

# **Sztuczna Inteligencja i Inżynieria Wiedzy**

## **Opis projektu**

„System do detekcji samochodów oraz rozpoznawania tablic  
rejestracyjnych”

Mateusz Pakuła  
238336

## 1. Cel projektu

Celem projektu jest opracowanie systemu, w skład którego wchodzi funkcjonalności takie jak:

- detekcja samochodów i innych obiektów
- określanie stanów samochodu takich jak: odjazd i przyjazd z wyznaczonego miejsca parkingowego, ruch
- określanie stanu oświetlenia (włączone/wyłączone)
- wykrywanie tablicy rejestracyjnej, segmentacja symboli z tablicy rejestracyjnej i ich klasyfikacja

## 2. Opis działania systemu

System zaimplementowany został w języku Python z wykorzystaniem bibliotek OpenCV, PyTorch i NumPy. Działanie opiera się na: określeniu stanu oświetlenia, wykryciu obiektów, śledzeniu, klasyfikacji stanu samochodów, wykryciu tablicy rejestracyjnej, segmentacji i następnie klasyfikacji znaków z tablicy rejestracyjnej.

### Określanie stanu oświetlenia

Do określania poziomu światła na obrazie została użyta funkcja, która oblicza średnią wartość piksela:

$$ps(img) = \frac{\sum_{y=0}^h \sum_{x=0}^w img[y][x]}{w * h}$$

gdzie:

img – obraz, przekonwertowany na skalę szarości (w projekcie użyto metody color z opencv)

w – szerokość obrazu w pikselach

h – długość obrazu w pikselach

Światło zostaje uznane za włączone, jeżeli wartość funkcji ps dla obrazu jest większa od poziomu ustalonego w programie (wartość domyślna w programie została arbitralnie ustawiona na 20)

### Detekcja

Do wykrywania obiektów na obrazie wykorzystano rozwiązanie o nazwie Single Shot MultiBox Detector (<https://arxiv.org/abs/1512.02325>), bazujące na konwolucyjnej sieci neuronowej MobileNetV1.

### Dane uczące

Dane uczące, na których został wytrenowany model składają się z danych utworzonych na bazie dostarczonych materiałów wideo obrobionych w narzędziu Scalabel (<https://www.scalabel.ai/>), oraz zbioru Open Images (<https://storage.googleapis.com/openimages/web/index.html>). Do pobierania

danych ze zbioru Open Images przygotowany został skrypt **open\_images\_downloader.py**, które można wywołać poleceniem o składni:

```
python open_images_downloader.py --root ŚCIEŻKA --class_names "nazwa1,nazwa2,..."  
[--include_depiction] [--num_workers L_WĄTKÓW] [--retry L_PRÓB] [--filter_file "id1,id2,id3"]  
[--remove_overlapped]
```

gdzie:

- root – katalog główny, w którym przechowywany ma być zbiór danych
- class\_names – nazwy klas, tj. samochód, które mają zostać pobrane
- include\_depiction – mówi, że dane mają zawierać opis
- num\_workers – liczba pracujących wątków
- retry – liczba prób podczas pobierania
- filter\_file – identyfikatory plików, które mają zostać zignorowane
- remove\_overlapped – mówi, że etykiety przesłonięte przez inne etykiety mają zostać usunięte

## Uczenie modelu

Szczegóły dotyczące funkcji celu dostępne są w dokumentacji modelu Single Shot MultiBox Detector. Do optymalizacji funkcji celu wykorzystana została metoda stochastycznego spadku wzdłuż gradientu. Do trenowania modelu został przygotowany skrypt **train\_ssd.py**, uruchamiany za pomocą polecenia o składni:

```
python train_ssd.py --train_datatype {bdd | open_images}  
--validation_datatype {bdd | open_images} --datasets "ściezka1 sciezka2 ..."  
--validation_dataset ŚCIEŻKA [--labels ŚCIEŻKA] [--freeze_base_net]  
[{--lr | --learning_rate} WARTOŚĆ] [--momentum WARTOŚĆ] [--weight_decay WARTOŚĆ]  
[--gamma WARTOŚĆ] [--base_net_lr WARTOŚĆ] [--extra_layers_lr WARTOŚĆ]  
[{--base_net ŚCIEŻKA | --pretrained_net ŚCIEŻKA | --resume ŚCIEŻKA}]  
[--scheduler {multi-step [--milestones WARTOŚĆ] | cosine [--t_max WARTOŚĆ]]  
[--batch_size WARTOŚĆ] [--num_epochs WARTOŚĆ] [--num_workers WARTOŚĆ]  
[--validation_epochs WARTOŚĆ] [--debug_steps WARTOŚĆ] [--use_cuda {True | False}]  
[--checkpoint_folder ŚCIEŻKA] [--balance_data]
```

gdzie:

- train\_datatype – typ zbioru danych uczących; musi być podany osobno dla każdej podanej ścieżki, np. mając 2 zbiory: jeden z narzędzia Scalabel, a drugi z Open Images należy napisać „--train\_datatype bdd open\_images”
- validation\_datatype – typ zbioru walidacyjnego
- datasets – ścieżka lub ścieżki do zbiorów treningowych
- validation\_dataset – ścieżka do zbioru walidacyjnego
- labels – ścieżka do pliku tekstowego z etykietami oddzielonymi przecinkiem, który mówi jaki numer powinien zostać przypisany danej etykietce, np. mając etykiety „car”, „bike” i „person”, umieszczając w pliku „car,person,bike” etykieta „car” będzie miała nr 1, etykieta „person” nr 2, a etykieta „bike” nr 3

--freeze\_base\_net – mówi o tym, że wagi bazowej sieci (MobileNetV1) nie powinny być zmieniane  
--lr lub --learning\_rate – początkowy współczynnik długości kolejnego kroku w gradiencie  
--momentum – współczynnik pędu gradientu stochastycznego  
--weight\_decay – współczynnik regularyzacji L2 gradientu stochastycznego  
--gamma – współczynnik gamma gradientu stochastycznego  
--base\_net\_lr – współczynnik długości kroku dla sieci bazowej (MobileNetV1)  
--extra\_layers\_lr – współczynnik długości kroku dla pozostałych warstw modelu  
--base\_net – ścieżka do sieci bazowej (sieć jest inicjalizowana przez metodę **init\_from\_base\_net**)  
--pretrained\_net – ścieżka pretrenowanej sieci (inicjalizacja przez metodę **init\_from\_pretrained\_ssd**)  
--resume – ścieżka do trenowanej wcześniej sieci („zwykła” inicjalizacja przez metodę **load**)  
--scheduler – wybór planisty do dynamicznego dostosowywania kroku w gradiencie  
--milestones – parametr określający „kamienie milowe” w planiście MultiStepLR  
--t\_max – maksymalna liczba iteracji w planiście Cosine Annealing  
--batch\_size – rozmiar batcha  
--num\_epochs – liczba epok, które mają się wykonać

--num\_workers – liczba wątków  
--validation\_epochs – określa liczbę epok, co które ma być dokonywana walidacja i zapis modelu  
--debug\_steps – liczba kroków, co które ma zostać wyświetlony log  
--use\_cuda – wartość logiczna wskazująca, czy używać GPU, czy nie  
--checkpoint\_folder – ścieżka do zapisu modelu  
--balance\_data – użycie powoduje wyrównanie liczby etykiet każdej klasy

## Śledzenie

Śledzenie samochodów jest procesem podzielonym na 3 etapy:

- pobranie wstępnego zbioru z detekcji obiektów
- utworzenie unikalnego identyfikatora dla każdego z samochodów
- monitorowanie każdego obiektu-samochodu podczas kolejnych klatek, zachowując przydzielone im unikalne identyfikatory

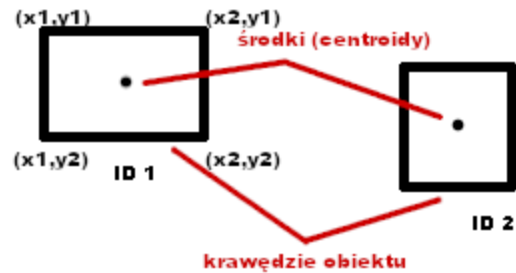
## Pobranie wstępnego zbioru

Podczas pobrania początkowego zbioru wykrytych obiektów, obiekty oznaczone jako samochód są odseparowywane od pozostałych.

## Utworzenie unikalnego ID

Każdy odseparowany samochód otrzymuje swój własny identyfikator.

## Monitorowanie

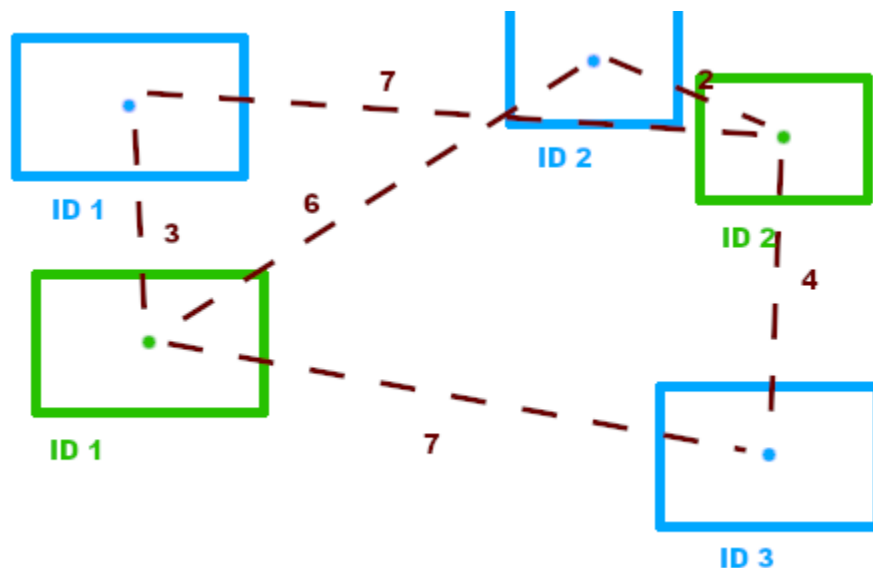


Rys. 1

Pierwszym krokiem jest obliczenie współrzędnych środka każdego z obiektów (samochodów) w układzie współrzędnych. Dla każdego prostokąta-ramki współrzędne wyliczane są ze wzorów:

$$x = \frac{x_1 + x_2}{2} \quad y = \frac{y_1 + y_2}{2}$$

Kolejnym krokiem jest porównanie środków obiektów z poprzedniej iteracji do obiektów z nowej iteracji.



Rys. 2 – Odległości pomiędzy środkami prostokątów

Dla każdego środka prostokąta z nowej iteracji (kolor niebieski) liczona jest odległość do każdego środka prostokąta z nowej iteracji (kolor zielony) zgodnie ze wzorem:

$$l(a, b) = |b_x - a_x| + |b_y - a_y| \quad (a, b - \text{środki prostokątów, będące punktami w układzie współrzędnych})$$

Następnie każdemu obiektowi z nowej iteracji przyznawany jest identyfikator tego prostokąta, którego środek był najbliższym środkiem nowego prostokąta. W rysunku drugim – niebieski prostokąt w lewym górnym rogu jest najbliższym prostokątem o ID 1, dlatego też zostaje mu przyznany jego identyfikator. W przypadku większej liczby nowych obiektów-samochodów, niż w poprzedniej iteracji, przyznawany jest nowy identyfikator.

## Określanie stanu samochodów

W systemie istnieją 4 stany opisujące samochód: domyślny – bez nazwy, w ruchu – MOVE, samochód zaparkował w pewnym miejscu parkingowym – ARRIVED, samochód wyjeżdża z ww. miejsca parkingowego – LEFT. Etykieta MOVE przyznawana jest samochodowi, którego środek przesunął się w jednej iteracji o 5 jednostek według wzoru na odległość między punktami stosowanego przy śledzeniu obiektów. Etykieta ARRIVED przyznawana jest wtedy, gdy prostokąt otaczający samochód i prostokąt, którym oznaczone jest miejsce parkingowe mają część wspólną. Etykieta LEFT przypisywana jest na określoną liczbę iteracji samochodowi, który poprzednio posiadał etykietę ARRIVED, a jego prostokąt-ramka i prostokąt oznaczający miejsce parkingowe nie nachodzą już na siebie.

## Wykrywanie tablicy rejestracyjnej

Pierwszym krokiem do odnalezienia fragmentu obrazu z tablicą rejestracyjną jest utworzenie binarnego obrazu na bazie obrazu (w skali szarości) z wykrytym przez SSD samochodem. Stosowane do tego jest progowanie adaptacyjne, gdzie obraz dzielony jest na obszary, potem w każdym obszarze obliczana jest średnia wartość intensywności i dla każdego obszaru dzielony jest osobny próg. (W projekcie wykorzystana została metoda adaptiveThreshold z opencv).



*Rys. 3 - Samochód w skali szarości*



*Rys. 4 - Samochód po zastosowaniu progowania binarnego*

Kolejnym krokiem jest utworzenie zbioru regionów z połączonymi pikselami (w projekcie wykorzystano metode `measure.labels` z pakietu `skimage`). Piksele nazywane są połączonymi wtedy, gdy są sąsiadujące oraz mają takie same wartości i wyselekcjonowaniem tych, które spełniają określone warunki: wysokość zawiera się w ustalonym przedziale; szerokość zawiera się w ustalonym przedziale; szerokość obrazu jest co najmniej trzy razy mniejsza niż wysokość; obszar w całości znajduje się w dolnej połowie obrazu. Następnie dla każdego obszaru, który spełnia kryteria dokonywana jest segmentacja.

## Segmentacja znaków tablicy rejestracyjnej

Początkową operacją przeprowadzoną na obrazie z tablicą rejestracyjną (już w skali szarości) jest nałożenie filtru bilateralnego, aby zredukować szum, a następnie nałożenie filtru Canny'ego, aby wyszczególnić krawędzie znaków.



Rys. 4 – Tablica rejestracyjna w skali szarości      Rys. 5 – Tablica rejestracyjna po filtrze Canny'ego

Następnie, analogicznie do wykrywania tablicy rejestracyjnej, tworzony jest zbiór regionów z połączonymi pikselami, które spełniają określone kryteria: wysokość i szerokość zawierają się w określonych przedziałach.



Rys. 6 – Na zielono zaznaczone regiony spełniające określone kryteria.

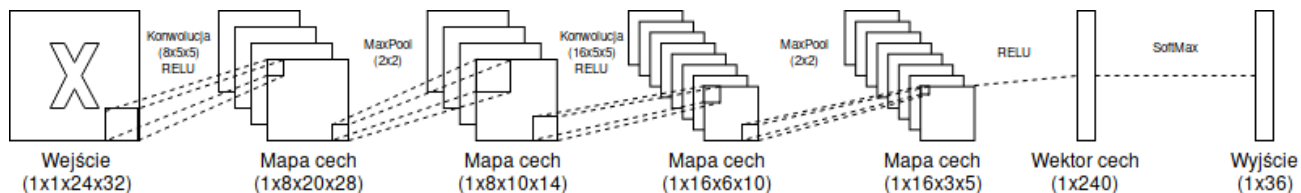
Później dokonywana jest korekta, w celu uzupełnienia możliwych brakujących regionów. Proces polega na sprawdzeniu, czy odległość pomiędzy środkami sąsiadujących regionów jest większa niż dwukrotność średniej szerokości regionu, jeżeli tak – wydzielany jest nowy region, którego wymiary są średnią z rozmiarów dwóch sąsiadów i koordynaty są średnią koordynatów sąsiadów.



Rys. 7 – Kolorem niebieskim został oznaczony region dodany podczas korekcji.

## Klasyfikacja znaków

Do rozpoznawania znaków została wykorzystana konwolucyjna sieć neuronowa o strukturze:



Rys. 8 – Schemat architektury konwolucyjnej sieci neuronowej.

Gdzie przesunięcie przy operacjach konwolucji jest równe 1, a MaxPool 2. Dwa pierwsze wymiary wejścia (1x1) związane są z faktem, że kolory obrazu są w skali szarości zamiast w formacie RGB, a sam obraz posiada tylko jedną warstwę.

## Dane uczące

Do uczenia modelu wykorzystany został plik graficzny, wewnątrz którego zebrane są wszystkie znaki występujące na tablicach rejestracyjnych oraz plik tekstowy mapujący nazwę klasy do . Zostały one podzielone na osobne obrazy o wymiarach 24x32. Dodatkowo cały zbiór został uzupełniony o 10000 kopii obrazów dla każdej klasy, na które zostały zaaplikowane metody data augmentation: skalowanie od 30% do 140%, obrót od -20 do 20 stopni, odwrócenie kolorów obrazu, dropout do 40% pikseli, rozmycie gaussa.

## Uczenie modelu

Do optymalizacji funkcji celu wykorzystana została metoda stochastycznego spadku wzdłuż gradientu. Do trenowania modelu został przygotowany skrypt **train\_cnn.py**, uruchamiany za pomocą polecenia o składni:

```
python train_cnn.py [--use_cuda {True | False}] [--dataset ŚCIEŻKA] [--batch_size WARTOŚĆ]
[--learning_rate WARTOŚĆ] [--num_epochs WARTOŚĆ] [--store_path ŚCIEŻKA]
[--model ŚCIEŻKA] [--imgaug {True [--mult WARTOŚĆ] | False}]
```

gdzie:

- use\_cuda – parametr mówiący o tym, czy podczas uczenia ma być wykorzystywany GPU
- dataset – ścieżka do folderu, w którym znajduje się plik charset.png i labels.txt
- batch\_size – rozmiar batcha
- learning\_rate – współczynnik długości kroku w gradiencie
- num\_epochs – liczba epok, przez które będzie trenowany model
- store\_path – ścieżka do katalogu, w którym zapisany zostanie model po wyuczeniu
- model – ścieżka do modelu, który ma być trenowany
- imgaug – parametr mówiący o tym, czy dane mają zostać poddane procesowi Data Augmentation
- mult – wartość mówiąca o tym, ile razy zbiór danych ma zostać zwielokrotniony przed procesem Data Augmentation