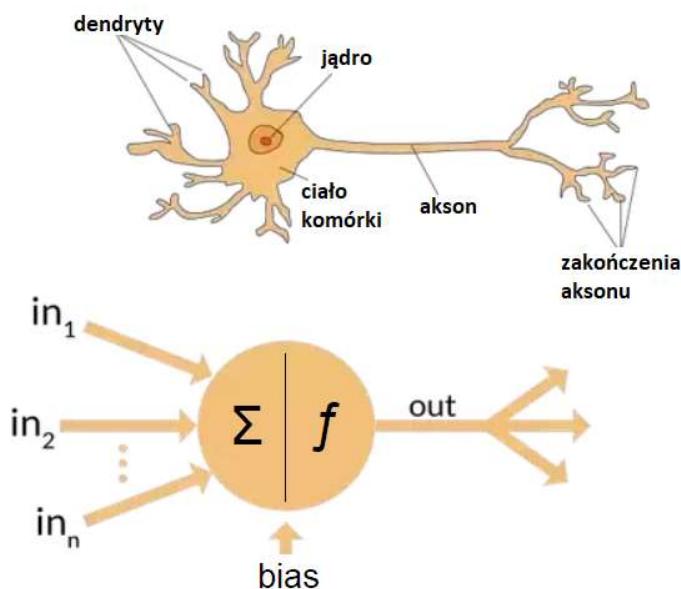
Ostatnim typem klasyfikacji w ML jest uczenie wieloetykietowe. Nie wszystkie algorytmy sobie z nim radzą tak dobrze jak metoda K-najbliższych sąsiadów (*k-Nearest Neighbors*). Stosuje się go w momencie, gdy dysponując wieloma klasami chcemy jednocześnie przypisać wiele etykiet. Może to dotyczyć jednego obiektu wspólnie dzielonego przez różne klasy, o różnych

kryteriach lub gdy wymagane jest określenie przynależności wielu obiektów do różnych klas jednocześnie, np. rozpoznanie na jednym zdjęciu człowieka i samochodu [13].

Klasyfikacja wg. jednego z przedstawionych typów mogłaby posłużyć do porównania jakości predykcji obiektów zgodnie z tematem pracy jednak zdecydowano się na algorytmy bardziej złożone, będące w stanie rozróżniać obiekty w oparciu o subtelne detale, pod warunkiem ich prawidłowego wytrenowania, mowa o wspomnianych sieciach neuronowych, które zostaną szczerzej omówione w dalszej części.

5. SIECI NEURONOWE

Zgodnie z tym co zostało przedstawione we wstępie, sieci neuronowe konceptualnie zostały zainspirowane pracą układu nerwowego człowieka, na co wskazuje nazwa tych algorytmów. Neurony biologiczne transportują informacje w obrębie naszego ciała, w postaci impulsów elektrycznych. Komórki te są połączone ze sobą wypustkami, odbierające informacje nazywane są dendrytami, a wysyłające sygnał do kolejnej komórki – aksonami. Każda komórka nerwowa posiada dendryty zbierające informacje i tylko jeden akson - wypustkę, którą impuls opuszcza komórkę, o budowie przystosowanej do pełnionej roli (każdy akson jest owinięty osłonkami mielinowymi, między którymi istnieją przerwy, tzw. przewężenia Ranviera – całość ma za zadanie szybszy i bezstratny transport ładunku od komórki do komórki) (Rys. 7).



Rysunek 7 Analogia pomiędzy neuronem biologicznym a tym sztucznej inteligencji [14]

Najprostszą sztuczną sieć neuronową (SSN) zaproponował Frank Rosenblatt w 1957 r. Została ona określona mianem *perceptronu* i polegała na zmodyfikowaniu sztucznego neuronu do

formy przyjmującej wartości liczbowe, a nie binarne, natomiast połączenia pomiędzy neuronami zostały obarczone wagami. Taka forma pojedynczego neuronu została nazwana *progową jednostką logiczną* (ang. *Threshold Logic Unit – TLU*) (rys 5). Zasada działania TLU jest następująca: oblicza ona średnią ważoną wartości wejściowych, by następnie na jej podstawie wyliczyć funkcję skokową i otrzymać wynik końcowy. Jako funkcję skokową często stosuje się funkcję skokową Heaviside-a. Perceptronem nazywamy więc sieć złożoną z jednej warstwy jednostek TLU, w której każdy neuron jest połączony ze wszystkimi wejściami. Wspomnianą warstwę może nawet stanowić tylko jedna sztuczna komórka TLU [10].

$$Heaviside_{(z)} = \begin{cases} 0 & \text{jeśli } z < 0 \\ 1 & \text{jeśli } z \geq 0 \end{cases} \quad (1)$$

Uczenie perceptronu jest oparte o regułę Hebbia, która mówi, że biologiczne połączenie neuronalne pomiędzy neuronami, które się często pobudzają, umacniają się. [15] Modyfikacja tej reguły stanowi podstawę idei uczenia perceptronu, a mianowicie – porównywany zostaje wynik przewidywany przez perceptron z wartością oczekiwana, następnie „wzmacnia się” połączenia, które minimalizują błąd powstały, wskutek tego porównania. Wzmocnienie polega na zmianie wag wg. wzoru:

$$w_{i,j}^{(następny krok)} = w_{i,j} + \eta(y_j - \hat{y}_j)x_i \quad (2)$$

W tym równaniu:

- 1 $w_{i,j}$ – waga połączenia pomiędzy i-tym neuronem wejściowym i j-tym neuronem wyjściowym,
- 2 x_i – i-ta wartość wejściowa danego przykładu uczącego,
- 3 y_j – docelowa wartość wyjściowa j-tego neuronu,
- 4 \hat{y}_j – aktualna wartość wyjściowa j-tego neuronu,
- 5 η – współczynnik uczenia (jego wartość jest sterowanym hiperparametrem)

Perceptron posiada wady wynikające z prostej budowy tej sieci. Nie pozwala na przeprowadzanie bardziej złożonych operacji jak na przykład rozstrzygnięcie alternatywy rozłącznej (XOR) [16]. Z tego powodu rozpoczęto rozbudowywanie sieci, dokładając dodatkowe warstwy jednostek TLU, w ten sposób powstał perceptron wielowarstwowy. Składa się on z warstwy wejściowej, warstwy ukrytej i warstwy wyjściowej. Warstwę ukrytą stanowi co najmniej jedna warstwa jednostek TLU, natomiast warstwę wyjściową zawsze tworzy jedna warstwa tych sztucznych neuronów. Do warstwy wejściowej i warstw ukrytych dodawane są neurony przeciżenia. Ilość warstw ukrytych określa, czy mówimy o płytcej czy głębokiej sieci neuronowej (ang. *Deep Neural Network – DNN*). Jest to jednak umowny podział zmieniający się w czasie wraz z rozwojem możliwości obliczeniowych i tak w latach 90. XX

wieku sieci określane mianem głębokich zawierały ponad dwie warstwy ukryte, obecnie nazywa ta odnosi się do sieci posiadających dziesiątki warstw w sferze ukrytej.

Uczenie perceptronów wielowarstwowych stanowiło wyzwanie dla naukowców aż do momentu zaimplementowania algorytmu propagacji wstecznej. Algorytm zakłada, że podczas dwóch przebiegów przez sieć – po jednym w przód i w tył, z wykorzystaniem algorytmu gradientu prostego i automatycznej metody obliczania gradientu, uda się obliczyć gradienty błędu dla całej sieci i zgodnie z nimi aktualizować wagę. Powtarzanie tej operacji prowadzi do uzyskania wag, przy których sieć uzyskuje najmniejszy błąd predykcji. Sposób uczenia sieci wielowarstwowych z wykorzystaniem propagacji wstecznej, to nie jedyna różnica w odniesieniu do funkcjonowania pierwotnego perceptronu jednowarstwowego, ponieważ algorytm wykorzystuje metodę gradientu prostego należało zmienić funkcję aktywacji gdyż wspomniana metoda nie radzi sobie z płaskimi fragmentami przebiegów ze względu na swoją naturę (metoda gradientu prostego ma na przykład problem z odnalezieniem minimum globalnego, ponieważ minima lokalne wstrzymują działanie algorytmu fałszując wskazanie), dlatego do uczenia z propagacją wsteczną zaczęto wykorzystywać funkcję sigmoidalną, jako funkcję aktywacji. Kolejnymi, z sukcesem wykorzystywany funkcjami są: funkcja tangensa hiperbolicznego, czy funkcja ReLU (ang. *Rectified Linear Unit*).

5.1. PROCES UCZENIA GŁĘBOKIEJ SIECI NEURONOWEJ

W rozdziale tym opisano strategie postępowania podczas nauki głębokich sieci neuronowych tak, by osiągać najlepsze efekty. Wiele z rozwiązań ewoluowało przez lata rozszerzając funkcjonalność tych złożonych algorytmów.

Algorytm propagacji wstecznej stanowi sedno uczenia sieci neuronowej, dlatego w ramach podsumowania przedstawiono cały proces w krokach:

- 1) Wyznaczenie wag losowych – umożliwia to przełamanie symetrii już na początku, determinując zróżnicowany rozwój poszczególnych gałęzi sieci, właściwie go ukierunkowując. W przypadku ustawienia początkowych wag jednakową wartością np. „0”, wagie zmienione przez algorytm dalej by miały jednakowe wartości.
- 2) Dane w postaci podgrup są przekazywane do warstwy wejściowej, a z niej przechodzą do pierwszej warstwy ukrytej. Sygnał wychodzący z każdego neuronu stanowi sumę iloczynów sygnałów wchodzących i odpowiadających im wag, z ewentualnym uwzględnieniem wartości biasu, która najczęściej jest stała i wynosi 1, natomiast w trakcie uczenia sieci zmianie ulega jedynie waga biasu. Wartość sumy jest następnie poddawana funkcji aktywacji. Wyniki obliczeń wszystkich neuronów danej warstwy przekazywane są do następnej warstwy ukrytej i tak dalej, aż do osiągnięcia warstwy wyjściowej.
- 3) Mierzony jest błąd na wyjściu w odniesieniu do wartości oczekiwanej.

- 4) Przy pomocy reguły łańcuchowej (ang. *chain rule*) obliczany jest wpływ każdego z połączów wyjściowych na błąd predykcji [9].
- 5) Cofając się warstwa po warstwie, z nieustannym wykorzystaniem reguły łańcuchowej (jedna z najważniejszych reguł w matematyce analitycznej, wykorzystywana przy różniczkowaniu funkcji złożonych), liczone są gradienty błędów przy przejściu przez każde z połączeń, przesuwając się w kierunku pierwszej warstwy (stąd nazwa algorytmu wstecznej propagacji).
- 6) Znając gradienty błędów wszystkich połączeń, przy pomocy metody gradientu prostego aktualizowane są ich wagi [2] [3] [17].

5.1.1. Znikające i eksplodujące gradienty

Jest to zjawisko dotyczące etapu wstecznej propagacji, podczas której zgodnie z tym co zostało wcześniej powiedziane, odbywa się obliczanie gradientów błędu. Mówimy o zanikających gradientach w sytuacji, gdy cofając się warstwa po warstwie począwszy od wyjściowej, gradient kolejnych połączeń staje się tak niewielki, że ostatnie połączenie – między warstwą wejściową a pierwszą warstwą ukrytą, praktycznie nie ulega zmianie, co powoduje brak rozwoju całej sieci. Odwrotną sytuację tj., gdy gradienty gwałtownie rosną w miarę postępowania wstecznej propagacji, nazywamy zjawiskiem gradientów eksplodujących. Te dwa zajścia zniechęciły wielu naukowców na początku XXI w. opóźniając rozwój głębokich sieci neuronowych. Dopiero w 2010 roku Xavier Glorot i Yoshua Bengio w swoim artykule zwróciли uwagę na potencjalne powody tych zjawisk. Okazało się, że możliwe jest ograniczenie ich poprzez odpowiednią inicjalizację wstępnych wag oraz zmianę funkcji aktywacji, co zostanie przybliżone w tym rozdziale. Wspomniani autorzy dostrzegli wymaganą do prawidłowego działania, zależność mówiącą o tym, że wariancja sygnałów wejściowych i wyjściowych w każdej warstwie musi być sobie równa, żeby nie doprowadzić do wzmacniania osiągającego nasycenie funkcji ani do jej wygaśnięcia. Zaproponowano wówczas inicjalizację wartości wag początkowych, która została nazwana od imienia autora Inicjalizacją Xaviera. Zanim zostanie ona przybliżona należy wspomnieć o obciążalności wejściowej (ang. *fan-in*) i wyjściowej (ang. *fan-out*) – są to wartości przypisane do każdej warstwy i opisujące, ile w tej warstwie jeden neuron posiada połączeń wejściowych (obciążalność wejściowa), a ile wyjściowych (obciążalność wyjściowa). Inicjalizacja Xaviera polega na przypisaniu wartości losowych ze zbioru o rozkładzie normalnym, średniej równej 0 i wariancji wg. poniższego wzoru:

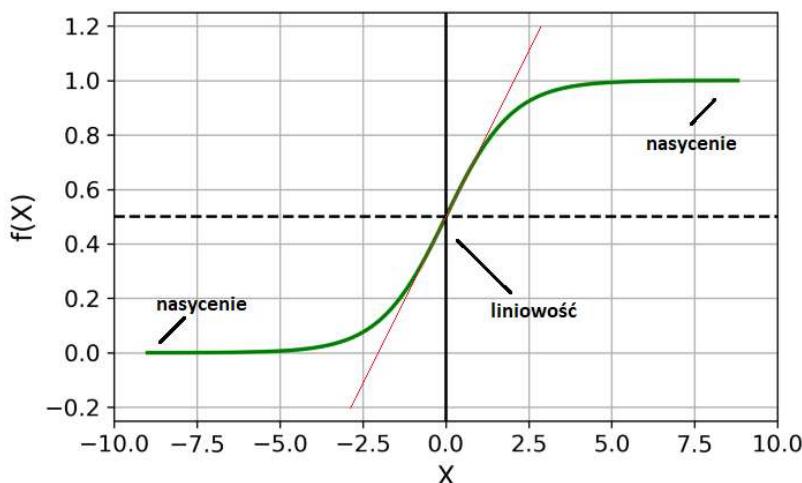
$$\sigma^2 = \frac{2}{obc_{wej} + obc_{wyj}} \quad (3)$$

Lub rozkładu jednorodnego pomiędzy -r i r, gdzie $r = \sqrt{\frac{3}{obc_{\text{sr}}}}$.

Do dzisiaj powstały różne propozycje inicjalizacji - metoda LeCun (ang. *LeCun initialization*) zastępuje średnią z obc_{wej} i obc_{wyj} jedynie wartością obciążalności wejściowej. Obecnie najczęściej stosowanej funkcji aktywacji, jaką jest funkcja ReLU (omówiona w dalszej części), towarzyszy inicjalizacja He (ang. *He initialization*). Główna różnica pomiędzy typami inicjalizacji tkwi w sposobie liczenia wariancji losowanych wartości wag, dla inicjalizacji He można przyjąć rozkład normalny o wariancji $\sigma^2 = \frac{3}{obc_{wej}}$,

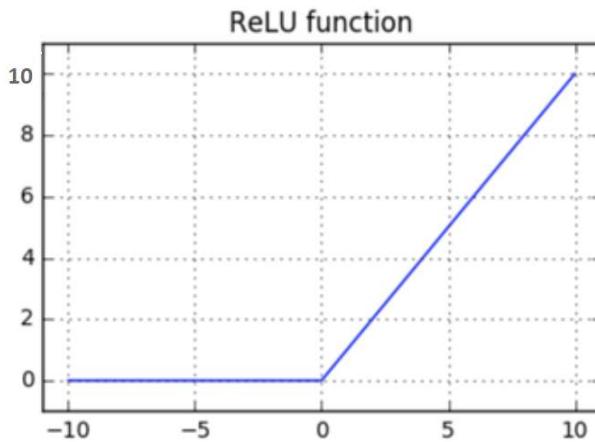
lub rozkład jednostajny z przedziału $(-r, r)$, dla $r = \sqrt{3\sigma^2}$.

Drugim ważnym powodem pojawiania się zjawiska zanikających gradientów jest niewłaściwie dobrana funkcja aktywacji. Do niedawna uważano, że najlepszą jest logistyczna funkcja sigmoidalna (Rys. 8). Pogląd ten wywodził się z obserwacji działania mózgu ssaka, na bazie której założono, że skoro funkcja o podobnym przebiegu pobudza neurony biologiczne, to stanowi najlepsze rozwiązanie w przypadku neuronów sztucznych. Było to mylne założenie a zdanie sobie z tego sprawy doprowadziło do drastycznej zmiany podejścia w rozwoju głębokich sieci neuronowych, które do tej pory kładło nacisk na szukanie analogii z sieciami biologicznymi.



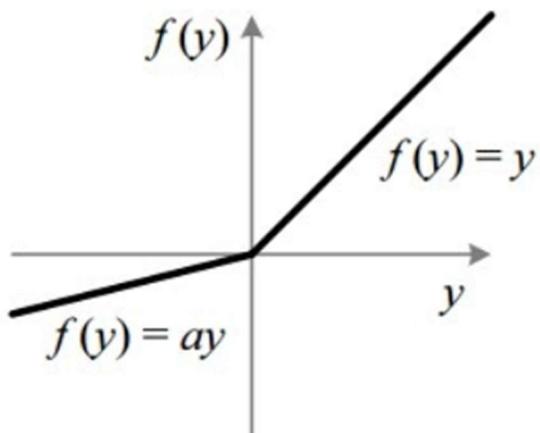
Rysunek 8 Przebieg funkcji sigmoidalnej [41]

Obecnie najpopularniejszą jest funkcja ReLU (Rys. 9). Jest stosowana powszechnie i uchodzi za uniwersalną, choć i ona ma swoją wadę – problem umierania ReLU (ang. *The Dying ReLU Problem*). Przypomnijmy, że wartość podawana do funkcji aktywacji jest to suma iloczynów sygnałów wychodzących z poprzedniej warstwy i odpowiadających im wag. Jeżeli do neuronu dochodzi również bias to jest on mnożony przez swoją wagę i dodawany do wymienionej sumy. Na wykresie przebiegu funkcji ReLU widać, że przekształca ona ujemny argument na wartość wyjściową równą 0.



Rysunek 9 Przebieg funkcji ReLU [18]

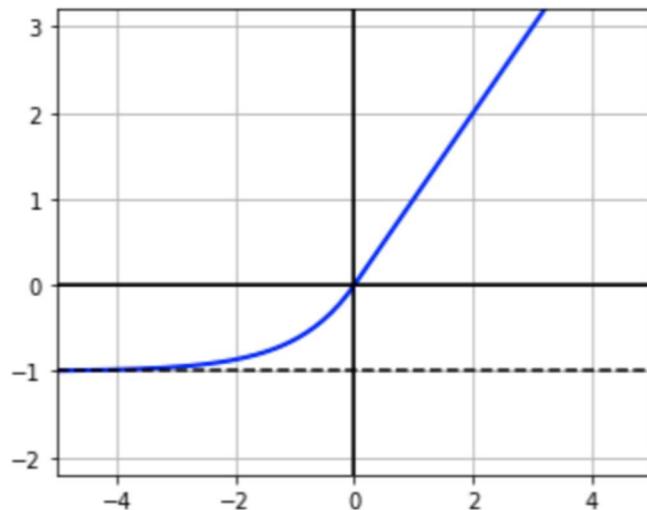
Powoduje to, że neurony „zamierają” (zjawisko nazywa się *problemem umierającego neuronu ReLU*) i przestają brać udział w uczeniu sieci. Spojrzawszy na wzór opisujący zasadę aktualizacji wag, widać, że powodem zamierania neuronów przy udziale funkcji ReLU, jest między innymi zbyt wysoka wartość współczynnika uczenia. Aby zapobiec temu zjawisku należy więc utrzymywać wartość współczynnika uczenia na niewysokim poziomie, zadbać o przełamanie symetrii i odpowiednie dobranie wag podczas inicjalizacji. Można też zastosować inną wersję omawianej funkcji – przeciekające ReLU (ang. *Leaky ReLU*) (Rys. 10) [2].



Rysunek 10 Przebieg funkcji Leaky ReLU [45]

Dzięki pomnożeniu argumentu ujemnego przez niewielką wartość (najczęściej 0.1), na wyjściu otrzymujemy niewiele odchylone od osi OX wartości ujemne. Brak zera na wyjściu tej wersji funkcji aktywacji, powoduje, że neuron nawet jeżeli zostanie chwilowo wyłączony z procesu nauki, to ma dużą szansę „obudzić się” i brać dalszy udział w treningu. Istnieją również inne funkcje aktywacji, które w połączeniu z zalecanymi im metodami inicjalizacji dają dobre efekty. Jedną z nich jest funkcja tangensa hiperbolicznego. Swoim przebiegiem przypomina

ona omówioną już funkcję sigmoidalną z tą różnicą, że średnia wartość osiągana przez nią wynosi 0, jest to zaletą, przez którą wyparła funkcję sigmoidalną. Niestety podobnie jak jej poprzedniczka, tak i ta funkcja łatwo ulega nasyceniu przez większe argumenty. Z wymienionych powodów prym wiedzie ReLU oraz jej modyfikacje - oprócz wymienionej Leaky ReLU zaproponowano również ELU (Rys. 11) i SELU.



Rysunek 11 Przebieg funkcji ELU [19]

Te współczesne funkcje dają dobre efekty pod warunkiem znacznej bazy danych, co nie zawsze jest możliwe do spełnienia. Ich zaletą jest łagodny spadek wartości dla argumentów ujemnych, jednak ze względu na użycie funkcji wykładniczej czas realizacji jest wydłużony [3].

5.1.2. Optymalizatory

Opisane wyżej techniki pozwalają skuteczniej osiągnąć pożądany cel i zapewniają prawidłowy przebieg procesu uczenia sieci neuronowej. W tej części zwrócono uwagę na metody optymalizacji czasu tego procesu. Osiąganie minimum błędu, przy ogromnej ilości danych sprawiają, że uczenie maszynowe nawet z wykorzystaniem wysokiej klasy sprzętu zajmuje bardzo dużo czasu. W dotychczas omawianym algorytmie uczenia sztucznej sieci neuronowej wspomniano, że wagi połączeń między neuronami poszczególnych warstw są aktualizowane z wykorzystaniem algorytmu gradientu prostego. Istnieją jednak liczne modyfikacje

usprawniające działanie tego prostego algorytmu, które pozwalają na wyraźną poprawę osiągów uczenia modeli. Jedną z nich jest algorytm SGD (ang. *Stochastic Gradient Descent*), który dobrze spisuje się przy obszernych bazach danych, pozwalając na przyspieszone trenowanie modelu przez podział danych wsadowych na podgrupy. Główną wadą algorytmu gradientu prostego jest poruszanie się w kierunku minimum drogą wydłużoną ze względu na swoiste zaszumienie (tzw. „zygzak”), wynikające z nieustannej korekcji kierunku obarczonej przeregulowaniem. Kolejną modyfikacją opublikowaną już w 1964 roku, był algorytm *momentum* (Równanie poniżej) [20]. Idea optymalizatora jest następująca: nie odejmuje od aktualnej wagi pojedynczego gradientu funkcji kosztu, lecz w wersji *momentum* tworzony jest wektor gradientów z poprzednich przebiegów, który jest dodawany do aktualnej wartości wagi. Aby przyspieszenie osiągnęło ustalony poziom i nie wzrastało w nieskończoność dodano hiperparametr o nazwie *moment*, mający za zadanie wyhamowanie spadku do zadanej granicy. Wersja ta znaczco skracza przebieg algorytmu aż do uzyskania celu.

$$1. \quad m \leftarrow \beta m - \eta \nabla_{\theta} J(\theta) \quad (4)$$

$$2. \quad \theta \leftarrow \theta + m \quad (5)$$

Gdzie:

- m – wektor momentu,
- θ – waga,
- $J(\theta)$ – funkcja kosztu od wag,
- β – moment (momentum). Przyjmuje wartości od zera do jeden, gdzie 0 to duże spowolnienie, 1 – brak wyhamowywania.
- η – współczynnik uczenia.

Modyfikacją algorytmu momentu jest pomysł z 1983 roku, który dokonuje sprawdzenia gradientu nie dla wagi aktualnie aktualizowanej, lecz poprzedzającej ją w kierunku momentu. Wersja ta nosi nazwę algorytmu Nesterova (Równanie poniżej) i niemal zawsze przyspiesza uczenie w porównaniu do algorytmu momentum, ponieważ kierunek schodzenia jest dokładniej obrany w stronę minimum i algorytm szybciej osiąga poszukiwaną wartość. Przewaga momentum i algorytmu Nesterova nad gradientem prostym polega na udziale poprzednich wag w ukierunkowywaniu spadku, co wpływa na jego wygładzenie.

$$1. \quad m \leftarrow \beta m - \eta \nabla_{\theta} J(\theta + \beta m) \quad (6)$$

$$2. \quad \theta \leftarrow \theta + m \quad (7)$$

Gdzie:

- m – wektor momentu,
- θ – waga,
- $J(\theta)$ – funkcja kosztu od wag,
- β – moment (momentum). Przyjmuje wartości od zera do jeden, gdzie zero to duże spowolnienie, 1 – brak wyhamowywania.
- η – współczynnik uczenia.

Kolejnym rozwiązańem, szczególnie chętnie wykorzystywanym przy prostych zadaniach kwadratowych, jest algorytm AdaGrad, widoczny poniżej:

$$1. \quad s \leftarrow s + \nabla_{\theta} J(\theta) \otimes \nabla_{\theta} J(\theta) \quad (8)$$

$$2. \quad \theta \leftarrow \theta - \eta \nabla_{\theta} J(\theta) \oslash \sqrt{s + \epsilon} \quad (9)$$

Gdzie:

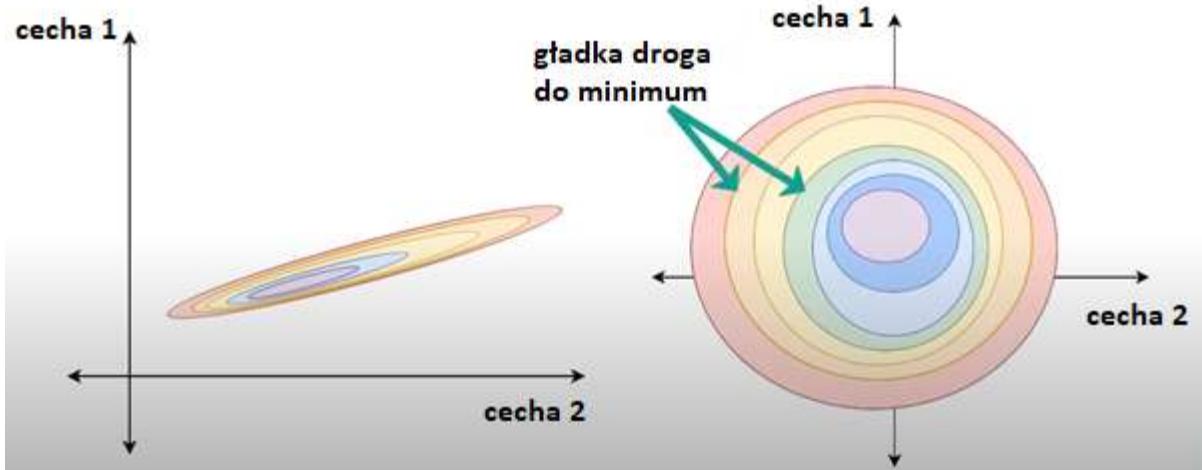
- \otimes - mnożenie przez poszczególne elementy,
- \oslash - dzielenie przez poszczególne elementy,
- θ – waga,
- $J(\theta)$ – funkcja kosztu od wag,
- s – wektor gromadzący kwadraty gradientów,
- η – współczynnik nauki,
- ϵ – współczynnik wygładzający

Algorytm ten jest uzbrojony w mechanizm samoregulacji współczynnika uczenia w zależności od wartości gradientu. Oznacza to, że algorytm przyspiesza, gdy przebieg funkcji jest stromy i zwalnia, gdy ten łagodnieje. Wymaga to dodatkowego hiperparametru, jednak jego strojenie jest łatwiejsze od parametru η (współczynnik uczenia) - wystarczy wartość ϵ ustawić na bardzo małą wartość (rzędu 10^{-10}), by wykluczyć dzielenie przez zero, wartość η zostanie automatycznie dostosowana poprzez stale aktualizowany mianownik $\sqrt{s + \epsilon}$. Jak wspomniano algorytm ten nie nadaje się do rozwiązywania skomplikowanych zadań, jak uczenie głębokich sieci neuronowych, jednak stanowi podstawę działania wspólnie najskuteczniejszych algorytmów. Słabością AdaGrad jest fakt, że samoregulujący się współczynnik uczenia wyhamowuje zbyt szybko, co prowadzi do zjawiska, gdy algorytm nigdy nie dociera do szukanego minimum. Kolejna wersja algorytmu o nazwie RMSProp posiada zabezpieczenie naprawiające ten błąd. Polega ono na gromadzeniu gradientów jedynie aktualnych, a nie wszystkich od początku całego procesu trenowania. Najnowsze optymalizatory opierają się o

fuzję algorytmu momentum oraz RMSProp. Wybranie najlepszych cech tych dwóch metod i połączenie w jeden algorytm skutkowało powstaniem algorytmu Adam, a następnie jego modyfikacji, jak na przykład Nadam. Adam (ang. *Adaptive momentum estimation*) jest algorytmem, który tak jak momentum poprawia ukierunkowanie spadku, natomiast z RMSProp czerpie metodę regulacji szybkości uczenia w zależności od stromości nachylenia przebiegu funkcji celu. Jest optymalizatorem zaawansowanym i współcześnie najczęściej stosowanym. Dzięki adaptacyjnemu współczynnikowi uczenia, nie wymaga skupienia podczas strojenia wartości η , któremu wystarczy nadać niewielką wartość, rzędu 10^{-3} a regulacja dokona się automatycznie. Warto wspomnieć o modyfikacji algorytmu Adam, która wzbogaca go o założenia algorytmu Nesterova, a mianowicie o algorytmie Nadam. Dzięki wybieganiu w przód i obserwacji gradientów poprzedzających aktualizowany, algorytm ten osiąga przeważnie zbieżność szybciej od tradycyjnej wersji [15] [3].

Omówione w tym rozdziale techniki wymagają uzupełnienia o dodatkowe dwie metody, poprawiające czas uczenia. Pierwszą z nich jest *uczenie transferowe* (ang. *transfer learning*), polega na wielokrotnym wykorzystaniu warstw sieci neuronowej do różnych projektów. Istotą tej techniki jest fakt, że sieci uczone do spełniania podobnych zadań wykazują podobieństwo w niższych warstwach. Jeżeli przykładowo chcemy rozszerzyć nasz model o rozpoznawanie kolejnego obiektu osadzonego w podobnym środowisku, to okazuje się, że nowym model od poprzednika będzie różnił się jedynie kilkoma ostatnimi warstwami (lub wręcz tylko ostatnią). Mając to na uwadze możliwe jest zaoszczędzenie ogromnej ilości czasu podczas trenowania nowego modelu, poprzez wykorzystanie tych niższych z poprzedniego modelu lub takiego, który był trenowany do pełnienia podobnej funkcji. Proces polega na dostosowaniu ostatnich warstw do wymaganej ilości wyjść, natomiast wagi warstw niższych są przepisywane z pierwowzoru. Warto zwrócić uwagę, że nowododane, ostatnie warstwy posiadają wagi zainicjalizowane losowo, co może doprowadzić do chwilowego rozstrojenia wag warstw niższych, o których działaniu jesteśmy przekonani na podstawie dobrych wyników osiąganych przez „model-dawcę”. W takiej sytuacji dobrą praktyką jest zablokowanie wag w tych zapożyczonych warstwach i odblokowanie ich po wytrenowaniu kilku czy kilkunastu pierwszych epok [21].

Ostatnią techniką przed przejściem do opisu regularyzacji procesu uczenia, jest normalizacja danych wsadowych (ang. *batch normalization*) (Rys. 12). Jest to rewolucyjna i bardzo opłacalna czasowo strategia postępowania z danymi w czasie procesu uczenia. Polega ona na przeskalowaniu i przesunięciu danych w taki sposób, by wartości cech były względem siebie proporcjonalne i symetryczne.



Rysunek 12 Zobrazowanie istoty normalizacji z wykorzystaniem wykresów cech [22]

Odbiera się to poprzez umieszczenie przed funkcją aktywacją w każdej warstwie, krótkiej serii dodatkowych obliczeń statystycznych. Zalicza się do nich wyliczenie średniej sygnałów wchodzących do warstwy oraz ich odchylenie standardowe dla każdej podgrupy. Na ich podstawie wyliczany jest wektor przesunięć i skalowania wejść, służący do ostatecznego uzyskania danych w pożądanej formie. Aby operacja prawidłowo działała w przypadku, gdy testowana grupa jest zbyt mała by wyliczyć rzetelną statystykę lub gdy wręcz testujemy pojedynczy element, dla którego nie da się obliczyć średniej i odchylenia standardowego, stosuje się dwa dodatkowe parametry: końcowy wektor średnich i końcowy wektor odchyleń standardowych. Te dwa wektory stanowią wykładniczą średnią kroczącą obliczaną w trakcie procesu uczenia, by ostatecznie ich wyniki zastąpiły średnie i odchylenia standardowe sygnałów wejściowych dla każdej warstwy. Ostatecznie uzyskujemy więc cztery nowe parametry: β – wyjściowy wektor przesunięcia, γ – wyjściowy wektor skali, μ – końcowy wektor średnich wejść, σ – końcowy wektor odchyleń standardowych wejść. Sukcesem pojawienia się metody normalizacji jest szereg pozytywnych cech wpływających na proces uczenia sieci. Cały algorytm normalizacji w wsadowej wygląda następująco:

$$1. \quad \mu_B = \frac{1}{m_B} \sum_{i=1}^{m_B} x^{(i)}, \quad (10)$$

$$2. \quad \sigma_B^2 = \frac{1}{m_B} \sum_{i=1}^{m_B} (x^{(i)} - \mu_B)^2, \quad (11)$$

$$3. \quad \hat{x}^{(i)} = \frac{x^{(i)} - \mu_B}{\sqrt{\sigma_B^2 + \varepsilon}} \quad (12)$$

$$4. \quad z^{(i)} = \gamma \otimes \hat{x}^{(i)} + \beta, \quad (13)$$

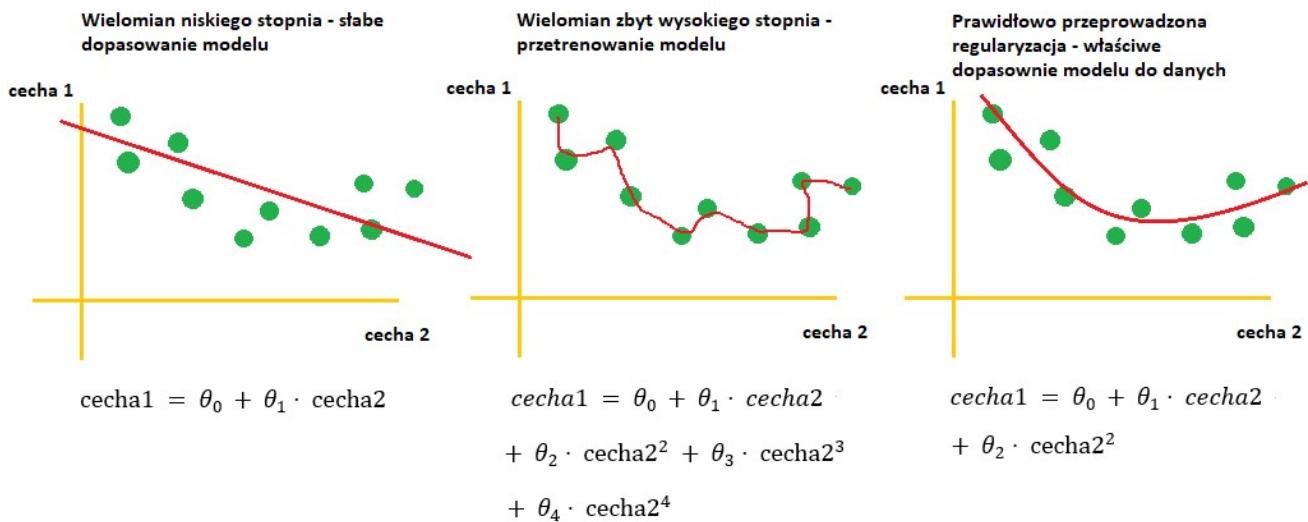
gdzie:

- μ_B – wektor gromadzący uśrednione wejścia dla całej podgrupy,
- σ_B^2 – wektor gromadzący odchylenia standardowe wejść dla całej podgrupy,
- $\hat{x}^{(i)}$ – wektor wyśrodkowania i znormalizowania dla i-tego przykładu,
- $z^{(i)}$ – oczekiwany wynik pracy normalizacji,
- γ – wektor parametrów skalowania, zawiera po jednym parametrze na jedno wejście,
- β – wektor przesunięć, każde wejście jest przesuwane o odpowiadający mu parametr w tym wektorze,
- \otimes - operator mnożenia każdego przykładu przez odpowiadający mu parametr skali,
- ε – współczynnik wygładzający, zapobiega dzieleniu przez zero.

Normalizacja przyspiesza proces uczenia, ponieważ pomimo złożoności obliczeniowej umożliwia zredukowanie ilości uczonych epok, co skutkuje w rezultacie skróceniem czasu nauki modelu. Usprawnia działanie głębokich sieci neuronowych, szczególnie w zadaniach klasyfikacji. Dodatkowo skutecznie zapobiega zjawisku eksplodującym i zanikającym gradientów, do tego stopnia, że stało się możliwe wykorzystywanie funkcji aktywacji łatwo nasycających, jak tangens hiperboliczny czy logistyczna funkcja aktywacji. W pewnym zakresie normalizacja jest również odpowiedzialna za omówioną w dalszej części regularyzację, umożliwiając zrezygnowanie z zabiegu *porzucania* (ang. *dropout*).

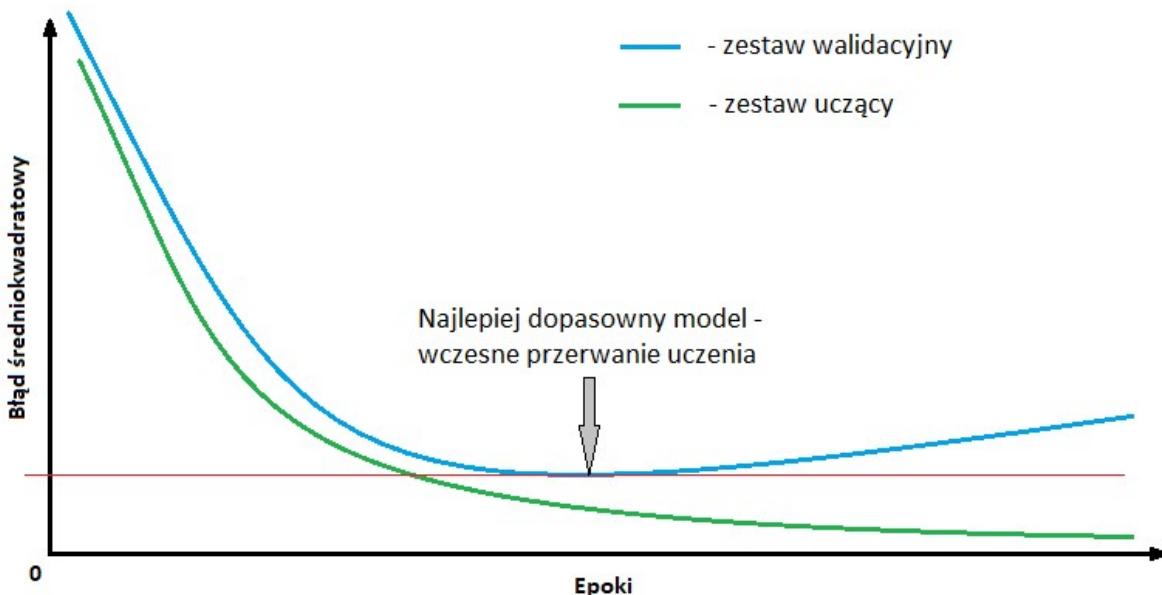
5.1.3. Regularyzacja

Regularyzacja jest zabiegiem mającym na celu powstrzymanie przetrenowania i można ją osiągnąć na wiele sposobów, adekwatnych do rozwiązywanego zadania. Jedną z metod jest regularyzacja l1 oraz l2, która polega na ograniczaniu współczynników funkcji wielomianowej tak, by zmniejszyć stopień tego wielomianu. Uzyskuje się mniej dokładne dopasowanie modelu do danych treningowych co poprawia jego pracę (Rys. 13). Ten rodzaj regularyzacji posiada parametr λ odpowiedzialny za czułość regularyzatora – jeżeli parametr ten ustawimy na wartość 0, to praktycznie wyłączmy regularyzację i otrzymujemy zwykłą regresję liniową, natomiast duża wartość λ skutkuje uzyskaniem wykresu funkcji kosztu w postaci prostej biegącej wzdłuż średniej danych treningowych.



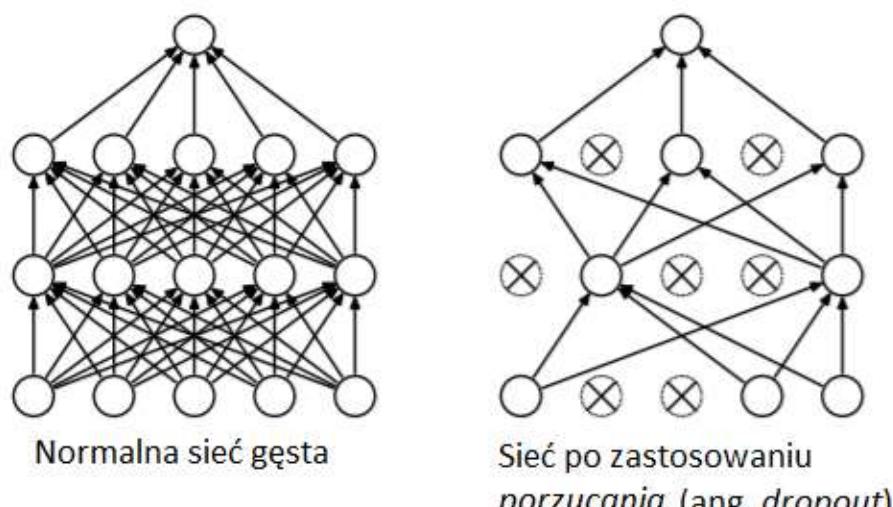
Rysunek 13 Zobrazowanie właściwie dobranego stopnia wielomianu funkcji kosztu [Źródło: opracowanie własnej]

Istnieje inna, łatwiejsza metoda radzenie sobie z przetrenowaniem, a mianowicie wcześnie zatrzymanie trenowania (ang. *early stopping*). Ta prosta strategia opiera się zgodnie z intuicją na przerwaniu procesu nauki w momencie, gdy krzywa nauki osiągnąwszy swą wartość minimalną zaczyna wzrastać. Po zatrzymaniu trenowania ustawia się parametry modelu zgodne z tymi, które uzyskał, gdy krzywa nauki osiągnęła minimum (Rys. 14).



Rysunek 14 Wskazanie właściwego momentu przerwania trenowania przy wykorzystaniu metody "Early stopping" [Źródło: opracowanie własnej]

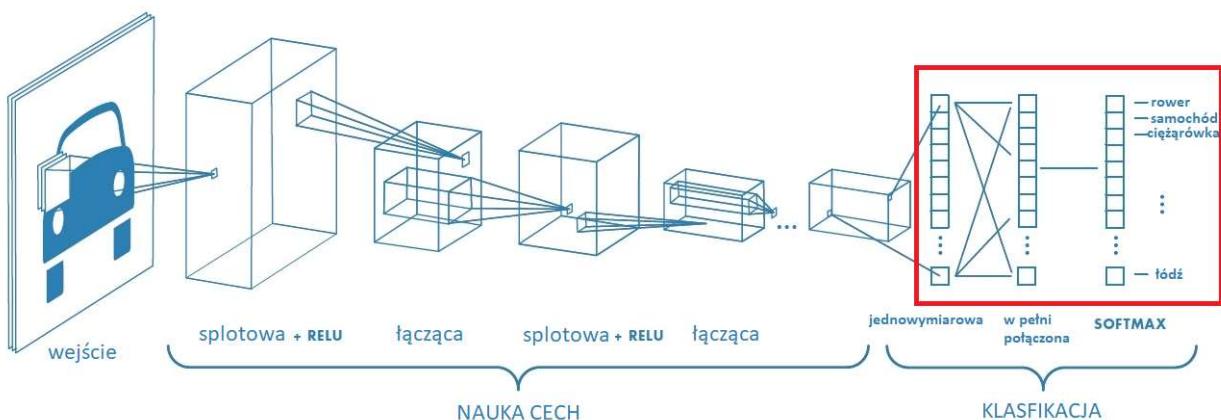
W przypadkach związanych z rozpoznawaniem obrazów i wykorzystywaniem splotowych sieci neuronowych najkorzystniejszą strategią pod względem osiąganych efektów, jest metoda regularyzacji zwana *porzuceniem* (ang. *dropout*). W poprzednim rozdziale wspomniano o niej w kontekście normalizacji, teraz zostanie szczegółowo omówiona. Zaproponowana w 2012 roku metoda bazuje na prostej idei wykluczania z nauki niektórych neuronów w każdym przejściu. Funkcja przyjmuje parametr ρ odpowiadający procentowi prawdopodobieństwa z jakim każdy neuron w warstwie ukrytej może zostać pominięty w danym przejściu treningowym (Rys. 15). W rzeczywistości ta metoda stosowana jest jedynie do kilku ostatnich warstw ukrytych (przeważnie około trzech). Jeżeli wartość parametru ρ zostanie ustawiona na 0.5, to oznacza to, że każdy neuron ma 50% szans na zostanie wyłączony. W efekcie w procesie trenowania udział weźmie średnio połowa wszystkich neuronów. Prowadzi to do zjawiska, gdy neuron w trakcie testowania będzie podłączony do co najmniej dwukrotnie liczby neuronów niż podczas nauki, a co za tym idzie wagie będą dwukrotnie większe, takie przeskalowanie sygnałów da rezultaty jak przy rozstrojeniu sieci. Żeby temu zapobiec należy wynikowe wagie po zakończonym treningu pomnożyć przez współczynnik zwany *prawdopodobieństwem utrzymania* (ang. *keep probability*), który wynosi $1 - \rho$, wówczas proporcja sumy wag na wejściu neuronu zostanie podtrzymana. Wartość współczynnika ρ przeważnie jest ustawiana w zakresie od 10% do 50%. Pomiędzy 20% a 30% przy sieciach rekurencyjnych oraz około 40% - 50% podczas pracy z sieciami konwolucyjnymi. Sukces tej metody polega na tym, że w praktyce w każdym przejściu podczas trenowania sieć wygląda inaczej, ponieważ inne neurony są wyłączone. Model jest uczony na wielu wariantach sieci utworzonych z bazowej, pełnej struktury. Okazuje się, że regularyzacja dropout źle wpływa na uczenie z funkcjami aktywacji, które posiadają zdolność samo normalizacji, jak na przykład funkcja SELU. Wówczas należy zastosować dostosowaną odmianę omawianego regularyzatora – *alpha dropout* [21] [3].



Rysunek 15 Zobrazowanie istoty metody Dropout [43]

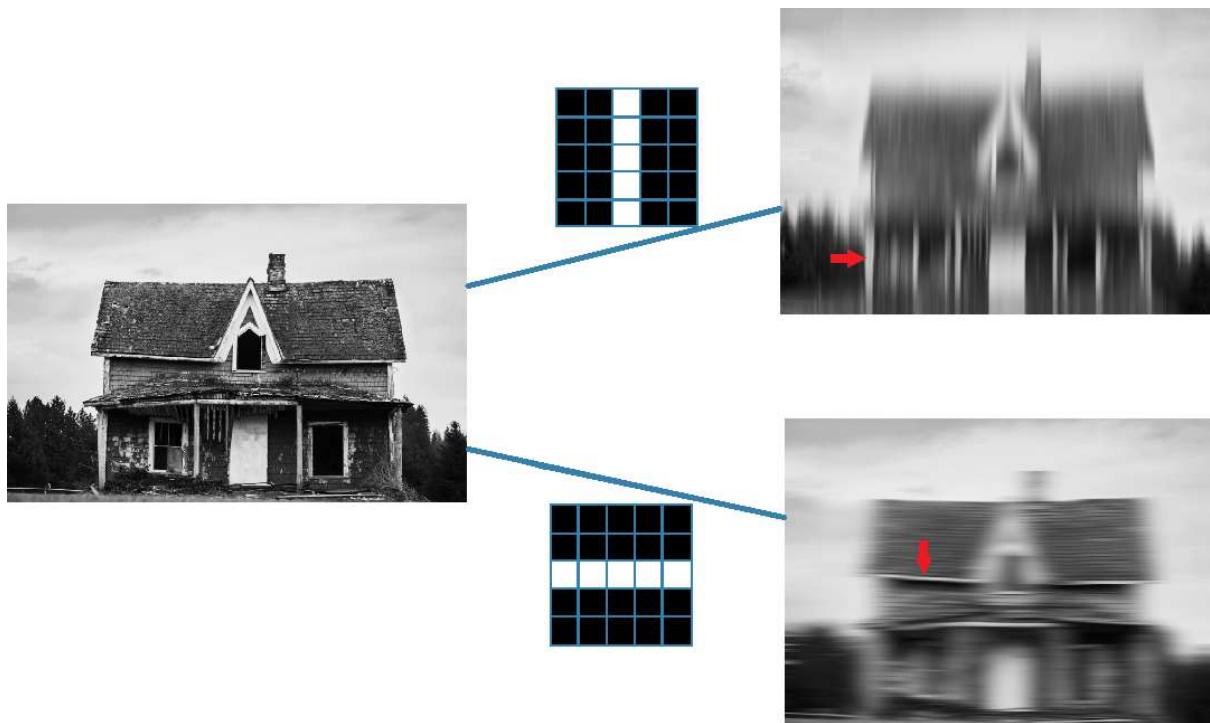
5.2. KONWOLUCYJNE SIECI NEURONOWE

Splotowe lub inaczej konwolucyjne sieci neuronowe (ang. *convolutional neuron networks* – CNN), stanowią najważniejsze narzędzie podczas rozwiązywania zadań rozpoznawania obiektów na obrazach. Zastosowanie tych struktur nie ogranicza się jedynie do pracy z obrazami, ponieważ sieci splotowe doskonale sprawdzają się przy zagadnieniach związanych z sygnałem dźwiękowym, jak np. rozpoznawanie mowy i przetwarzanie języka naturalnego, jednak z racji tematu tej pracy wgląd w sieci konwolucyjne skupi się na ich roli w widzeniu maszynowym. Podobnie jak cała idea sieci, tak również ten konkretny ich rodzaj bazuje na adaptacji zasad przetwarzania obrazu przez mózg. W latach siedemdziesiątych ubiegłego stulecia naukowcy udowodnili na podstawie eksperymentów przeprowadzonych na ssakach, w tym małpach, że tzw. lokalne pola recepcyjne kory wzrokowej są wrażliwe na bodźce odbierane w odpowiadających im fragmentach obszaru widzenia. Obraz jest przetwarzany sekwencyjnie w niewielkich fragmentach a informacja płynąca z niego jest analizowana od szczegółu do ogólnego. Za swoje odkrycie David H. Hubel i Torsten Wiesel otrzymali nagrodę Nobla w 1981 roku. Prace nad nową architekturą sieci zostały wymuszone faktem ogromnego obciążenia dotychczas znanej głębokiej sieci neuronowej, przez dane pochodzące z obrazu, szczególnie gdy ten był obrazem kolorowym o znacznych rozmiarach. Klasyczna sieć głęboka, w której każdy neuron warstwy ukrytej jest połączony z każdym neuronem warstwy niższej, generowała zbyt dużą ilość parametrów, by uczenie, a później testowanie odbywało się w czasie, który można by uznać za efektywny i praktyczny. Warstwa splotowa w sieci CNN nie łączy się ze wszystkimi neuronami poprzedniej warstwy, lecz z tymi, które należą do niewielkiego *pola recepcyjnego*. W kolejnych warstwach połączenia wyglądają analogicznie (Rys. 16). Oznacza to, że neuron pierwszej warstwy splotowej analizuje jedynie fragment obrazu wejściowego i tylko z pikselami tego fragmentu jest połączony. Neurony kolejnej warstwy również łączą się tylko z neuronami w obrębie swojego pola recepcyjnego, czyli analizowany jest większy obszar obrazu. Prowadzi to ostatecznie do uogólnienia i otrzymania informacji z obrazu w całości. To jak duża jest ramka pikseli czy neuronów przypisanych do neuronu w warstwie wyższej, jest parametrem możliwym do strojenia. Oprócz rozmiaru samej ramki można również ustawić krok o jaki ma się ona przesuwać. Jeżeli ustawienia rozmiaru ramki i jego przesunięcia (inaczej *kroku* – ang. *stride*) spowodują, że przy krawędziach ramka będzie wystawała poza obraz, możliwe jest uzupełnienie tych wystających pól zerami (ang. *zero padding*) [10] [3].



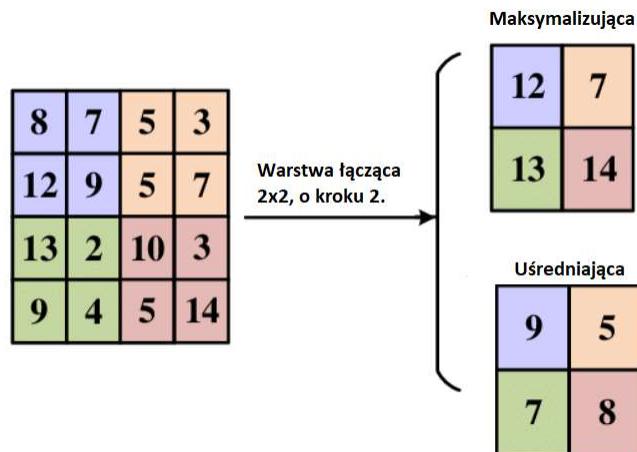
Rysunek 16 Idea działania sieci konwolucyjnej [42]

Wspomniane ramki ograniczające pole receptive nazywane są *jądrami splotowymi* lub w odniesieniu do nazewnictwa angielskiego – kernelami. Gdy do ramki przypiszemy wagę odpowiadającą poszczególnym neuronom czy pikselom, które ta ramka przykrywa stworzymy filtr umożliwiający ekstrahowanie poszukiwanych wzorców na obrazie. Przypuśćmy, że ramka składa się z pięciu kolumn i pięciu wierszy a każdej komórce powstałej w ten sposób przypiszemy wagę. Następnie, gdy myślowo przysłonimy ramką fragment obrazu i wymnożymy wartości pikseli przez odpowiadające im wartości w komórkach ramki, otrzymamy filtr przesuwny służący do wzmacniania poszukiwanych cech obrazu. Jeżeli przykładowo wartości w ramce będą następujące: środkowa kolumna otrzyma wartości 1, a pozostałe otrzymają wartości 0, tak powstały filtr zastosowany dla każdego neuronu w warstwie, wzmacni na jednokanałowym obrazie (obrazie w odcieniach szarości) rozmieszczenie linii pionowych, rozmywając pozostałe tło (Rys.17). Eksperymentowanie z filtrami o innym rozmieszczeniu wag prowadzi do wniosku, że warstwa neuronów zaopatrzonych w ten sam filtr generuje nam obraz uwidaczniający wyraźniej te elementy obrazu, które są zbliżone wyglądem do wzoru na filtrze, a sam efekt filtracji nazywany jest mapą cech (*ang. feature map*). Generowane filtry i mapy cech są układane warstwowo w stosy w obrębie jednej tylko warstwy splotowej i wraz z przechodzeniem do wyższych warstw sieci ich filtry są coraz bardziej złożone, by wyszukiwane wzorce były coraz ogólniejsze.



Rysunek 17 Wpływ filtra pionowego i poziomego na obraz [Źródło: opracowanie własne]

Istnieje jeszcze jeden rodzaj warstwy biorącej wyspecjalizowany udział w trenowaniu CNN, mianowicie warstwa łącząca. Jest to warstwa mająca za zadanie przeskalowanie obrazu wejściowego tak, by zmniejszyć obciążalność obliczeń. Zasada działania tej warstwy jest następująca: podobnie jak warstwa splotowa, tak samo i łącząca posiada neurony, które łączą się tylko z pikselami fragmentu objętego ustaloną ramką. Następnie w obrębie tej ramki zachodzi funkcja przekazująca ostateczny wynik do właściwego neuronu kolejnej warstwy. Do takich funkcji, należą między innymi funkcja uśredniająca lub częściej stosowana funkcja maksymalizująca (Rys. 18). Jej działanie jest bardzo proste, otóż spośród wartości ograniczonych przez wspomnianą ramkę wybiera to pole, które ma największą wartość. Najczęściej korzysta się z warstwa splotowych i łączących ustawionych naprzemiennie co zostanie dokładniej omówione w rozdziale poświęconym architekturom sieci konwolucyjnych.



Rysunek 18 Zasada działania warstwy łączącej – maksymalizującej oraz uśredniającej [23]

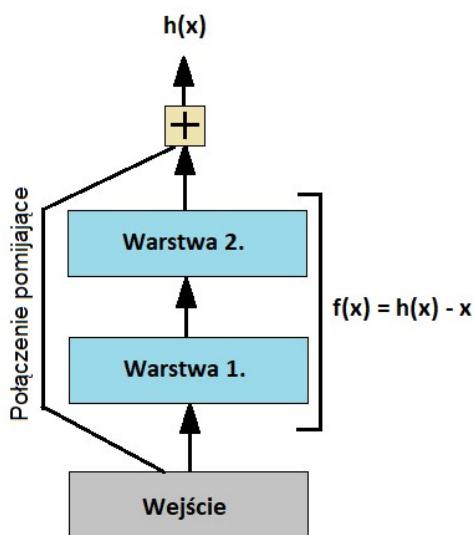
Wspomniano już wcześniej o tym, że sieci splotowe osiągają niższe wykorzystanie mocy obliczeniowej niż ma to miejsce w przypadku konwencjonalnych sieci głębokiego uczenia. W rzeczywistości współczesny sprzęt daje możliwość trenowania i testowania sieci w relatywnie krótkim czasie, jednak nie dysponując zaawansowanym zapleczem technicznym należy liczyć się z treningiem sieci trwającym kilkanaście lub nawet kilkadziesiąt godzin. Głównym problem jest ogromne wykorzystanie pamięci operacyjnej, która w przypadku testowania musi pomieścić jedynie dwie sąsiadujące warstwy natomiast wcześniejsze na bieżąco może usuwać zwalniając pamięć. Niestety w przypadku procesu nauki, pamięć RAM musi pomieścić zawartość wszystkich warstw, by etap propagacji wstecznej mógł się odbyć. Jeżeli nasz sprzęt nie radzi sobie i uczenie zostaje przerywane możliwe jest zmniejszenie podgrupy (ang. *batch*), by w pojedynczym przebiegu analizie podlegała mniejsza ilość danych. Innym rozwiązaniem jest zredukowanie dokładności danych – obrazy 32 bitowe można zdegradować do 16 bitowych.

5.3. ARCHITEKTURY SIECI NEURONOWYCH STOSOWANE DO ROZPOZNAWANIA OBIEKTÓW

Rok 2015 okazał się przełomowy dla metod rozpoznawania obrazów przez sieci neuronowe, opublikowano wówczas dwie najważniejsze obecnie architektury wykorzystywane w omawianej dziedzinie – ResNet oraz YOLO. Są to sieci różniące się od siebie co do zasady działania i dlatego zostaną szczegółowo omówione.

5.3.1. ResNet

Autorzy tej architektury we wspomnianym 2015 roku zwyciężyli wyzwanie ILCVRC, osiągając znakomity wówczas wynik błędu na poziomie poniżej 3.6%. Dokonali tego dzięki zaskakująco głębokiej sieci CNN, składającej się z 152 warstw. Zademonstrowana sieć posiadała dotąd nieznane usprawnienie, otóż niektóre warstwy były dodatkowo łączone ze sobą z pominięciem warstwy pośredniej (ang. *Skip connections*), co zostało zaprezentowane na ilustracji (Rys.19).



Rysunek 19 Schemat ilustrujący istotę działania metody „Skip connection” [Źródło: opracowanie własne]

Taki typ sieci sprawił, że trenowany model nie zbliżał się do odwzorowania funkcji celu $h(x)$, jak dotychczas, lecz odwzorowywał funkcję $f(x) = h(x) - x$; gdzie x jest wartością wejściową dodawaną do wyjścia sieci poprzez połączenie pomijające. Takie trenowanie sieci nazywa się uczeniem rezydualnym (ang. *Residual learning*) i tworzy rodzinę struktur wykorzystujących pomija nie warstw (stąd również pochodzi nazwa *ResNet*). Taka strategia niesie ze sobą wiele korzyści. Po pierwsze często bywa tak, że docelowa funkcja jest funkcją zbliżoną do tożsamościowej, wówczas sygnał wyjściowy niewiele różni się od wejściowego. Jeżeli zastosujemy połączenia pomijające, początkowo nasz model staje się tożsamościowy, ponieważ wejście jest bez zmian przenoszone aż do warstwy wyjściowej (szczególnie, że przeważnie wagi początkowe są bardzo niewielkie i nie zwiększały sygnału). Prowadzi to do sytuacji, gdy model znacznie szybciej osiąga kształt zbliżony do docelowego. Dodatkowo omawiane omijanie warstw przyspiesza rozprzestrzenianie się sygnału wejściowego po całej sieci i powoduje niemal jednocześnie rozpoczęcie trenowania wszystkich warstw. Istnieją warianty sieci ResNet różniące się ilością warstw głębokich, a ich wybór jest podektywany złożonością zadania. Struktura sieci ResNet jest następująca: po przejściu przez pierwszą warstwę splotową i łączącą (maksymalizującą), sygnał przechodzi przez wiele warstw splotowych, w tym także przez połącznie pomijające. Każda z nich wykorzystuje normalizację wsadową oraz funkcję aktywacyjną ReLU. Co kilka warstw ilość filtrów w warstwie zwiększa

się dwukrotnie, ale również rozmiar obrazu jest pomniejszany do połowy szerokości i wysokości dzięki warstwie splotowej o kroku równym 2. Ostatnie warstwy tworzą: globalna uśredniająca, jedna warstwa w pełni połączona oraz warstwa z funkcją Softmax, jest to funkcja podejmująca decyzję o przynależności do klasy. Detekcja przy pomocy takiej sieci polega na przesuwaniu ramki ograniczającej z zadanym krokiem po obrazie i ocenie czy wewnątrz ramki znajduje się znany sieci obiekt. Ramka przesuwając się z krokiem równym 1, często wielokrotnie obejmuje ten sam obiekt tworząc wiele ramek predykcji. Ramki, które najsłabiej obejmują obiekt są na koniec usuwane i zostaje jedna o najlepszym dopasowaniu. Jeżeli na obrazie znajduje się wiele obiektów różniących się rozmiarem należałyby powtórzyć skanowanie z wykorzystaniem większych ramek. Przedstawiona strategia charakteryzuje działanie tego rodzaju sieci i widać na pierwszy rzut oka, że algorytm jest czasochłonny, ponieważ obraz jest analizowany wielokrotnie. W dalszej części przedstawiona zostanie sieć o innym działaniu – znacznie przyspieszająca detekcję nawet wielu obiektów przez jednorazową analizę obrazu.

Omawiając architektury sieci należy wyróżnić pojęcie *Backbone*, służy ono do nazwania części sieci obsługującej ekstrakcję cech z danych wejściowych. Istnieją różne sieci typu backbone i ResNet należą nich, innym przykładem jest struktura *Darknet*. Części ekstrahujące cechy (backbone) można łączyć z różnymi strukturami strojącymi wagi sieci. Najczęściej, jeżeli korzysta się z ekstraktora ResNet, to w połączeniu z siecią RetinaNet, natomiast backbone Darknet pracuje najczęściej z architekturami YOLO (omówionymi w dalszej części), ze względu na wysoką efektywność takiej kompozycji [24].

5.3.2. YOLO

Nazwa architektury pochodzi z języka angielskiego i stanowi akronim zwrotu *You Only Look Once*, czyli „*patrzysz tylko raz*”. Zasada ta jest istotą szybkości działania sieci, która znacząco wybija się spośród konkurencji w dziedzinie klasyfikacji i lokalizacji obiektów na obrazach. Sieć tego pomysłu różni się od opisywanej wcześniej sieci ResNet tym, że stanowi sieć *w pełni połączoną* (ang. *Fully connected network - FCN*), znaczy to tyle, że najwyższe warstwy gęste zostały z sukcesem zastąpione warstwami splotowymi. Na skutek tego działania powstała architektura, która stała się standardem stosowanym w zadaniach detekcji ze względu na swoją szybkość. Jak już wspomniano sieci CNN działają na zasadzie przesuwania ramek ograniczających o krok mniejszy niż wymiar ramki i każdorazowej analizie ich zawartości. W przypadku algorytmu YOLO sytuacja wygląda nieco inaczej. Obraz od razu dzielony jest na mniejsze obszary i każdy z nich generuje osobny wektor zawierający siedem wartości – informację o tym, czy obiekt (bądź fragment) znajduje się w danym segmencie, współrzędne środka ramki okalającej obiekt (ang. *bounding box*), wysokość i szerokość ramki oraz klasę przynależności obiektu. Jeżeli w jednym segmencie znajdują się więcej niż obiekt jednej klasy to wektor ma dwa razy większy rozmiar i zawiera informacje o obu obiektach bez znaczenia

czy są tej samej klasy czy różnych. Dzięki temu detekcja kilku obiektów na jednym obrazie nie stanowi przeszkody. Może się zdarzyć, że jeden obiekt zostanie otoczony więcej niż jedną ramką okalającą, wówczas z pomocą przychodzi indeks Jaccarda. Jest to współczynnik pozwalający ocenić jakość dopasowania obwiedni do obiektu i stanowi iloraz części wspólnej obwiedni ocenianej i referencyjnej, oraz ich sumy pól. Nie wybieramy więc obwiedni o największym dopasowaniu, ponieważ moglibyśmy pominąć oznaczenie obiektu tej samej klasy umieszczonego w innym miejscu obrazu, i mniejszym dopasowaniu. Architektura YOLO jest stale rozwijana dając coraz lepsze efekty, w tej pracy zostaną omówione i porównane jej dwie wersje YOLOv3 oraz YOLOv4.

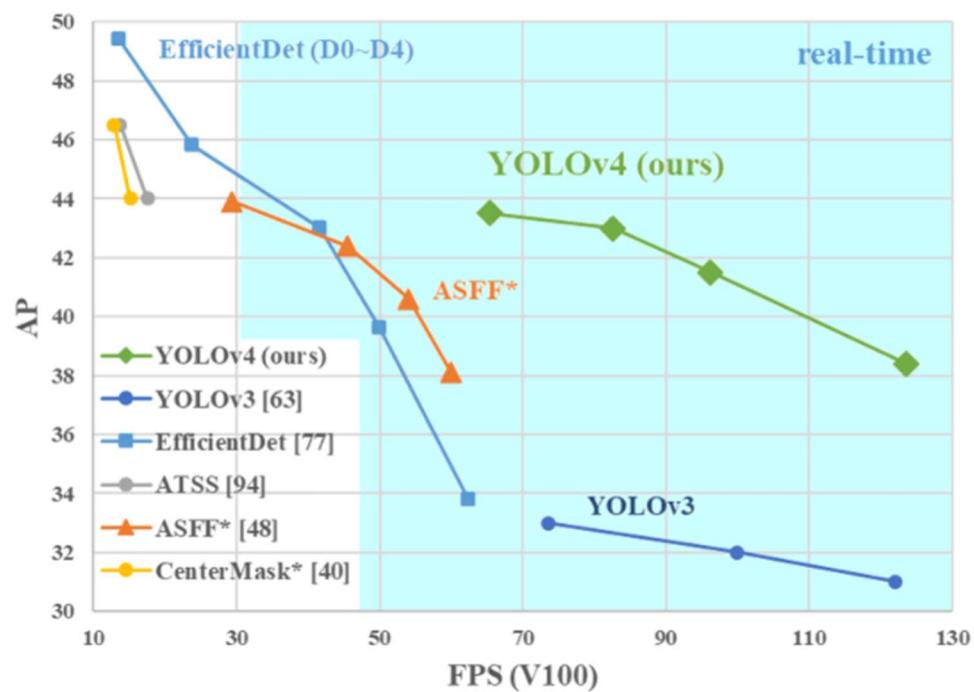
Starsza, YOLOv3 jest architekturą, która wykorzystuje połączenia pomijające. Podczas pracy przekształca obrazy i tworzy mapy cech obrazów w trzech skalach. Dodatkowo w wersji tej, w porównaniu do poprzednich zrezygnowano z funkcji Softmax, ponieważ zdano sobie sprawę, iż dobre efekty przynosi regresja logistyczna podczas podejmowania decyzji o przynależności obiektu do klasy. Nowością dotyczącą tej sieci jest również używany w połączeniu z nią backbone, ponieważ w YOLOv2 korzystano z Darknet-19 natomiast YOLOv3 obsługuje wyższa wersja – Darknet-53, posiadająca 53 warstwy konwolucyjne.

Nowymi cechami, które zostały dodane do wersji YOLOv4 jest dodatkowy blok łączący backbone z częścią trenującą, nazywany *Neck*. Część trenująca wagi jest użyta ta sama, co w poprzedniej wersji. Zadaniem bloku Neck jest generowanie odpowiednich rozmiarów map cech nieważne jakich rozmiarów były dane wejściowe. Dodatkowymi usprawnieniami w tej wersji było rozszerzenie modyfikowania obrazów uczących przez dodanie między innymi tworzenie mozaik czy nakładanie na siebie fragmentów różnych obrazów. Dodatkowo zastosowano regularyzację typu DropBlock, czyli zamiast wyłączania pojedynczych neuronów z procesu trenowania podczas poszczególnych iteracji, wyłączane są wspólnie sąsiadujące komórki, znajdujące się w niewielkich regionach. Wszystkie te zabiegi usprawniające poprawiły działanie sieci względem poprzedniej wersji zwiększąc współczynnik mAP (średnia precyzja) o 10%, oraz szybkość detekcji w czasie rzeczywistym o 12% [25].

Podsumowując, można dostrzec, iż sieci są stale rozwijane w różnych kierunkach by poprawiać działanie detekcji oraz klasyfikacji. Z wykresów (Rys. 20) i (Rys. 21) wynika różnorodność cech i potencjalnego zastosowania poszczególnych modeli. Zobrazowano wyraźnie, że sieci ResNet działają znacznie wolniej od sieci YOLO, jednak z o wiele wyższą precyją. Biorąc pod uwagę, że obiektem do rozpoznawania w niniejszym projekcie są znaki drogowe, szybkość ich rozpoznawania na drodze w czasie jazdy jest bardzo ważna, dlatego eksperyment zostanie wykonany z wykorzystaniem sieci z rodziny YOLO [26].



Rysunek 20 Zestawienie architektur sieci dla porównania ich szybkości i dokładności działania [38]



Rysunek 21 Zobrazowanie jak szybkość pracy wpływa na dokładność dla różnych sieci [39]

5.4. MIARY WYDAJNOŚCI

Do oceny pracy modelu powstały różne wskaźniki i w niniejszym rozdziale zostaną przedstawione te najważniejsze.

5.4.1 Sprawdżian krzyżowy

Sprawdżian krzyżowy (zwany inaczej krosowalidacją) jest to metoda polegająca na sprawdzeniu jak nasz model potencjalnie może radzić sobie z danymi, z którymi nie miał do czynienia, przy pomocy danych uczących. Zasada działania jest następująca: zbiór uczący jest dzielony na k podzbiorów. Następnie model jest trenowany na $k-1$ podzbiorach i na koniec testowany na pozostałym zbiorze. Żeby mieć pewność reprezentatywności takich podzbiorów testujących – całość operacji jest powtarzana k razy, aby model był uczony i testowany na wszystkich podzbiorach. W zbiorach, w których jedne klasy występują znacznie częściej niż inne, ten test nie da wiarygodnej oceny modelu, jednak, jeżeli rozkład klas w bazie danych jest równomierny to krosowalidacja jest przydatnym i często stosowanym narzędziem.

5.4.2. Macierz pomyłek

Jest to technika znacznie dokładniejsza i niosąca więcej informacji niż krosowalidacja. Polega na utworzeniu tablicy, w której kolumny stanowią przewidywania modelu wobec prezentowanych mu danych, a wiersze odpowiadają wartościom rzeczywistym (Rys. 22). Jeżeli weźmiemy za przykład ocenę modelu klasyfikatora binarnego, to otrzymamy tablicę o wymiarach 2×2 . Pierwsza kolumna stanowić będzie przykłady przewidziane przez model jako pozytywne, druga kolumna negatywne. Co do wierszy, to pierwszy dotyczy wartości rzeczywistych pozytywnych, drugi zaś negatywnych. Podsumowując, model może rozpoznać prawidłowo wartość pozytywną (TP), jak i negatywną (TN), lub też pomylić się i fałszywie określić wartość pozytywną jako negatywną (FN), jak i odwrotnie (FP). Z takiej reprezentacji danych można wyczytać ciekawe informacje np. w modelu pracującym idealnie wartości nieujemne znajdowałyby się jedynie w głównej przekątnej takiej tablicy. Aby łatwiej się było zorientować w wynikach utworzonej macierzy zaproponowano dwa dodatkowe współczynniki, które na niej bazują. Tymi współczynnikami są precyzja (ang. *precision*) i czułość modelu (ang. *recall*), które zostaną szczegółowo opisane w dalszej części.

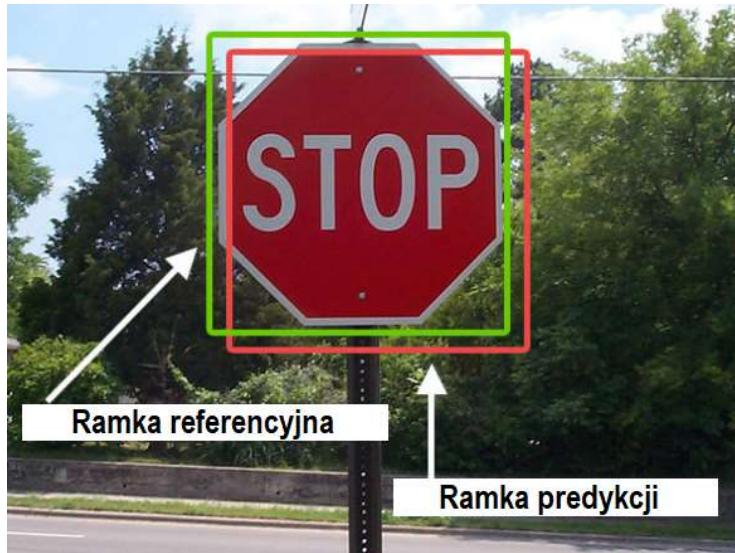
| | | Klasa predykowana – wynik testu | |
|-------------------|----------------|---|---|
| | | Klasyfikacja pozytywna | Klasyfikacja negatywna |
| Klasa rzeczywista | Populacja | | |
| | Stan pozytywny | prawdziwie dodatnia, TP | fałszywie ujemna (błąd drugiego rodzaju, FN) |
| | Stan negatywny | fałszywie dodatnia (błąd pierwszego rodzaju, FP) | prawdziwie ujemna, TN |

Rysunek 22 Tablica pomyłek służąca do obliczenia wskaźników – precyzji oraz czułości [44]

5.4.3. Precyzja a czułość modeli

Precyzja i czułość modelu są wartościami wynikającymi z analizy macierzy pomyłek. Precyzja jest stosunkiem wartości prawidłowo zaklasyfikowanych jako pozytywne (TP), do sumy wszystkich wartości rozpoznanych jako pozytywne (TP+FP). Wskaźnik ten jednak nie wyczerpuje wszystkich informacji płynących ze wspomnianej macierzy, dlatego powstał jeszcze jeden współczynnik – czułość i stanowi stosunek prawidłowo rozpoznanych wartości pozytywnych (TP) do ilościowej sumy tych wartości oraz tych rozpoznanych fałszywie jako negatywne (TP+FN). *Precyzja* określa jak często model myli się przypisując za dużo wartości do zbioru prawdziwie pozytywnych, natomiast *czułość* określa jak wrażliwy jest model przy klasyfikowaniu obiektów jako pozytywne spośród wszystkich pozytywnych. Istnieje trzecia metoda oceny klasyfikatorów, szczególnie pomocna przy ich porównywaniu, zawierająca w sobie zarówno czułość jak i precyzję, otóż współczynnik o nazwie F1 jest średnią harmoniczną tych wymienionych wcześniej. Jego wadą może okazać się to, że najwyższą wartość otrzymują klasyfikatory o zbliżonej wartości precyzji i czułości, co nie zawsze jest pożądane z punktu widzenia problemu do rozwiązania. Często zależy nam bardziej na precyzji modelu a czasem na czułości, np. chcemy, żeby model rzadziej się mylił lub żeby nawet kosztem pomyłek nie pomijał żadnych obiektów, czyli był czulszy. Kompromis pomiędzy tymi dwiema wartościami jest kluczowy w prawidłowym dostrojeniu modelu i ustala się go przez przyjęcie odpowiedniego progu decyzyjnego. Można go wyznaczyć np. z wykresu krzywych precyzji i czułości w funkcji progu decyzyjnego i po wybraniu preferowanej wartości precyzji odczytać wartość progową, zgodnie z którą wartość jest wg. funkcji decyzyjnej zaklasyfikowana jako wartość pozytywna albo negatywna [10] [3].

5.4.4. Intersection over Union (IoU)



Rysunek 23 Przykład działania detektora obiektów dający możliwość obliczenia współczynnika IoU [46]

Wymienione metody są skuteczne w ocenie modeli klasyfikatorów, jednak, gdy należy wyciągnąć wnioski na temat pracy detektora obiektów na obrazach, nie wystarczy nam informacja czy obiekt na obrazie jest obecny i czy został przydzielony do prawidłowej klasy, lecz również czy jego położenie zostało prawidłowo wskazane na obrazie. W takiej sytuacji wykorzystuje się wspomniany już indeks Jaccard-a, inaczej zwany współczynnikiem IoU (ang. *Intersection over Union*). Jest to stosunek powierzchni części wspólnej ramki predykcji i referencyjnej do powierzchni jaką zajmują w sumie (Rys. 23). Najczęściej, jeżeli wartość IoU wynosi powyżej 0.7, to uznaje się dokładność wskazania za dobrą. Poniżej zaprezentowano fragment kodu napisanego w języku Python realizującego wyliczenie współczynnika IoU (Rys. 24) [27].

```
def intersection_over_union(boxA, boxB):
    # wyznaczenie współrzędnych pierwszych narożników ramek
    xA = max(boxA[0], boxB[0])
    yA = max(boxA[1], boxB[1])
    xB = min(boxA[2], boxB[2])
    yB = min(boxA[3], boxB[3])
    # obliczenie części wspólnej obu ramek
    interArea = max(0, xB - xA + 1) * max(0, yB - yA + 1)
    # obliczenie sumy pól obu ramek
    # poszczególne ramki
    boxAArea = (boxA[2] - boxA[0] + 1) * (boxA[3] - boxA[1] + 1)
    boxBArea = (boxB[2] - boxB[0] + 1) * (boxB[3] - boxB[1] + 1)
    # IoU
    iou = interArea / float(boxAArea + boxBArea - interArea)
    # zwrot wartości przez funkcję
    return iou
```

Rysunek 24 Kod funkcji realizującej wyliczenie IoU [Źródło: opracowanie własne]

5.4.5. Funkcja straty (*Loss*)

Funkcja Loss jest ważnym narzędziem wykorzystywanym do oceny procesu trenowania. Stanowi sumę kwadratów błędów pomiędzy wartością predykcji a rzeczywistą etykietą. Współczynnik Loss jest wartością wyświetlaną w procesie trenowania celem oceny w jakim kierunku zmierza nauka – czy model stale ewoluje, czy już osiągnął granicę możliwości jakie dały dane trenujące i algorytm. Możliwe jest stwierdzenie, że model w ogóle nie rozpoczął nauki, jeżeli wartość Loss w kolejnych iteracjach się nie zmienia. Pozwala dostosować wartość współczynnika uczenia, można zwiększyć jego wartość, jeżeli widać, że model uczy się powoli, co obrazuje płaski wykres i powolny spadek krzywej funkcji Loss. W czasie trenowania generowanych jest wiele ramek predykcji jednak wybierane są te o najwyższym współczynniku IoU w odniesieniu do etykiet. W ten sposób dochodzi do ich specjalizacji w rozpoznawaniu konkretnych rozmiarów i stosunku szerokości do wysokości obiektów. Funkcja Loss składa się z trzech mniejszych funkcji:

- funkcja straty w odniesieniu do klasyfikacji obiektów, stanowi kwadratowy błąd prawidłowego sklasyfikowania obiektu i wartości rzeczywistej, wzór,

$$\sum_{i=0}^{S^2} \mathbf{1}_i^{obj} \sum_{c \in classes} (p_i(c) - \hat{p}_i(c))^2 \quad (0)$$

- funkcja straty w odniesieniu do lokalizacji ramki predykcji, stanowi błąd kwadratowy pomiędzy rozmiarami i rozmieszczeniem ramki predykcji a ramką referencyjną (15),

$$\lambda_{coord} \sum_{i=0}^{S^2} \sum_{j=0}^B \mathbf{1}_{ij}^{obj} \left[(x_i - \hat{x}_i)^2 + (y_i - \hat{y}_i)^2 \right] + \lambda_{coord} \sum_{i=0}^{S^2} \sum_{j=0}^B \mathbf{1}_{ij}^{obj} \left[(\sqrt{w_i} - \sqrt{\hat{w}_i})^2 + (\sqrt{h_i} - \sqrt{\hat{h}_i})^2 \right] \quad (15)$$

- funkcja straty w odniesieniu do pewności co do zawartości ramki predykcji, stanowi kwadratowy błąd między współczynnikiem *Confidence* a faktycznym stanem tego co otacza ramka (16).

$$\sum_{i=0}^{S^2} \sum_{j=0}^B \mathbf{1}_{ij}^{obj} (C_i - \hat{C}_i)^2 + \lambda_{noobj} \sum_{i=0}^{S^2} \sum_{j=0}^B \mathbf{1}_{ij}^{noobj} (C_i - \hat{C}_i)^2 \quad (0)$$

Te trzy niezależne wskaźniki oceny jakości predykcji sumują się na całkowitą postać funkcji Loss, a celem trenowania modelu jest minimalizacja tej funkcji (17) [28] [29] [30].

$$\begin{aligned}
 \text{Loss}_{\text{yolo}} = & \lambda_{\text{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^B \mathbb{1}_{ij}^{\text{obj}} [(x_i - \hat{x}_i)^2 + (y_i - \hat{y}_i)^2] \\
 & + \lambda_{\text{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^B \mathbb{1}_{ij}^{\text{obj}} [(\sqrt{w_i} - \sqrt{\hat{w}_i})^2 + (\sqrt{h_i} - \sqrt{\hat{h}_i})^2] \\
 & + \sum_{i=0}^{S^2} \sum_{j=0}^B \mathbb{1}_{ij}^{\text{obj}} (C_i - \hat{C}_i)^2 + \lambda_{\text{noobj}} \sum_{i=0}^{S^2} \sum_{j=0}^B \mathbb{1}_{ij}^{\text{noobj}} (C_i - \hat{C}_i)^2 \\
 & + \sum_{i=0}^{S^2} \mathbb{1}_i^{\text{obj}} \sum_{c \in \text{classes}} (p_i(c) - \hat{p}_i(c))^2
 \end{aligned} \tag{17}$$

W powyższych wzorach:

S^2 – rozmiar mapy cech (ilość komórek w mapie),

B – liczba zakotwiczeń (ang. *anchor boxes*),

x_i, y_i – współrzędne centrum zakotwiczenia,

w_i, h_i - szerokość i wysokość zakotwiczenia,

C_i – obiektywość (współczynnik confidence),

$p_i(c)$ – jakość klasyfikacji,

$\mathbb{1}_{ij}^{\text{obj}}$, $\mathbb{1}_{ij}^{\text{noobj}}$ – markery przyznawane każdej komórce mapy cech, informują czy w danej komórce znajduje się klasyfikowany obiekt, czy nie. Dzięki temu możliwe jest egzekwowanie kary za predykcję obiektu w obszarze obrazu, w którym jest on nieobecny.

6. DANE UCZĄCE I TESTUJĄCE - POZYSKANIE I OPRACOWANIE

6.1. DATA MINING

Dane liczbowe są zdecydowanie powszechnie dostępne jako wynik uboczny operowania na bazach danych, tworzonych i rozwijanych na całym świecie przez niemal każdą firmę w obrębie jej działalności. Powstają tzw. hurtownie danych (ang. data warehouse), służące analitykom do poszukiwania dodatkowych związków. Niestety do zadań detekcji i klasyfikacji potrzebne są obrazy, te nie są już tak łatwo dostępne w zadowalającej ilości. Choć wydawać się mogło, że przy pomocy wyszukiwarki Google Grafika można zaspokoić potrzeby procesu uczenia maszynowego, o tyle trzeba zdać sobie sprawę, że nawet to źródło nie dostarczy odpowiedniej ilości zdjęć obiektów specyficznych. Omawiany sposób zdobywania informacji ze środowisk rzeczywistości wirtualnej, jest przykładem nowoczesnego i praktycznego podejścia do dziedziny jaką jest Data Mining.

6.2. ZNAKI DROGOWE – OBIEKT DETEKCJI

Wybór znaków drogowych do przeprowadzenia testów nie był przypadkowy, lecz wynikał z wielu cech tych obiektów przemawiających za tym, że proces uczenia modeli będzie przebiegał bez utrudnień. Ogromną zaletą znaków drogowych jest powszechny dostęp w świecie rzeczywistym, jak również istnieją symulacje w postaci gier komputerowych z rozbudowaną i dopracowaną infrastrukturą drogową, której zadanie jest odwzorowanie warunków drogowych w największym stopniu. Znaki drogowe są obiektami wyjątkowymi, ponieważ rola, do pełnienia której zostały stworzone w samym założeniu wymaga ich jak najlepszej widoczności i zwracania uwagi oka ludzkiego. Osoby prowadzące samochód muszą w pełni kontrolować swoje pole widzenia, by uniknąć niebezpiecznych sytuacji lub by samemu ich nie stwarzać. Dodatkowa obserwacja oznakowania drogi w przypadku, gdyby było ono słabo widoczne powodowałoby, że znaki umykałyby uwadze kierowcy. W związku z tym powstały dokładne wytyczne co do kształtu i koloru znaków drogowych zawarte w krajowych zarządzeniach, jak również w konwencjach międzynarodowych [31] [32].

Znaki drogowe w Europie na kanwie tych konwencji różnią się od siebie w bardzo niewielkim stopniu i stanowią przede wszystkim znaki graficzne, w przeciwieństwie na przykład do oznakowania w Stanach Zjednoczonych, gdzie organizacja ruchu bazuje w dużej mierze na komunikatach tekstowych. Europejskie podejście do graficznego prezentowania obowiązujących przepisów sprzyja osobom nieznającym lokalnego języka. Z punktu widzenia eksperymentu jest to również istotne, ponieważ musimy poddać pod zastanowienie amerykańskie produkcje gier do tworzenia baz trenujących obrazów.

Najważniejsze zalety wyboru znaków drogowych na obiekty detekcji podczas eksperymentu porównywania modeli:

- Powszechna dostępność w świecie rzeczywistym,
- Obecność znaków w wielu grach i programach komputerowych,
- Jednoznaczność koloru, kształtu i proporcji wymiarów większości znaków w całej Europie,
- Kolory i kształty są kontrastujące z otoczeniem.

Istniały badania rozpoznawania znaków drogowych z uprzednim przekształceniem ich do skali szarości, jednak zgodnie z opiniami badaczy lepsze efekty daje rozpoznawanie znaków w ich oryginalnej trójkątowej formie, co z wykorzystaniem dzisiejszego sprzętu nie stanowi problemu [33]. Do przeprowadzenia eksperymentu zdecydowano się wybrać trzy znaki różniące się od siebie kształtem oraz kolorystyką, w oparciu o te proste kryteria wybrano następujące znaki (Rys. 25):

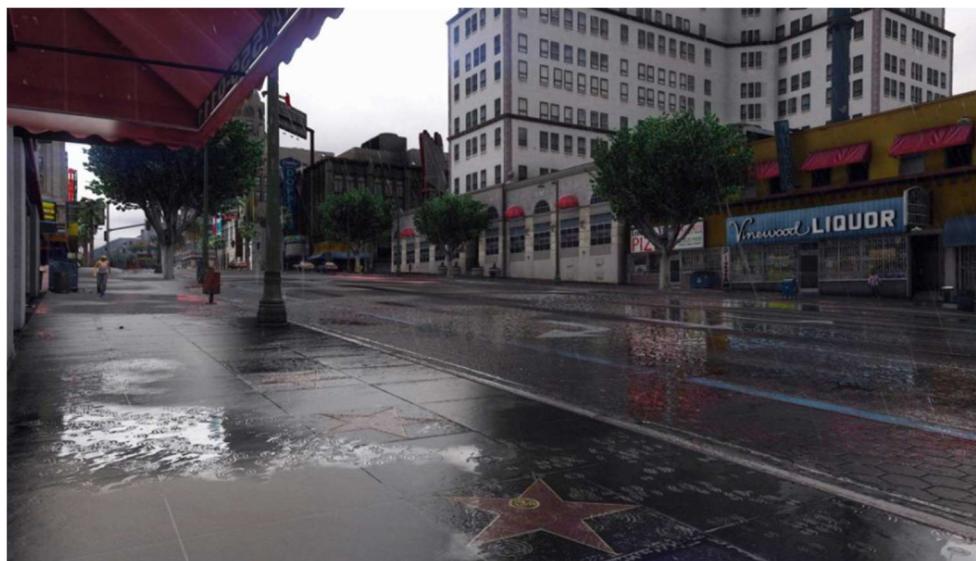


Rysunek 25 Znaki drogowe wybrane do eksperymentu. Od lewej Ustęp pierwszeństwa(A-7), Droga z pierwszeństwem(D-1), Zakaz wyprzedzania(B-25) [47]

6.3. POZYSKANIE DANYCH UCZĄCYCH ZE ŚRODOWISKA WIRTUALNEGO

Wybór środowiska jako źródła obrazów jest pierwszym, a za razem kluczowym etapem. Należy się zastanowić, czy istnieje gotowy program, gra, czy symulacja, ponieważ wykluczy to tworzenie nowej przestrzeni wirtualnej na potrzeby projektu. Jeżeli nie istnieje satysfakcyjująca symulacja możliwe jest skorzystanie z programu do modelowania trójwymiarowego z opcją nakładania dowolnej tekstury i tła. Umożliwia to urozmaicenie ujęć poprzez szybkie i wygodne zmiany otoczenia obiektu i co najważniejsze, możliwa jest wygodna konfiguracja oświetlenia sceny, zgodnie z prawami optyki. Zmienne warunki oświetleniowe stanowią najtrudniejsze wyzwanie dla dziedzin zajmujących się dynamiczną analizą obrazu w czasie rzeczywistym. Jeżeli dane uczące są ubogie i nieurozmaicone, a więc stanowią zdjęcia obiektu oświetlonego równomiernie z tym samym natężeniem, to należy się spodziewać, że nawet nieznaczne zacienienie obiektu może sprawić, że obiekt ten nie zostanie prawidłowo zlokalizowany i zaklasyfikowany na obrazie.

W ramach eksperymentu zdecydowano się wybór znaków drogowych jako obiektów do oceny jakości rozpoznawania przez modele. Wybór ten był podyktowany w dużej mierze powszechną obecnością tych obiektów w wielu produkcjach rozrywkowych łatwo dostępnych. Ponadto współczesne gry odznaczają się wysoką jakością modeli i rozdzielczością obrazu. Obecnie najbardziej zaawansowane pod względem jakości grafiki i za razem traktujące o motoryzacji są dwie znaczące pozycje – gra GTA V (Rys. 26), wspomniano o niej w przeglądzie, oraz symulator jazdy ciężarówką po Europie – EuroTruck Simulator 2 (Rys. 27). Obie produkcje umożliwiają poruszanie się po otwartym i dopracowanym w szczegółach świecie. Istnieją jednak aspekty techniczne, przez którymi EuroTruck Simulator wypada lepiej. Po pierwsze gra GTA V osadzona jest w świecie odwzorowującym warunki na drogach w Stanach Zjednoczonych, co utrudniałoby przeprowadzenie testów na znakach drogowych według rodzimego systemu oznakowania dróg. Kolejną kwestią jest to, że Euro Truck Simulator umożliwia ustawienie preferowanej konfiguracji zjawisk pogodowych i pór dnia, cecha ta przyczynia się do optymalizacji czasu pobierania widoków o wysokim stopniu zróżnicowania. Na koniec warto zauważyć, że symulator jazdy samochodem ciężarowym stawia ogromny nacisk na zachowanie wszystkich zasad i przepisów o ruchu drogowym, oznakowanie dróg jest szczegółowe i dokładne, by gracz mógł poruszać się z jak największym poczuciem realizmu. Z wymienionych powodów zdecydowano się na użycie w eksperymencie symulatora jazdy EuroTruck Simulator 2.



Rysunek 26 Widok ekranu gry Grand Theft Auto V [48]



Rysunek 27 Widok ekranu gry Eurotruck Simulator 2 [Źródło: opracowanie własne]

Program dostarczył wysokiej jakości obrazy w rozdzielczości 1920x1080 piksli, co zdecydowanie przekracza wymagania potrzebne do stworzenia wiarygodnego modelu. W związku z czym zdjęcia zostały pomniejszone do rozdzielczości 1280x720. W dziedzinie uczenia maszynowego taki rozmiar nadal jest uważany za duży i wydłużający czas uczenia, jednak wpływa również na dokładność modelu, dlatego uznano go za właściwy. Jak wspomniano EuroTruck Simulator 2 daje możliwość zmiany warunków atmosferycznych, a dokładniej częstotliwość zachodzenia ich zmian. Funkcja ta dała możliwość stworzenia bazy danych obejmującej szerokie spektrum warunków drogowych, to na czym się skupiono, to aby zdjęcia przedstawiały jedynie pory dnia z pominięciem nocy, wyróżniamy więc ranek i wieczór, kiedy Słońce jest najniżej powodując często utrudnienia w widoczności, gdy jest w tle, oraz porę dnia, gdy Słońce jest wysoko i wówczas nie oślepia kierowców. Bywa, że przedmioty stojące na drodze promieni światła zacieniają częściowo lub całkowicie znaki drogowe, takie widoki również są uwzględnione w powstałej bazie danych. Co do warunków atmosferycznych. Tutaj udało się wyróżnić trzy podstawowe stany pogodowe – słoneczne, bezchmurne warunki, jak również zachmurzenie oraz opady deszczowe z zachmurzeniem. Na rysunku 17 ukazano porównanie zdjęć wykonanych o różnych porach dnia, jak i różnej aurze pogodowej. Ze zdjęć wynika, że odwzorowana w grze została ciepłapora roku (Rys. 28).



Rysunek 28 Gra Eurotruck Simulator 2 zapewnia zmiennosć pory dnia i warunków atmosferycznych na wysokim poziomie odwzorowania szczegółów.

6.4. POZYSKANIE DANYCH UCZĄCYCH ZE ŚRODOWISKA RZECZYWISTEGO

Jak wielokrotnie podkreślano, pozyskiwanie zdjęć ze środowiska wirtualnego stanowi doskonałą alternatywę dla rzeczywistych sesji w terenie, gdyż niektóre obiekty w naturze mogą występować rzadko. Znaki drogowe nie należą do rzadko występujących czy trudno dostępnych obiektów, jednak wbrew intuicji nie ma to znaczenia dla powodzenia eksperymentu, ponieważ, celem jest porównanie efektywności pracy modeli a znaki drogowe są tylko wygodną i łatwo dostępną miarą tej efektywności. Szczególnie, dlatego łatwo dostępną, ponieważ powszechnie znana usługa Google Maps daje możliwość rzeczywistego widoku dróg z niemal całego świata (Rys. 29). Narzędzie to umożliwia sprawne skompletowanie zbioru uczącego, bez konieczności mozolnego wykonywania zdjęć samodzielnie w terenie.



Rysunek 29 Widok ekranu aplikacji Google Maps [Źródło: opracowanie własne]

Obrazy otrzymane tą metodą zostały rozmiarowo dostosowane do swoich odpowiedników z bazy danych pochodzenia wirtualnego, czyli zostały zmniejszone do rozdzielczości 1280x720 pikseli. Ich zawartość nie jest tak zróżnicowana, jeśli chodzi o porę dnia i warunki atmosferyczne. Wynika to z trudności odnalezienia w aplikacji Google Maps zdjęć o preferowanych wymienionych warunkach. W związku z tym w zebranym zbiorze nie można wyróżnić wyraźnie wszystkich pór dnia, istnieją zdjęcia z godzin, gdy Słońce znajduje się wysoko oraz z wczesnego popołudnia. Większa różnorodność dotyczy zachmurzenia co zaprezentowano na rysunku 30. W bazie zdjęć rzeczywistych nie pojawiają się te wykonane podczas opadu deszczu, a wszystkie zdjęcia są wykonane w ciepłych miesiącach roku.



Rysunek 30 Zdjęcia pozyskane z Google Maps różnią się aurą pogodową w tle i oświetleniem obiektów [Źródło: opracowanie własne]

Przemierzanie ulic miast i okolic w aplikacji Google Maps poskutkowało powstaniem w niedługim czasie bazy zawierającej taką samą ilość zdjęć tych samych znaków drogowych, co zbiór zebrany w środowisku wirtualnym. W rezultacie powstały dwa zbiory – pochodzący z symulacji i z rzeczywistości, każdy zawierał po 150 obrazów każdego z trzech znaków: „Droga z pierwszeństwem” (D-1), „Ustęp pierwszeństwa” (A-7) oraz „Zakaz wyprzedzania” (B-25).

6.5. DANE TESTUJĄCE

Dane testujące pochodzą ze środowiska rzeczywistego, lecz w odróżnieniu od części danych trenujących pochodzących z rzeczywistości, dane testujące zostały pozyskane poprzez własnoręczne wykonywanie zdjęć. Takie podejście wynikło z założenia, iż zdjęcia testowe miały różnić się nieznacznie od zdjęć trenujących, lecz nadal być obrazami rzeczywistymi. Udało się to uzyskać dzięki temu, że porą roku wykonywania zdjęć testowych była jesień, a więc tło znaków drastycznie zmieniło kolorystykę. Zdarzało się, że znak „Ustęp pierwszeństwa” swoją kolorystyką pokrywał się z koronami pożółkłych drzew w tle (Rys. 31).



Rysunek 31 Zdjęcia testowe wykonane jesiennej porą zwiększącą trudność w rozpoznawaniu znaków przez zmianę koloru tła [Źródło: opracowanie własne]

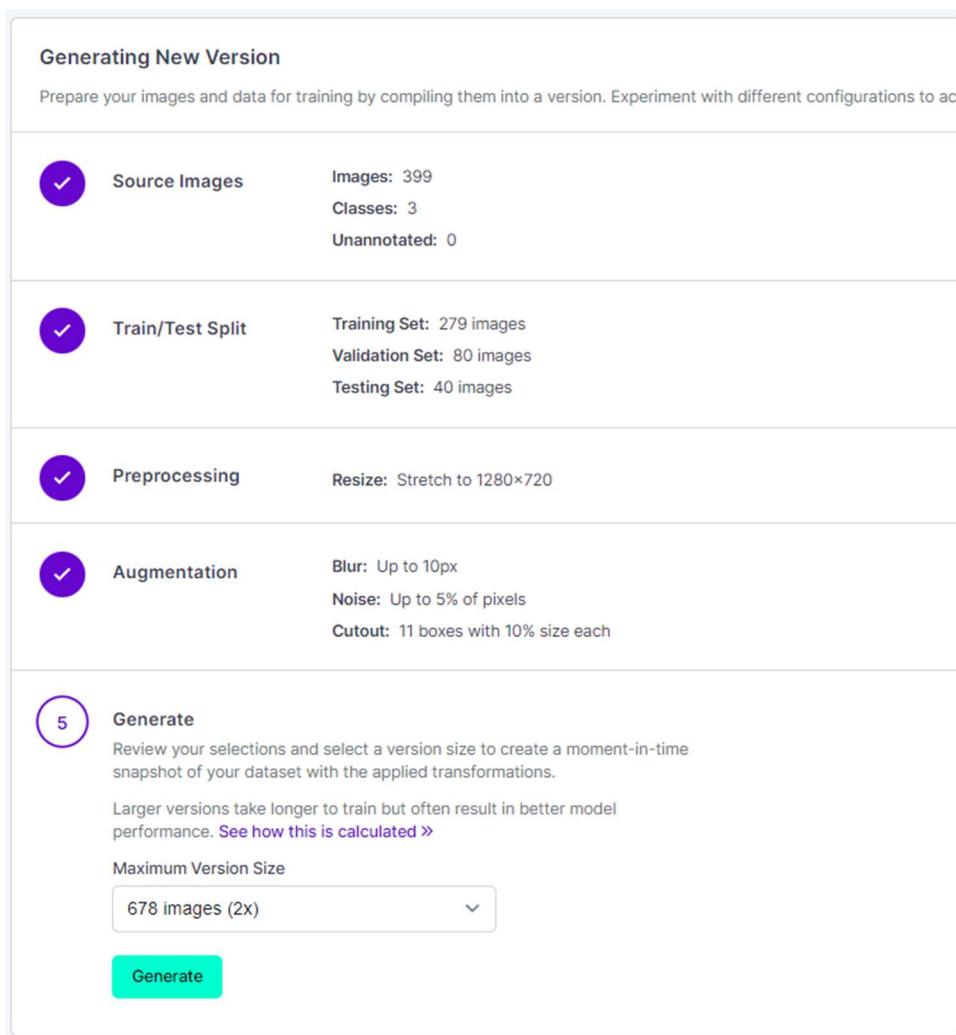
Kolejnym zróżnicowaniem było słabsze doświadczalnie znaków niż to na zdjęciach pozyskanych z Google Maps, ze względu na znaczne zachmurzenie. W rezultacie powstała baza danych zawierająca pięćdziesiąt zdjęć każdego znaku. Pierwszym krokiem przed przystąpieniem do porównania było zaetykietowanie ręczne danych w zbiorze testowym, czyli naniesienie wartości referencyjnych na każdy obraz, by móc ocenić zbieżność pracy modelu z oczekiwaniemi. Następnie utworzono program w języku Python, który iterując przez wszystkie obrazy obliczał wartości IoU i confidence, by zapisać te wartości do pliku [34].

7. TRENOWANIE MODELI

7.1. SERWIS TRENUJĄCY MODELE

W oparciu o przedstawioną dotychczas wiedzę należało podjąć decyzje dotyczące konfiguracji procesu uczenia modeli będących przedmiotem eksperymentu. Jak już wspominano uczenie modeli jest procesem wymagającym pod względem mocy obliczeniowej i zajmuje wiele czasu, dlatego powstały serwisy trenujące modele na dostarczonym im materiale w znacząco krótszym czasie, przy wykorzystaniu urządzeń wysokiej mocy. Celem przyspieszenia prac postanowiono skorzystać z takiej usługi. Praca z tym rozwiązań wymaga jedynie wgrania swojej bazy danych, następnie możliwe jest jej modyfikowanie, np. zmiana rozmiaru, jak również zastosowanie zabiegu „dogenerowania danych” (ang. *data augmentation*). Polega on na zwiększeniu ilości danych trenujących model poprzez przekształcenie obrazów istniejących. Odbywa się to przez tworzenie kopii w lustrzanym odbiciu, z zastosowaniem zaszumienia, poprzez przysłonięcie fragmentów zdjęć, czy np. rozmycie. Zabieg ten pozytywnie wpływa na jakość modelu i utrudnia przetrenowanie. Rozważanym serwisem trenującym stał się *Roboflow* [35]. Zgodnie z tym co powiedziano wyróżnia się on wyjątkową łatwością w obsłudze, pozwalającą na korzystanie z technologii trenowania modeli sieci neuronowych bez konieczności posiadania wyspecjalizowanej wiedzy w tej dziedzinie. Wystarczy dysponować odpowiednią co do złożoności projektu, ilością danych. Pierwszym krokiem podczas pracy z tym oprogramowaniem online jest wgranie zdjęć oraz ich zaetykietowanie poprzez tradycyjne obwiedzenie obiektu na zdjęciu ramką. Następnie należy ustalić stosunek zdjęć trenujących względem walidacyjnych i testowych. Tak wprowadzone zdjęcia można wedle potrzeby zmodyfikować pod względem rozmiaru czy np. zmienić ich ilość kanałów, przekształcając zdjęcia kolorowe do postaci zdjęć w odcieniach szarości. Szeroki wachlarz możliwości dotyczy wspomnianego zabiegu rozszerzania bazy danych poprzez modyfikację istniejących obrazów. Na tym etapie możliwy jest wybór o jaki rodzaj nałożonego efektu chcemy rozwinąć naszą bazę danych. Metod modyfikacji zastosowanych jednocześnie może być wiele, każda z nich posiada parametry konfiguracyjne takie, jak natężenie efektu. Tłumacząc - w przypadku chęci zastosowania metody powodującej zaszumienie obrazu, możliwe jest regulowanie intensywności tego zjawiska poprzez ustawienie jej procentowej wartości. Ostatnim krokiem jest wskazanie jaka ilość zdjęć ma zostać dogenerowana do pierwotnej bazy danych. Wszystkie kroki konfiguracyjne prezentuje zdjęcie poniżej (Rys. 32). Po przygotowaniu danych w opisany sposób, pozostaje uruchomić trenowanie modelu, o zakończeniu procesu uczenia dowiadujemy się otrzymując e-mail.

Po przetestowaniu usługi Roboflow otrzymano model wytrenowany na podstawie dostarczonych obrazów. Baza składała się z czterystu pięćdziesięciu zdjęć, podzielonych na partię trenującą, walidacyjną i testującą. Model osiągnął następujące wartości parametrów: mAP - 72.9%, precyza - 51.2%, czułość - 74.9%. Niestety to jedynie wartości jakie są prezentowane użytkownikowi, dlatego postanowiono zrezygnować z usługi Roboflow, by mieć wpływ na proces uczenia i móc go świadomie modyfikować. W niektórych przypadkach bezobsługowość systemu Roboflow może stanowić dużą zaletę, w tym przypadku udzielane informacje i możliwość ingerencji jest zbyt ograniczona.



Rysunek 32 Widok okna konfiguracyjnego bazy danych. Program daje możliwość wybrania proporcji danych trenujących, walidacyjnych i trenujących, zmianę rozmiaru i dodanie modyfikacji obrazów. Zastosowano rozmycie, zaszumienie oraz przysłonięcia obrazów czarnymi polami [Źródło: opracowanie własne]

7.2. WYKORZYSTANE TECHNOLOGIE

Ostatecznie zrezygnowano ze zlecania trenowania modelu na rzecz większej kontroli nad procesem uczenia, ponieważ przedstawiona metoda posiada wadę – brak możliwości ingerencji w konfigurację trenowania, a nawet uzyskania informacji o ustawieniach i architekturze sieci uczącej. W związku z tym powrócił problem wysokiego zapotrzebowania sprzętowego, jednak rozwiązaniem okazała się usługa dostarczona przez Google, mianowicie *Google Colaboratory*. Jest to środowisko online wyposażone w interpretator, umożliwiające pisanie i realizację kodu w języku Python. Największą zaletą Google Colab jest dostęp do wysokiej mocy kart graficznych wykonujących obliczenia zdalnie. Niestety korzystanie z tej usługi w wersji darmowej obarczone jest ograniczonym czasem dostępu do mocnego obliczeniowo GPU w ciągu doby. Skutkuje to rozłączeniem i przerwaniem trenowania po około czterech godzinach pracy w ciągu doby.

Wysokie wykorzystanie mocy obliczeniowej to ogromny problem trenowania modeli sieci neuronowych, ze względu na złożoną obliczeniowo propagację wsteczną. Na szczęście wykorzystanie zapasu mocy drastycznie spada podczas inferencji modelu, czyli jego utrzymania w pełnieniu zadań, do których został wytrenowany. Wówczas zaprzeganie GPU jest przerostem środków nad potrzebami, tańsze i zużywające mniej energii CPU jest wystarczające do tego, by optymalnie wykorzystać model do predykcji. W przypadku trenowania modeli z Google Colab wykorzystywana była karta graficzna *PNY Tesla K80 24GB*.

Jak wspomniano wcześniej, zrezygnowano w projekcie z pierwotnego pomysłu usprawnienia uczenia sieci przez wykorzystanie serwisu Roboflow, celem większej kontroli nad parametrami procesu. Taką możliwość dostarcza backbone o nazwie *Darknet*, Zawiera on architektury sieci YOLO w różnych wersjach, natomiast w tym projekcie oba modele zostały ukształtowane przez architekturę YOLOv3 oraz dla porównania powstały dwa modele wg. kolejnej wersji – YOLOv4. Omawiany Darknet daje bezpośredni dostęp do konfiguracyjnego pliku tekstowego, w którym w łatwy sposób można dokonać zmian dopasowanych do naszych potrzeb.

7.3. PROCEDURA TRENOWANIA

W tym podrozdziale opisany zostanie szczegółowo proces trenowania modeli YOLOv3 oraz YOLOv4, który należało rozpoczęć od zalogowania się do aplikacji Google Colaboratory (w skrócie *Colab*). Tam użytkownik ma możliwość tworzenia fragmentów kodu, by następnie móc je uruchamiać w preferowanej kolejności. Na koniec pracy nad tak stworzonym notatnikiem możliwy jest jego zapis na dysku Google. W pierwszej kolejności należy połączyć się z serwerem, następnie wybrać środowisko wykonawcze GPU. Kolejnym krokiem jest sprawdzenie popranego działania i połączenia z GPU, dokonujemy tego prostą komendą:

```
!nvidia-smi
```

W efekcie uzyskujemy wydruk z informacjami o obsługującej nas karcie graficznej, co zaprezentowano poniżej (Rys. 33).

```
# Check if NVIDIA GPU is enabled
!nvidia-smi

Tue Nov  9 16:40:49 2021
+-----+
| NVIDIA-SMI 495.44      Driver Version: 460.32.03      CUDA Version: 11.2 |
| Persistence-M | Bus-Id     Disp.A    | Volatile Uncorr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap | Memory-Usage | GPU-Util  Compute M. |
|          |          |          |          |          |          |          |
|-----|-----|-----|-----|-----|-----|-----|-----|
|     0  Tesla K80      Off  00000000:00:04.0 Off   0          |
| N/A   35C   P8    28W / 149W  0MiB / 11441MiB   0%      Default |
|          |          |          |          |          |          |          |
+-----+-----+-----+-----+-----+-----+-----+-----+
Processes:
+-----+
| GPU  GI  CI      PID  Type      Process name          GPU Memory |
| ID   ID                  |          |          | Usage          |
+-----+
| No running processes found
+-----+
```

Rysunek 33 Widok okna wydruku z informacjami o używanej karcie graficznej [Źródło: opracowanie własne]

Wadą Google Colab, oprócz krótkich sesji z dostępem do GPU i braku możliwości wykupienia wersji Pro w naszym kraju, jest to, że podczas każdej sesji chcąc korzystać z plików znajdujących się na dysku Google (tak jak w naszym przypadku), należy importować dostęp do niego i autoryzować dostęp dodatkowym logowaniem. Zanim połączono bieżącą sesję z dyskiem Google utworzono na nim folder zawierający dane trenujące (obrazy wraz z etykietami) spakowane do postaci ZIP i plik zawierający nazwy klas do obsługi etykiet. Jak już wspomniano wykorzystano w eksperymencie backbone Darknet w wersji zawierającej pięćdziesiąt trzy warstwy splotowe, a więc *Darknet-53*. Do jego prawidłowej pracy wymagany jest udział GPU i związanej z jego obsługą biblioteki CUDNN, która odpowiada za akcelerację działania kart graficznych NVIDIA, oraz biblioteka OpenCV. Należy zainstalować wszystkie wymienione komponenty. Następnym krokiem jest dostosowanie procesu trenowania do potrzeb projektu. Darknet umożliwia łatwą modyfikację poprzez edytowanie pliku konfiguracyjnego. Poniżej wymieniono kluczowe parametry wraz z ich opisem ich działania i strojenia:

- **Batch = 64**, jest to parametr mówiący o ilości przykładów analizowanych przez sieć w jednej iteracji. Wartość jest dostosowana do mocy obliczeniowej, dzięki zastosowaniu zewnętrznego sprzętu poprzez alkilację Google Colab, który charakteryzuje się wysoką wydajnością karty graficznej oraz dużą pamięcią RAM, możliwe przetworzenie

podczas jednej iteracji znacznie większej ilości obrazów niż z wykorzystaniem domowego PC.

- **Subdivision = 16**, określa na ile części ma być podzielona wsadowa grupa (batch), jeżeli wartość wynosi 16, to iteracja skończy się po równoległym przeanalizowaniu szesnastu podgrup, podnosi to wydajność procesu, lecz podobnie jak parametr batch, również subdivision jest uzależnione od wydajności sprzętu. Używając GPU wysokiej mocy jesteśmy w stanie realizować równolegle większą ilość procesów, dla porównania – CPU nie daje możliwości przeprowadzania tak skomplikowanych operacji równolegle, w ilości przekraczającej ilość rdzeni, dlatego gorzej sprawdza się w procesie trenowania,
- **Max_baches = 6000**, jest to parametr, który rośnie liniowo w zależności od ilości klas – zgodnie z instrukcją w dokumentacji Darknet stanowi iloczyn ilości klas i liczby 2000, lecz nie może osiągać wartości mniejszej niż ilość obrazów trenujących i nie może być mniejszy od 6000. W przypadku trzech klas, jakimi są trzy różne znaki, wartość tę należy ustawić na 6000, a jej interpretacją jest ilość iteracji, po których model zostaje uznany za wytrenowany,
- **Filters** = ilość filtrów powiązana z ilością klas wg. zależności:

$$Filters = (ilosc\ klas + 5) \cdot 3,$$

Co wynika z faktu, że wersja YOLOv3 może utworzyć trzy ramki ograniczające dla każdej komórki mapy cech. Wartość „5” wynika z tego, że każda ramka posiada współrzędną x oraz y pierwszego narożnika ramki, jej wysokość i szerokość oraz parametr *objectness score*, który ocenia w skali 0-1 jak dokładnie detektor ustalił lokalizację i klasę obiektu [36] [37],

- **Class = 3**, ilość klas,
- **Momentum = 0,9**, hiperparametr związany z omawianym optymalizatorem, którego celem jest lepsze ukierunkowanie zbiegania do minimum. Mówiąc o tym jaki wpływ na obecny wynik ma poprzedni. Przy tak wysokiej wartości z przedziału <0,1>, wyhamowanie zbiegania do minimum będzie niewielkie.
- Architektura wykorzystuje Leaky ReLU jako funkcję aktywacyjną oraz korzysta z normalizacji wsadowej (ang. *Batch Normalization*).

Przed pierwszym uruchomieniem trenowania wykorzystano podstawianie wstępnych wag do sieci. Jest to zabieg przyspieszający trenowanie, ponieważ model trafia na właściwe tory uczenia się bez wstępnego błądzenia. Po czym następującą komendą uruchamiamy trening:

```
!./darknet detector train data/obj.data cfg/yolov4_training.cfg darknet53.conv.74
```

Z komendy wynika, że trenowany model będzie detektorem, wskazana jest ścieżka do danych trenujących, następnie do pliku konfiguracyjnego, a na koniec odwołanie do wag wstępnego trenowania od których właściwy trening ma się rozpocząć. Chcąc zbadać przebieg funkcji Loss w kolejnych iteracjach, do powyższej komendy dodać odpowiedni modyfikator:

```
-ext_output </content/gdrive/MyDrive/yolov4/train.txt> /content/gdrive/MyDrive/yolov4/results.txt
```

Informacje o dostępnych modyfikatorach i ich działaniu są klarownie przedstawione w dokumentacji w repozytorium Darknet. Powyższa komenda tworzy plik tekstowy we wskazanej lokalizacji, po czym w ciągu kolejnych iteracji umieszcza w nim informacje m.in. o średniej wartości Loss dla aktualnej iteracji. Daje to możliwość oceny szybkości uczenia na podstawie utworzonego wykresu.

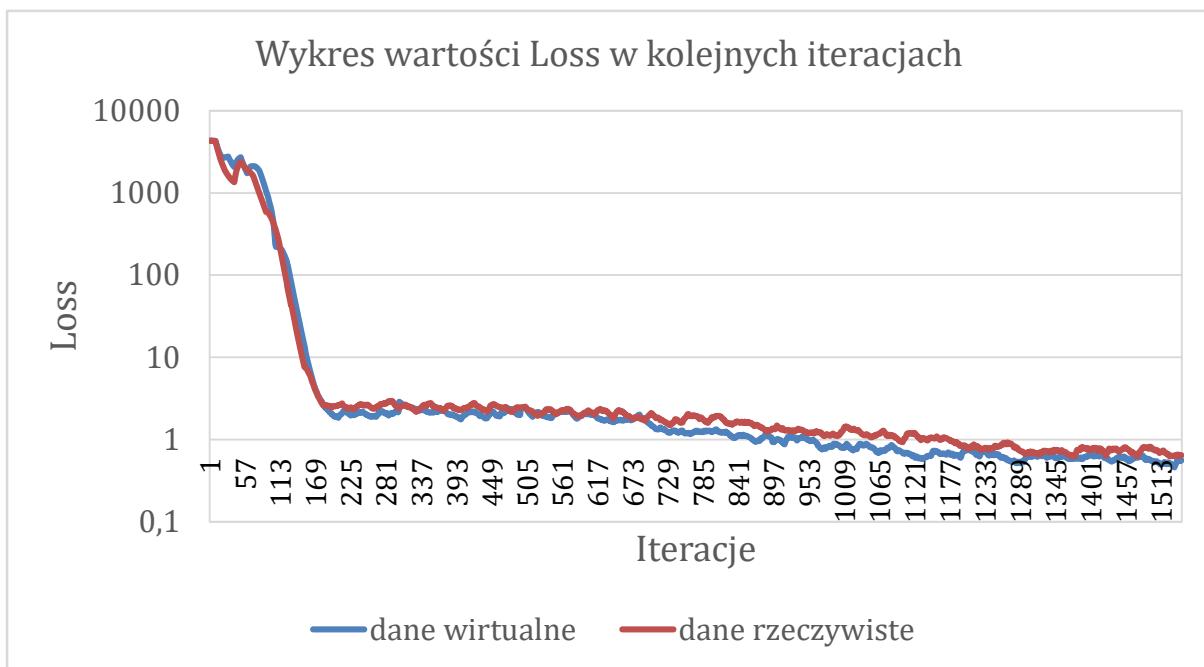
Podczas trenowania modeli z użyciem Darknet, program wyświetla cenne informacje na temat przebiegu treningu. Informacje te stanowią podsumowanie poszczególnych iteracji, a także efektu uczenia podgrup. Pozwala to ocenić, czy trening jest właściwie ukierunkowany i daje zadowalające efekty formowania modelu. Takimi współczynnikami wyświetlonymi na ekranie są między innymi:

- Numer iteracji, który został wykonany,
- Wskaźnik „Loss”, czyli wartość określająca błąd między predykcją a wartością rzeczywistą. Celem jest uzyskanie wartości bliskiej zeru, która oznacza niemal idealne dopasowanie wag modelu i wyrazu wolnego. Zakłada się, że Loss na poziomie 0.06 stanowi wystarczającą wartość do tego, by przerwać uczenie.
- Czas poświęcony na trenowanie modelu na podstawie ostatniej grupy,
- Średnia wartość IoU dla całej grup oraz czynnik IoU dla poszczególnych klas.
- Wartość „Class” określa stosunek prawidłowo sklasyfikowanych obiektów i powinien zbliżać się do wartości 1.

Modele uczone według architektury YOLOv3 zostały wytrenowane do osiągnięcia wartości Loss zbliżonej do 0,06. Pozwala to na przeprowadzenie rzetelnego porównania modeli uczonych na danych wirtualnych i rzeczywistych. Model trenowane w oparciu o nowszą wersję – YOLOv4, osiągnęły wartość funkcji Loss równą 0,6, ich trening zostanie oceniona w kolejnych rozdziałach.

8. OCENA MODELI ORAZ PORÓWNANIE WYNIKÓW ICH UCZENIA NA DANYCH ZE ŚRODOWISKA RZECZYWISTEGO I WIRTUALNEGO

Postępując zgodnie z opisem z poprzedniego rozdziału w sumie wytrenowano cztery modele – dwa na bazie architektury YOLOv3 w oparciu o dane rzeczywiste i wirtualne, oraz dwa modele oparte na architekturze YOLOv4. Porównanie modeli różniących się pochodzeniem zdjęć wykonano wyłącznie na modelach YOLOv3, ze względu na osiągniętą podczas treningu niską wartość funkcji straty (Loss). Modele YOLOv4 posłużą do przedstawienia jak wygląda przebieg zmian wartości Loss w kolejnych iteracjach. Testowanie wytrenowanych modeli odbyło się przy pomocy zdjęć wykonanych w terenie. Na każdy znak przypadało pięćdziesiąt obrazów w terenie zabudowanym jak i poza nim. Zbiór zdjęć charakteryzuje różnorodność kolorystyki tła, rozmiaru znaków na obrazie oraz kąta pod jakim się one pojawiają. Miarami jakie przyjęto do porównania modeli jest współczynnik IoU, czułość, precyzyja oraz *confidence* – ilościowe ujęcie pewności modelu co do swojej predykcji. W przypadku modelu YOLOv4 proces uczenia został rozszerzony o funkcję zapisującą wartość funkcji straty po każdej iteracji, co dało możliwość przyjrzenia się dynamiczemu treningu podczas jego trwania. Na wykresie poniżej (Rys. 34) zaprezentowano zestawienie zmieniającej się funkcji straty w ciągu trwania trenowania.



Rysunek 34 Przebieg treningu modeli z udziałem architektury Darknet YOLOv4 [Źródło: opracowanie własne]

Wykres prezentuje prawidłowy postęp w trenowaniu modeli. Widać, że po gwałtownym spadku wartości loss wagi zaczęły ustalać się w znacznie spowolnionym tempie, jednak występuje zauważalny trend malejący. Trening modelu opartego na architekturze YOLOv4 zatrzymano po około 1550 iteracjach co doprowadziło do osiągnięcia wartości zbliżonej do 0,6.

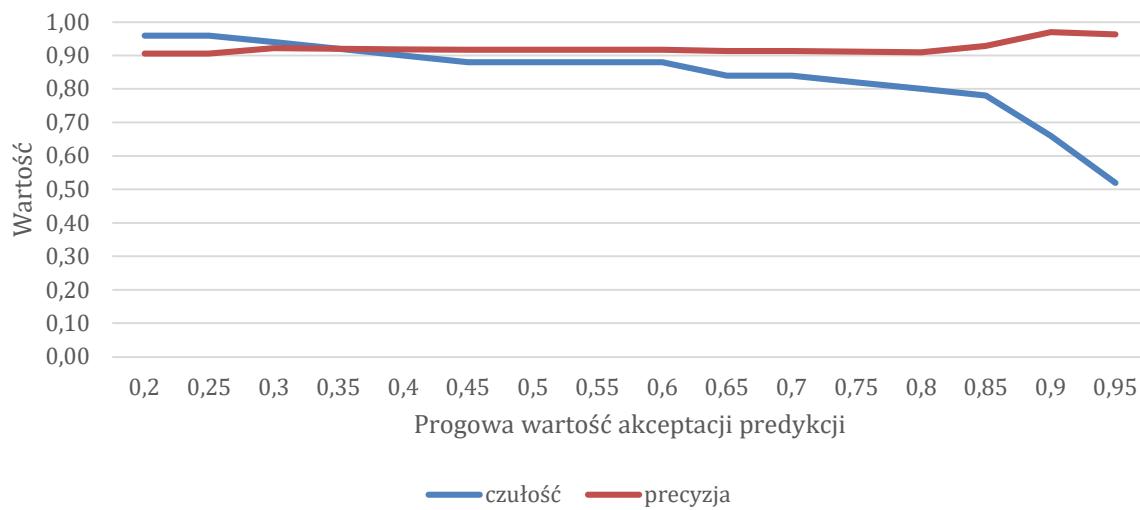
Efekt pracy modeli nie jest odzwierciedlony jedynie w liczbach, lecz jak przystało na modele pełniące rolę detektorów, dają również wynik w postaci wizualnej, a mianowicie wskazują prostokątną obwiednią obszar obrazu, w którym przewidują, że znajduje się obiekt zaliczony przez model do konkretnej klasy. Na testowej fotografii poniżej zaznaczono kolorem zielonym ramkę referencyjną, natomiast niebieska ramka jest wskazaniem modelu. Jak widać współczynnik IoU w tym przypadku wyniósł 0,61 (Rys. 35).



Rysunek 35 Fotografia z naniesioną predykcją modelu w postaci granatowej obwiedni oraz ramka referencyjna w kolorze zielonym [Źródło: opracowanie własne]

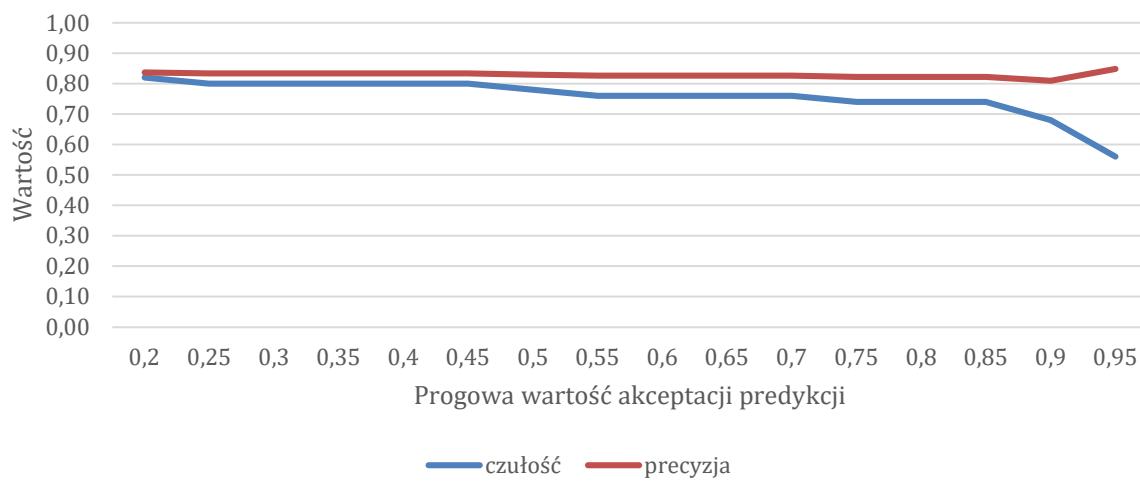
Dane zebrane podczas testowania modelu YOLOv3 posłużyły do obliczenia i porównania precyzji i czułości modeli. Do zbioru obiektów zaklasyfikowanych jako FP (fałszywie pozytywne) trafiły predykcje, które pomyliły klasy lub wskazały obiekt w miejscu, w którym się on nie znajdował. Natomiast do zbioru FN (fałszywie negatywne) trafiły predykcje, które powstrzymały się od zaznaczenia obiektu na obrazie. Progiem pewności modelu, przy którym zaznaczał obiekt na obrazie (confidence), była domyślnie wartość 0,2 – poniżej tej wartości model ignorował obiekt. Wykresy poniżej obrazują, jak zmienia się wartość czułości i precyzji modeli przy zmieniającej się wartości progowej.

Poziom precyzji i czułości modelu w zależności od przyjętego progu wartości confidence dla znaku "Droga z pierwszeństwem". Model danych rzeczywistych

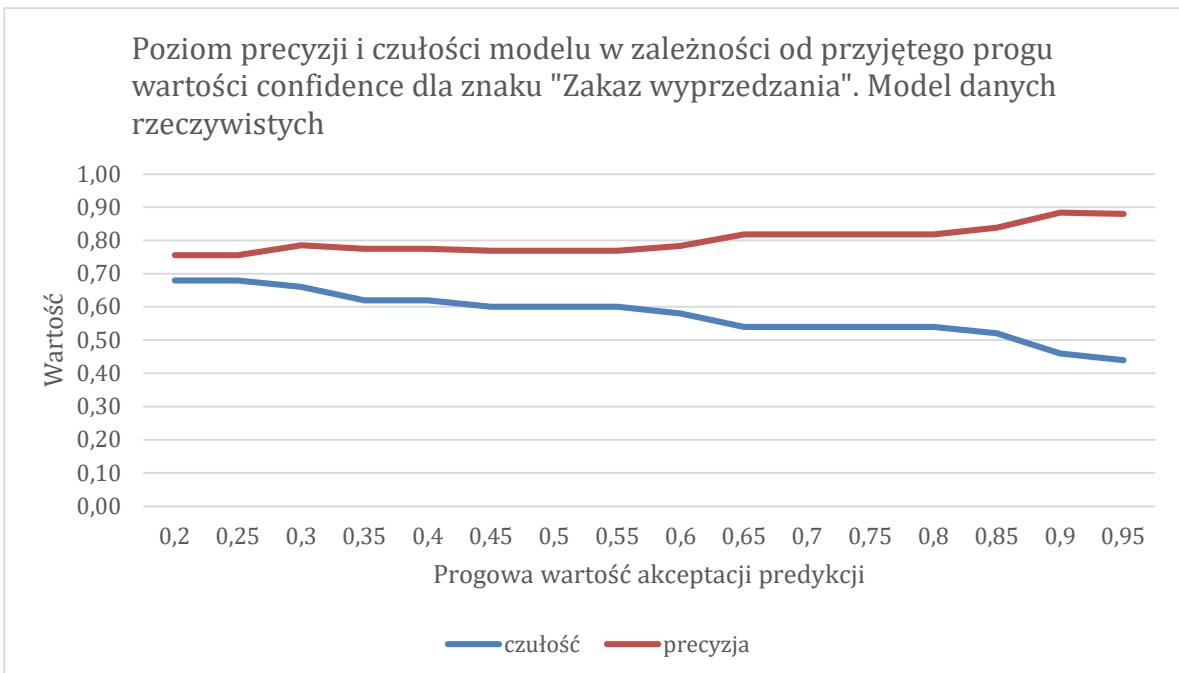


Rysunek 36 Zależność precyzji i czułości modelu od progu wartości confidence - dane trenujące: rzeczywiste, znak: Droga z pierwszeństwem [Źródło: opracowanie własne]

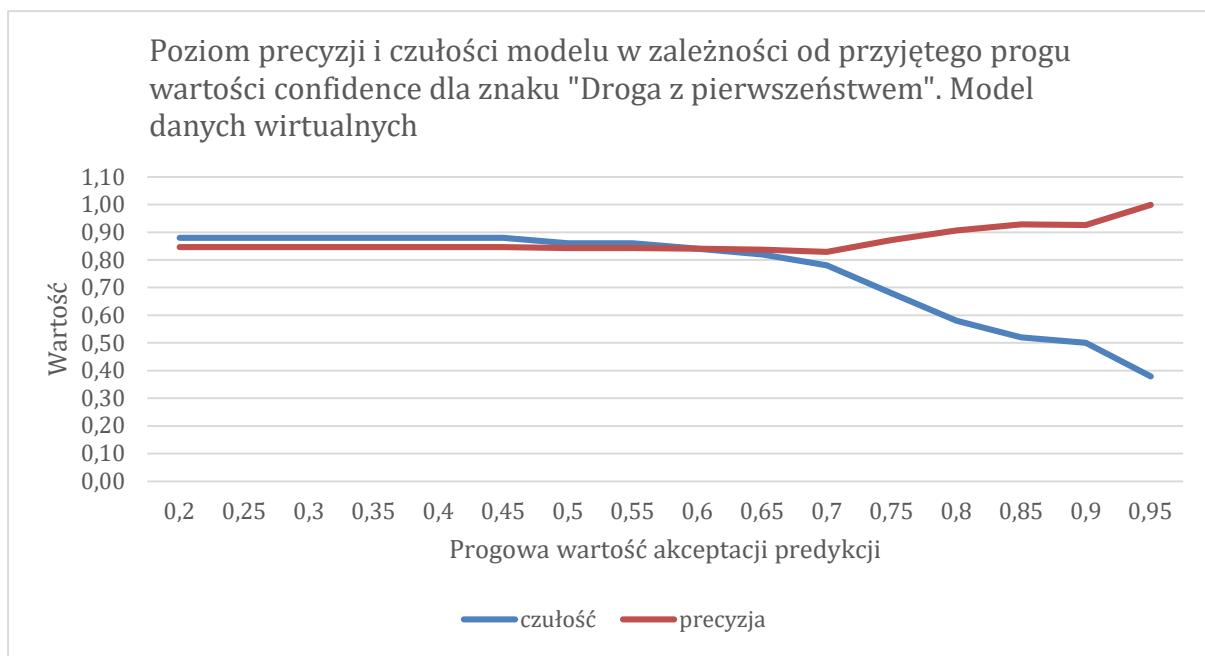
Poziom precyzji i czułości modelu w zależności od przyjętego progu wartości confidence dla znaku "Ustęp pierwszeństwa". Model danych rzeczywistych



Rysunek 37 Zależność precyzji i czułości modelu od progu wartości confidence - dane trenujące: rzeczywiste, znak: Ustęp pierwszeństwa [Źródło: opracowanie własne]

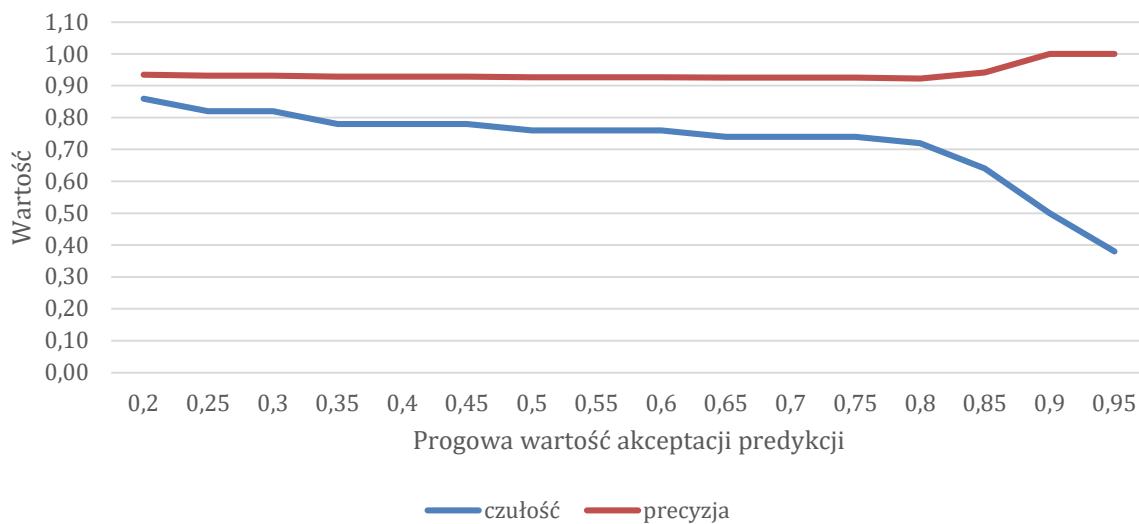


Rysunek 38 Zależność precyzji i czułości modelu od progu wartości confidence - dane trenujące: rzeczywiste, znak: Zakaz wyprzedzania [Źródło: opracowanie własne]



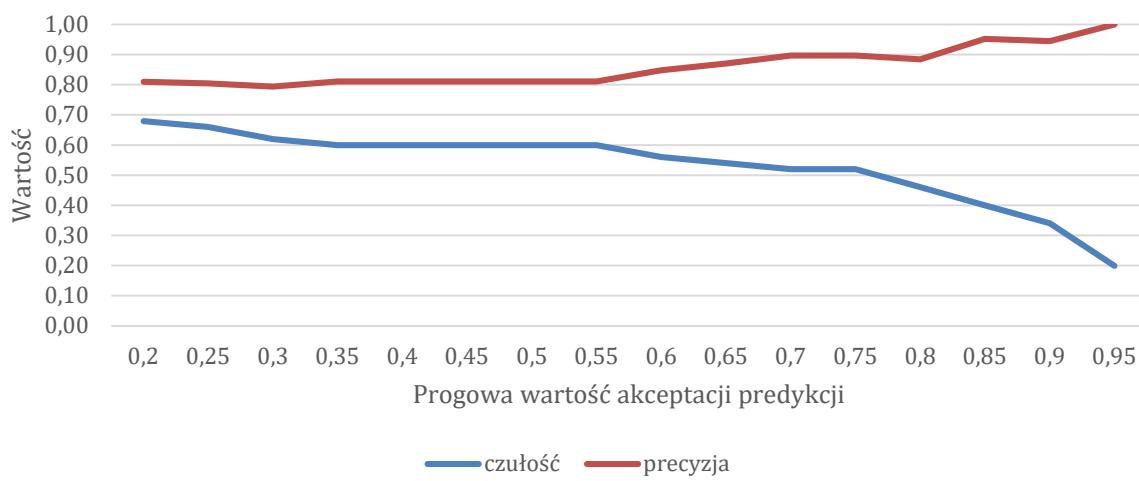
Rysunek 39 Zależność precyzji i czułości modelu od progu wartości confidence - dane trenujące: wirtualne, znak: Droga z pierwszeństwem [Źródło: opracowanie własne]

Poziom precyzji i czułości modelu w zależności od przyjętego progu wartości confidence dla znaku "Ustęp pierwszeństwa". Model danych wirtualnych



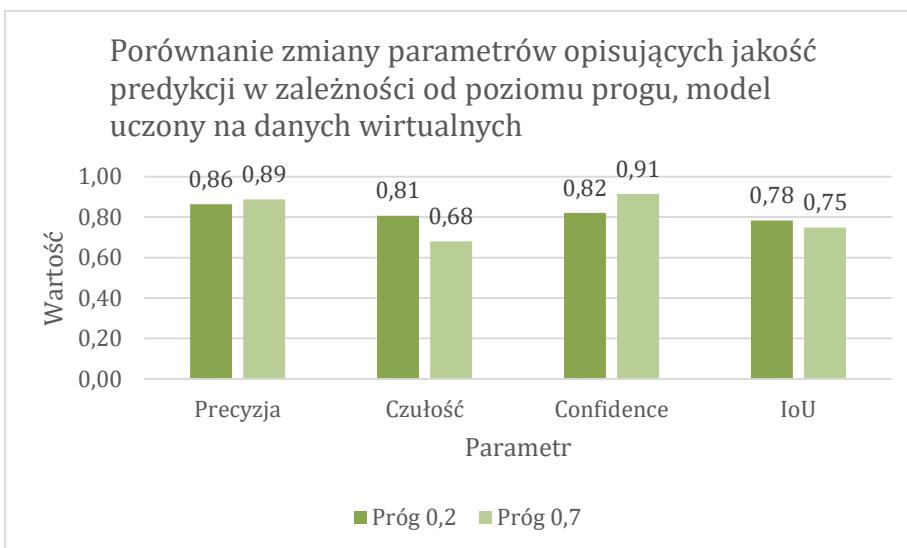
Rysunek 40 Zależność precyzji i czułości modelu od progu wartości confidence - dane trenujące: wirtualne, znak: Ustęp pierwszeństwa [Źródło: opracowanie własne]

Poziom precyzji i czułości modelu w zależności od przyjętego progu wartości confidence dla znaku "Zakaz wyprzedzania". Model danych wirtualnych

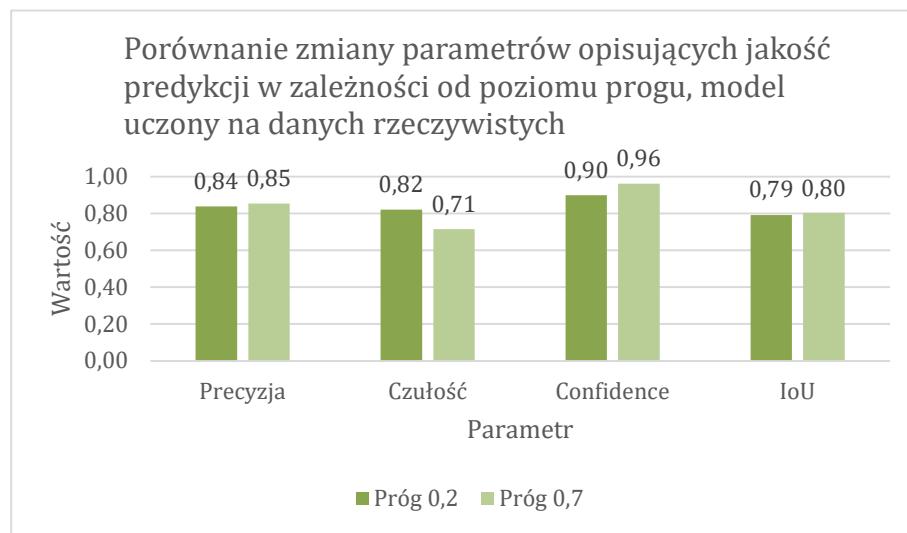


Rysunek 41 Zależność precyzji i czułości modelu od progu wartości confidence - dane trenujące: wirtualne, znak: Zakaz wyprzedzania [Źródło: opracowanie własne]

Analiza powyższych wykresów prowadzi do spodziewanych obserwacji: wraz ze wzrostem wartości progowej akceptacji predykcji, precyzja rośnie, natomiast czułość modelu maleje. Te dwa wykresy często przecinają się i precyzja początkowo mniejsza od czułości, przy wyższej wartości progowej, znacznie ją przewyższa. Pojawia się pytanie jaki próg z punktu widzenia eksperymentu jest praktyczny, tj. jaka wartość progu zapewnia dokładność predykcji i szczelność systemu. Wykresy poniżej przedstawiają porównanie parametrów działania modelu dla progu akceptacji na poziomie 0,2 oraz 0,7. Pierwsza wartość jest poziomem domyślnym, natomiast wartość 0,7 została ustalona w drodze wnioskowania na podstawie powyższych wykresów i stanowi przybliżoną wartość, przy której następuje wyraźne załamanie przebiegów czułości i precyzji (Rys. 36-41).



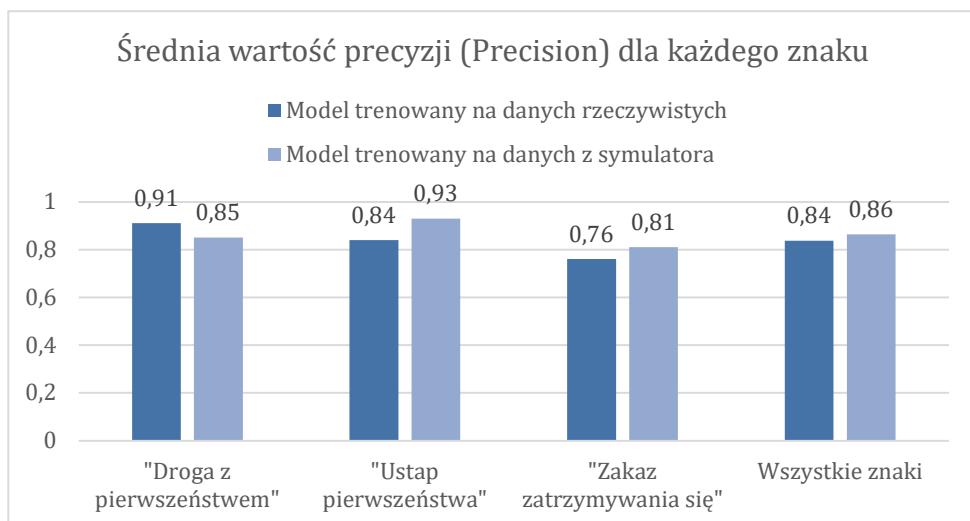
Rysunek 42 Porównanie zmiany parametrów opisujących jakość predykcji w zależności od poziomu progu, model uczony na danych wirtualnych [Źródło: opracowanie własne]



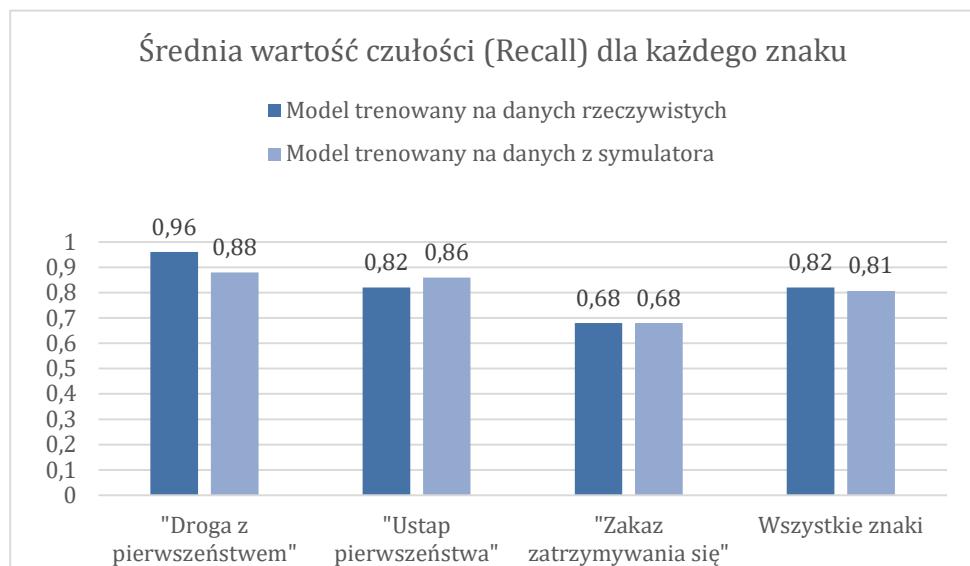
Rysunek 43 Porównanie zmiany parametrów opisujących jakość predykcji w zależności od poziomu progu, model uczony na danych rzeczywistych [Źródło: opracowanie własne]

Zgodnie z powyższymi wykresami zwiększenie wartości progowej zapewnia podniesienie precyzji predykcji, jednak kosztem obniżenia czułości. Rys. 42 i 43 dodatkowo informują, iż istnieje brak proporcji między tymi zmianami i precyza rośnie niewiele wobec znaczącego spadku czułości. Biorąc pod uwagę, iż w projekcie zdecydowano się na rozpoznawanie znaków drogowych, uznano, że priorytetem jest ich odnajdowanie na obrazach i nieomijanie, dlatego korzystniej będzie pozostać przy niższym progu. Sprawi to, że precyza będzie o średnio 0,02 mniejsza od tej przy progu 0,7, jednak zapobiegnie to również spadkowi czułości o średnio 0,12. Wpływ zmiany progu na IoU jest pomijalny.

Na podstawie przyjętego progu akceptacji predykcji wyliczono precyzję i czułość poszczególnych modeli, czego wyniki przedstawiają się następująco:

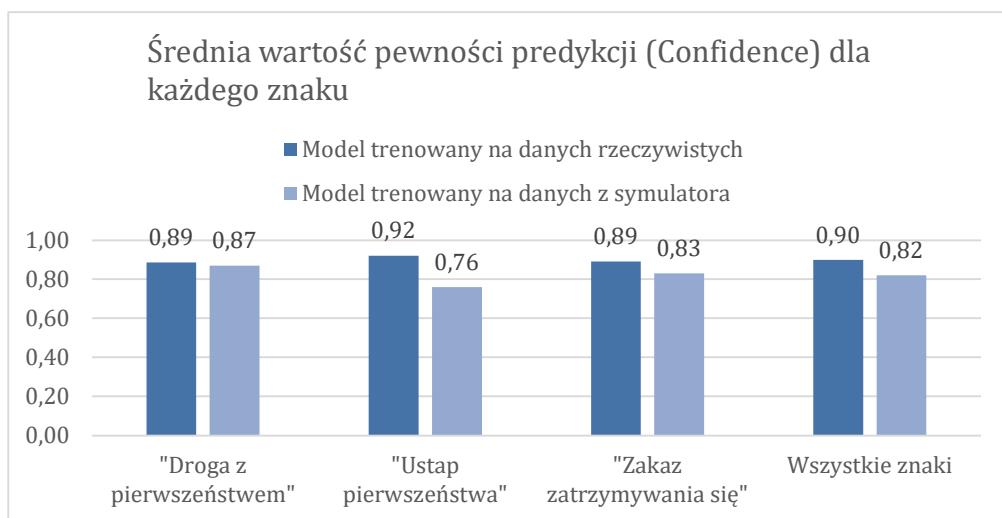


Rysunek 44 Średnia wartość precyzji dla każdego znaku, wybranego do eksperymentu [Źródło: opracowanie własne]



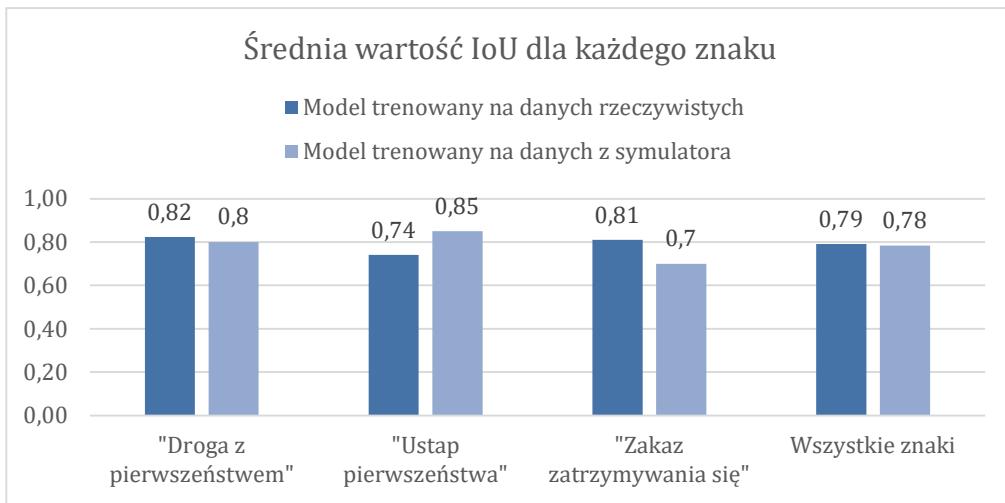
Rysunek 45 Średnia wartość czułości dla każdego znaku, wybranego do eksperymentu [Źródło: opracowanie własne]

Na powyższych wykresach można zauważać różnice wartości precyzji dla modeli, sięgające nawet 0,9, jednak wynik nie zawsze jest na korzyść jednego modelu, lecz każdy model lepiej radził sobie z rozpoznawaniem innych znaków. Średnia wartość precyzji detekcji dla wszystkich znaków jest zbliżona dla obu modeli, a różnica wynosi 0,02 (Rys. 44). W przypadku *czułości*, wartości dla poszczególnych znaków są zdecydowanie bardziej zbliżone do siebie w przypadku obu modeli, natomiast to co zwraca uwagę, to wartości czułości dla znaku „zakaz zatrzymywania się”, w tym przypadku detekcja obiektu była dla obu modeli na odbiegającym niższym poziomie. Również dla czułości, jej średnia wartość podczas rozpoznawania wszystkich znaków jest podobna dla obu modeli, a różnica wynosi 0,1 (Rys. 45).



Rysunek 46 Średnia wartość confidence dla każdego znaku, wybranego do eksperymentu [Źródło: opracowanie własne]

Z wykresu powyżej wynika wyraźnie, że istnieje niewielka różnica w pewności co do detekcji obiektów na obrazach, przeprowadzanej przez modele uczone na danych rzeczywistych i trenowanych przy pomocy obrazów pochodzących z gry. We wszystkich przypadkach znaków model uczony na symulacji jest średnio o 0,08 mniej pewny swojej decyzji. Można ten wynik powiązać bezpośrednio z faktem, iż tło znaku w grze jest mimo wszystko mniej złożone niż to na obrazach wykonanych między innymi w miejskiej scenerii, gdzie oprócz znaków drogowych i budynków widnieją również kolorowe banery i bilbordy. Ma to również wpływ na obniżenie parametru recall (*czułość*) modelu trenowanego na symulatorze (Rys. 46).



Rysunek 47 Średnia wartość IoU dla każdego znaku, wybranego do eksperymentu [Źródło: opracowanie własne]

Opisane dotychczas wskaźniki dotyczą oceny zdolności modelu do klasyfikowania obiektów, jednak detekcja wymaga również oceny poprawności wskazania obiektu na obrazie. W tym celu dla każdego obrazu policzono wartość IoU (*intersection over union*). Dla przypomnienia jest to parametr mówiący jak dokładnie dopasowana jest ramka detektora względem referencyjnej. Ramkę referencyjną nałożono analogicznie jak w przypadku etykietowania danych do w partii treningowej – jej wymiary i położenie zapisane w konwencji YOLO znajdowały się w pliku tekstowym dołączonym do każdego z obrazów. Wykres powyżej obrazuje wyniki dopasowania ramek dla poszczególnych znaków przez oba modele. Wyniki w obrębie testowanych znaków są zróżnicowane jednak wygląda na to, że średnio oba modele poradziły sobie z zadaniem na podobnym poziomie, ponieważ różnica wynosi jedynie 0,01 (Rys. 47).

9. PODSUMOWANIE I WNIOSKI

W pracy został poruszony problem trudnego dostępu do danych trenujących w postaci obrazów. Postawiono za zadanie przyjrzenie się i ocenę gier komputerowych lub innego środowiska wirtualnego jako potencjalnej alternatywy do zdjęć wykonywanych obiektom rzeczywistym. W efekcie wytrenowano modele sieci neuronowej w architekturze YOLOv3 do porównania jakości danych pochodzących z gry komputerowej Eurotruck Simulator 2 z danymi rzeczywistymi. Wytrenowano również model w wyższej generacji architekturze - YOLOv4, celem zapisania przebiegu treningu i oceny czy oba źródła zdjęć zapewniają prawidłowy przebieg procesu uczenia. Do tej oceny posłużył przebieg funkcji kosztu w kolejnych iteracjach. Analiza wykresu funkcji kosztu ujawniła, że proces przebiega prawidłowo i zgodnie z przewidywaniami. Kluczową część eksperimentu stanowią wyniki uzyskane z porównania dwóch modeli, o różnym pochodzeniu danych uczących. Po porównaniu jakości predykcji przez oba modele, dokonanym na zdjęciach rzeczywistych wykonanych własnoręcznie w terenie, nasuwają się wnioski potwierdzające możliwość wykorzystania zdjęć pochodzących ze środowiska wirtualnego celem stworzenia modelu, którego inferencja przebiegałaby w otaczającym nas świecie, świadczą o tym zbliżone wartości wybranych wskaźników porównawczych, takich jak pełność, precyzja czy współczynnik IoU. Okazuje się jednak, że świat wirtualny przez swoje uproszczenie i brak tak urozmaiconego tła dla znaków drogowych, jakie istnieje w rzeczywistym mieście, może wpływać na te niewielkie różnice w wynikach pracy modeli. Najbardziej rzuającymi się w oczy różnicami jest właśnie środowisko miejsce – w grze zauważalnie mniej rozbudowane o bilbordy, reklamy, podświetlane banery, jak również o mniejszej ilości samochodów. Wynik eksperimentu jest zachęcający do dalszego poszukiwania alternatywnych metod pozyskiwania danych trenujących niskim kosztem i nakładem czasu. Kierunkiem postępowania rozpoczętego przez eksperiment opisany w tej pracy może być stworzenie systemu rozpoznającego i alarmującego o nietypowych zrachowaniach jak uliczne bójki, napady na lokale i banki czy zasłabnienia ludzi w miejscach publicznych. W tym celu również można by wykorzystać symulację do tworzenia obszernej bazy danych, stanowiącej na przykład zapisy widoków z gry komputerowej. Najważniejszą i podkreślaną cechą takiej bazy jest niski koszt i łatwość jej uwożenia.

Zabiegami mogącymi poprawić wiarygodność eksperimentu jest zebranie liczniejszej bazy danych niż ta wykorzystana w opisywanym projekcie. Porównanie modeli na większej ilości znaków drogowych również mogłoby wpłynąć na wynik końcowy. Jeżeli model nie byłby wykorzystywany w dynamicznej inferencji, warto byłoby rozważyć inną architekturą, np. ResNet, której średnia precyzja w zadaniach detekcji jest wyższa, jednak model działa powolnie. Ponieważ znaki drogowe są obiektami łatwymi do rozpoznania, ze względu na charakterystykę pełnionego przez nie zadania, ewentualna kontynuacja badań w tym kierunku mogłaby obejmować rozpoznawanie obiektów bardziej trudnych do dostrzeżenia lub np. zasugerowanych wcześniej niebezpiecznych zajść ulicznych.

10. BIBLIOGRAFIA

- [1] [Online]. Available: <https://gamerant.com/best-looking-pc-games-ranked/>. [Accessed 05 11 21].
- [2] W. Duch, Biocybernetyka i inżynieria biomedyczna 2000, vol. 6, Warszawa: Akademicka Oficyna Wydawnicza EXIT, 2000.
- [3] A. Geron, Uczenie maszynowe z użyciem Scikit-Learn i Tensorflow, 2 ed., Helion, 2020.
- [4] S. Russel and P. Norvig, Artificial Intelligence A Modern Approach, Pearson, 2010.
- [5] A. Venkatesh, "Object tracking in games using convolutional neural networks," 2018.
- [6] Y. Haung, C. Lv and D. Dong, "Obtain dataset for Self-driving perception from Video Games automatically," in *ICRMS*, 2018.
- [7] B. Wu, "SqueezeSeg: Convolutional Neural Nets withwith recurrent CRF for Real-Time Road-Object Segmentation from 3D LiDAR Point Cloud," in *ICRA*, 2018.
- [8] [Online]. Available: <https://papers.readthedocs.io/en/latest/imageseg/squeezeseg/>. [Accessed 23 11 21].
- [9] S. Weidman, Uczenie głębokie od zera, Helion, 2020.
- [10] D. Esposito and F. Esposito, Wprowadzenie do uczenia maszynowego według Esposito, APN Promise, 2020.
- [11] R. Socher, M. Ganjoo, C. D. Manning and A. Y. Ng, "Zero-Shot Learning Through Cross-Modal Transfer".
- [12] J. Grus, Data science od podstaw, Helion, 2018.
- [13] A. Bifet, Mechanie learning for data streams, The MIT Press, 2018.
- [14] [Online]. Available: <https://towardsdatascience.com/the-differences-between-artificial-and-biological-neural-networks-a8b46db828b7>. [Accessed 11 10 21].
- [15] G. Bancorso, Algorytmy uczenia maszynowego, Helion, 2019.
- [16] S. Osowski, Sieci Neuronowe, Oficyna Wydawnicza Politechniki Warszawskiej, 1994.
- [17] J.-S. Jang, C.-T. Sun and E. Mizutani, Neuro-Fuzzy and Soft Computing, Prentice-Hall, Inc., 1997.

- [18] [Online]. Available: <https://www.aitude.com/comparison-of-sigmoid-tanh-and-relu-activation-functions/>. [Accessed 12 09 21].
- [19] [Online]. Available: <https://towardsdatascience.com/7-popular-activation-functions-you-should-know-in-deep-learning-and-how-to-use-them-with-keras-and-27b4d838dfe6>. [Accessed 12 11 21].
- [20] [Online]. Available: <https://www.youtube.com/watch?v=wrEcHhoJxjM>. [Accessed 12 12 21].
- [21] L. Moroney and A. Ng, Sztuczna inteligencja i uczenie maszynowe dla programistów, Helion, 2021.
- [22] [Online]. Available: <https://www.youtube.com/watch?v=DtEq44FTPM4>. [Accessed 28 11 21].
- [23] [Online]. Available: https://www.researchgate.net/figure/Pooling-layer-operation-oproaches-1-Pooling-layers-For-the-function-of-decreasing-the_fig4_340812216. [Accessed 02 12 21].
- [24] A. B. Amjoud and M. Amrouch, "Convolutional Neural Networks Backbones".
- [25] [Online]. Available: <https://ai-pool.com/a/s/yolov3-and-yolov4-in-object-detection>. [Accessed 01 11 21].
- [26] [Online]. Available: <https://becominghuman.ai/explaining-yolov4-a-one-stage-detector-cdac0826cbd7>. [Accessed 04 10 21].
- [27] [Online]. Available: <https://www.pyimagesearch.com/2016/11/07/intersection-over-union-iou-for-object-detection/>. [Accessed 12 12 21].
- [28] J. Redmon, S. Divvala, R. Girshick and A. Farhadi, "You Only Look Once: Unified, Real-Time Object Detection," 2016.
- [29] [Online]. Available: <https://jonathan-hui.medium.com/real-time-object-detection-with-yolo-yolov2-28b1b93e2088>. [Accessed 12 11 21].
- [30] [Online]. Available: <https://jonathan-hui.medium.com/real-time-object-detection-with-yolo-yolov2-28b1b93e2088>. [Accessed 01 12 21].
- [31] [Online]. Available: <http://drogipubliczne.eu/>. [Accessed 21 11 21].
- [32] L. Hui and L. Quan, "Layout Research on Traffic Information Sign of Ring Expressway," in *International Conference on Intelligent Computation Technology and Automation*, 2010.

- [33] S. Zhu, L. Liu and X. Lu, "Color-Geometric Model for Traffic Sign Recognition," in "*Computational Engineering in Systems Applications*"(CESA), Beijing, China, 2006.
- [34] M. Lutz, Python Wprowadzenie, IV ed., Helion, 2011.
- [35] "Roboflow," [Online]. Available: <https://roboflow.com/>. [Accessed 12 12 21].
- [36] [Online]. Available: <https://www.youtube.com/watch?v=vRqSO6RsptU>. [Accessed 12 12 21].
- [37] [Online]. Available: <https://arxiv.org/abs/1909.05626>. [Accessed 12 12 21].
- [38] C. E. Kim, M. M. D. Oghaz, V. A. Jiri Fajtl and P. Remagnino, "A Comparison of Embedded Deep Learning Methods for Person," in *3rd Asia-Pacific World Congress*.
- [39] [Online]. Available: <https://blog.roboflow.com/a-thorough-breakdown-of-yolov4/>. [Accessed 25 09 21].
- [40] [Online]. Available: <https://docs.aws.amazon.com/machine-learning/latest/dg/model-fit-underfitting-vs-overfitting.html>. [Accessed 05 11 21].
- [41] [Online]. Available: <https://ichi.pro/pl/funkcje-aktywacji-wszystko-co-musisz-wiedziec-58662895551070>. [Accessed 17 06 21].
- [42] [Online]. Available: <https://medium.com/@RaghavPrabhu/understanding-of-convolutional-neural-network-cnn-deep-learning-99760835f148>. [Accessed 21 10 21].
- [43] [Online]. Available: <https://medium.com/geekculture/a-2021-guide-to-improving-cnns-training-strategies-training-methodology-regularization-b4af696f854d>. [Accessed 16 11 21].
- [44] [Online]. Available: https://pl.wikipedia.org/wiki/Tablica_pomy%C5%82ek. [Accessed 13 11 21].
- [45] [Online]. Available: <https://www.quora.com/What-is-leaky-ReLU>. [Accessed 01 12 21].
- [46] [Online]. Available: <https://www.pyimagesearch.com/2016/11/07/intersection-over-union-iou-for-object-detection/>. [Accessed 03 12 21].
- [47] [Online]. Available: <https://www.prawo-jazdy-360.pl/znaki-drogowe>. [Accessed 11 10 21].
- [48] [Online]. Available: <https://planetagracza.pl/los-santos-z-gta-v-zyskuje-sporo-po-takich-zmianach-swietne-screeny/>. [Accessed 11 12 21].

11. SPIS RYSUNKÓW

| | |
|--|----|
| Rysunek 1 Współczesne gry komputerowe stanowią najwygodniejsze źródło takiej symulacji. Widok ekranu gry Microsoft Flight Simulator [1]..... | 9 |
| Rysunek 2 Widok okna programu LabelImg, służącego do etykietowania obiektów [Źródło: opracowanie własne]..... | 11 |
| Rysunek 3 Efekt segmentacji semantycznej [6] | 14 |
| Rysunek 4 Zastosowanie technologii LiDAR w grze GTAV [7] | 15 |
| Rysunek 5 Przykład zastosowania uczenia nienadzorowanego do analizy skupień. Widać na wykresie podobieństwo obiektów z klas „kot” i „pies”, oraz znaczącą różnicę pomiędzy klasami „żaba” i „statek” [11]..... | 18 |
| Rysunek 6 Zobrazowanie różnic pomiędzy modelami wytrenowanymi właściwie oraz nieprawidłowo [40]..... | 20 |
| Rysunek 7 Analogia pomiędzy neuronem biologicznym a tym sztucznej inteligencji [14].... | 22 |
| Rysunek 8 Przebieg funkcji sigmoidalnej [41]..... | 26 |
| Rysunek 9 Przebieg funkcji ReLU [18]..... | 27 |
| Rysunek 10 Przebieg funkcji Leaky ReLU [45]..... | 27 |
| Rysunek 11 Przebieg funkcji ELU [19]..... | 28 |
| Rysunek 12 Zobrazowanie istoty normalizacji z wykorzystaniem wykresów cech [22] | 32 |
| Rysunek 13 Zobrazowanie właściwie dobranego stopnia wielomianu funkcji kosztu [Źródło: opracowanie własne]..... | 34 |
| Rysunek 14 Wskazanie właściwego momentu przerwania trenowania przy wykorzystaniu metody "Early stopping" [Źródło: opracowanie własne] | 34 |
| Rysunek 15 Zobrazowanie istoty metody Dropout [43]..... | 35 |
| Rysunek 16 Idea działania sieci konwolucyjnej [42] | 37 |
| Rysunek 17 Wpływ filtra pionowego i poziomego na obraz [Źródło: opracowanie własne] . | 38 |
| Rysunek 18 Zasada działania warstwy łączącej – maksymalizującej oraz uśredniającej [23] | 39 |
| Rysunek 19 Schemat ilustrujący istotę działania metody „Skip connection” [Źródło: opracowanie własne]..... | 40 |
| Rysunek 20 Zestawienie architektur sieci dla porównania ich szybkości i dokładności działania [38]..... | 43 |
| Rysunek 21 Zobrazowanie jak szybkość pracy wpływa na dokładność dla różnych sieci [39] | |
| | 43 |

| | |
|---|----|
| Rysunek 22 Tablica pomyłek służąca do obliczenia wskaźników – precyzyji oraz czułości [44] | 45 |
| Rysunek 23 Przykład działania detektora obiektów dający możliwość obliczenia współczynnika IoU [46]..... | 46 |
| Rysunek 24 Kod funkcji realizującej wyliczenie IoU [Źródło: opracowanie własne] | 46 |
| Rysunek 25 Znaki drogowe wybrane do eksperymentu. Od lewej Ustęp pierwszeństwa(A-7), Droga z pierwszeństwem(D-1), Zakaz wyprzedzania(B-25) [47] | 50 |
| Rysunek 26 Widok ekranu gry Grand Theft Auto V [48] | 51 |
| Rysunek 27 Widok ekranu gry Eurotruck Simulator 2 [Źródło: opracowanie własne] | 52 |
| Rysunek 28 Gra Eurotruck Simulator 2 zapewnia zmienność pory dnia i warunków atmosferycznych na wysokim poziomie odwzorowania szczegółów..... | 53 |
| Rysunek 29 Widok ekranu aplikacji Google Maps [Źródło: opracowanie własne] | 54 |
| Rysunek 30 Zdjęcia pozyskane z Google Maps różnią się aurą pogodową w tle i oświetleniem obiektów [Źródło: opracowanie własne] | 55 |
| Rysunek 31 Zdjęcia testowe wykonane jesienną porą zwiększąją trudność w rozpoznawaniu znaków przez zmianę koloru tła [Źródło: opracowanie własne] | 56 |
| Rysunek 32 Widok okna konfiguracyjnego bazy danych. Program daje możliwość wybrania proporcji danych trenujących, walidacyjnych i trenujących, zmianę rozmiaru i dodanie modyfikacji obrazów. Zastosowano rozmycie, zaszumienie oraz przysłonięcia obrazów czarnymi polami [Źródło: opracowanie własne] | 58 |
| Rysunek 33 Widok okna wydruku z informacjami o używanej karcie graficznej [Źródło: opracowanie własne]..... | 60 |
| Rysunek 34 Przebieg treningu modeli z udziałem architektury Darknet YOLOv4 [Źródło: opracowanie własne]..... | 63 |
| Rysunek 35 Fotografia z naniesioną predykcją modelu w postaci granatowej obwiedni oraz ramka referencyjna w kolorze zielonym [Źródło: opracowanie własne]..... | 64 |
| Rysunek 36 Zależność precyzyji i czułości modelu od progu wartości confidence - dane trenujące: rzeczywiste, znak: Droga z pierwszeństwem [Źródło: opracowanie własne]..... | 65 |
| Rysunek 37 Zależność precyzyji i czułości modelu od progu wartości confidence - dane trenujące: rzeczywiste, znak: Ustęp pierwszeństwa [Źródło: opracowanie własne] | 65 |
| Rysunek 38 Zależność precyzyji i czułości modelu od progu wartości confidence - dane trenujące: rzeczywiste, znak: Zakaz wyprzedzania [Źródło: opracowanie własne] | 66 |
| Rysunek 39 Zależność precyzyji i czułości modelu od progu wartości confidence - dane trenujące: wirtualne, znak: Droga z pierwszeństwem [Źródło: opracowanie własne] | 66 |
| Rysunek 40 Zależność precyzyji i czułości modelu od progu wartości confidence - dane trenujące: wirtualne, znak: Ustęp pierwszeństwa [Źródło: opracowanie własne | 67 |

| | |
|---|----|
| Rysunek 41 Zależność precyzji i czułości modelu od progu wartości confidence - dane trenujące: wirtualne, znak: Zakaz wyprzedzania [Źródło: opracowanie własne]..... | 67 |
| Rysunek 42 Porównanie zmiany parametrów opisujących jakość predykcji w zależności od poziomu progu, model uczony na danych wirtualnych [Źródło: opracowanie własne] | 68 |
| Rysunek 43 Porównanie zmiany parametrów opisujących jakość predykcji w zależności od poziomu progu, model uczony na danych rzeczywistych [Źródło: opracowanie własne] | 68 |
| Rysunek 44 Średnia wartość precyzji dla każdego znaku, wybranego do eksperymentu [Źródło: opracowanie własne]..... | 69 |
| Rysunek 45 Średnia wartość czułości dla każdego znaku, wybranego do eksperymentu [Źródło: opracowanie własne]..... | 69 |
| Rysunek 46 Średnia wartość confidence dla każdego znaku, wybranego do eksperymentu [Źródło: opracowanie własne] | 70 |
| Rysunek 47 Średnia wartość IoU dla każdego znaku, wybranego do eksperymentu [Źródło: opracowanie własne]..... | 71 |