

Projekt zespołowy

Analiza obrazu w symulatorze Carla

Opiekun: dr inż. Witold Czajewski

Zespół: Karol Urban, Arsentii Plotavchenko

Część pracy wykonana przez Karol Urbana:

1. Ogólny opis realizacji i wyników

Podczas projektu zostało zrealizowane napisane 3 skrypty w języku Python:

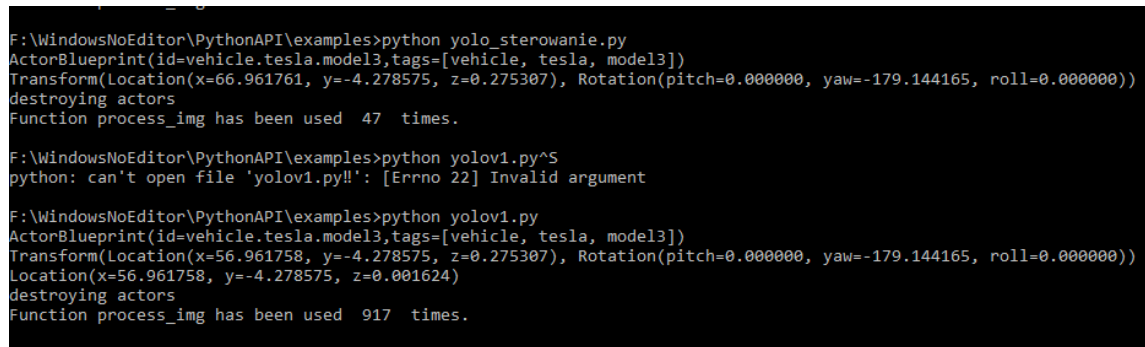
- yolo1.py
- yolo_odleglosc.py
- yolo_sterowanie.py

Pierwszy z nich tworzy samochód w środowisku Carla sterowany autopilotem wraz z kamerą przyczepioną do przodu samochodu, a następnie przetwarza obraz poprzez zastosowanie algorytmu „yolo” z użyciem zdefiniowanych przez użytkownika parametrów sieci.

Następny skrypt służy do weryfikacji algorytmu wyznaczania odległości. Tworzy on dwa samochody których środki są oddalone od siebie o 10 metrów. Algorytm zwraca wartość 6,37m jest ona poprawna ponieważ jest to odległość pomiędzy przodem a tyłem samochodu.



Ostatnim skrypt odpowiada za próbę podążania za samochodem sterowanym autopilotem. Algorytm opiera się na wyznaczeniu odległości od wykrytego samochodu i dostosowaniu w zależności od niej prędkości. Natomiast sterowanie skręcaniem polega na ustawieniu środka ramki wykrytego samochodu w środku obrazu kamery. W przypadku braku wykrycia samochodu sterowanie przedstawionym algorytm zamienia się na sterowanie autopilotem. Niestety w tym przypadku algorytm wykonuje się za wolno i samochód albo się nie porusza albo porusza się na autopilocie. Poniższe zdjęcie pokazuje ile razy wywołuje się funkcja w przypadku wywołania skryptu yolo1.py(917 razy) i yolo_sterowanie.py(47 razy). Może to być problem z wielowątkowością Pythona.



```
F:\WindowsNoEditor\PythonAPI\examples>python yolo_sterowanie.py
ActorBlueprint(id=vehicle.tesla.model3,tags=[vehicle, tesla, model3])
Transform(Location(x=66.961761, y=-4.278575, z=0.275307), Rotation(pitch=0.000000, yaw=-179.144165, roll=0.000000))
destroying actors
Function process_img has been used 47 times.

F:\WindowsNoEditor\PythonAPI\examples>python yolo1.py^S
python: can't open file 'yolo1.py!': [Errno 22] Invalid argument

F:\WindowsNoEditor\PythonAPI\examples>python yolo1.py
ActorBlueprint(id=vehicle.tesla.model3,tags=[vehicle, tesla, model3])
Transform(Location(x=56.961758, y=-4.278575, z=0.275307), Rotation(pitch=0.000000, yaw=-179.144165, roll=0.000000))
Location(x=56.961758, y=-4.278575, z=0.001624)
destroying actors
Function process_img has been used 917 times.
```

Warto wspomnieć o próbie stworzenia inteligentnego agenta którego zadaniem miało być nauczenie się poruszania się w symulacji. Jego zadaniem było wykorzystanie wyników sieci yolo do sterowania prędkością. Jednak ze względu na nierozwiązany problem z bibliotekami nie udało się zrealizować tego zadania, efektem tego jest skrypt reinforcment_projekt.py.

2. Szczegółowa realizacja

Aby wszystkie skrypty działały muszą one się znajdować w folderze PythonAPI\examples. Aby je uruchomić trzeba wcześniej uruchomić Carłę, a następnie uruchomić konsolę w folderze examples i wywołać je w konsoli np. python yolo1.py. W skryptach są komentarze ułatwiające zrozumienie działania programu.

Skrypt yolo1.py:

Na początku skryptu znajdują się biblioteki i zmienne globalne wykorzystywane w reszcie programu. Później są zdefiniowane dwie funkcje process_img i process_img2. Służą one do przetwarzania obrazu druga z nich wyświetla obraz z kamery a druga wyświetla obraz po przepuszczeniu przez sieć yolo. Później jest właściwa część programu gdzie skrypt łączy się z serwerem Carli i stworzenie samochodu i kamery. Następnie jest uruchamiany autopilot i samochód zaczyna się poruszać. Bardzo ważna jest instrukcja: *sensor.listen(lambda data: process_img(data))*, ponieważ pozwala ona na wywołanie funkcji proces_img za każdym razem kiedy odbiera sygnał od sensora. Właśnie w dzięki tej instrukcji możliwe jest pozyskanie informacji z obrazu. Na końcu programu wszystkie elementy stworzone są usuwane i połączenie z serwerem kończy się.

Skrypt yolo_odleglosc.py:

W tym skrypcie zostały przeprowadzone bardzo podobne procesy co we wcześniejszym algorytmie z tym że została wyznaczona ogniskowa kamery i została wykorzystana wysokość ramki(otrzymanej dzięki yolo). Ogniskowa została wyznaczona przy pomocy następującego wzoru:

$$\frac{\frac{w}{2}}{f} = \tan \frac{\alpha}{2}$$

Gdzie α to HFOV(Horizontal Field of View), a w to szerokość obrazu w pikselach. Następnie przy pomocy wzoru:

$$d = \text{rzeczywista_wysokość_obiektu} * \frac{\text{ogniskowa}}{\text{wysokość_obiektu_w_pikselach}}$$

W skrypcie jest podana rzeczywista wysokość pojazdu Tesla Model 3- 1,443m. We wzorze można używać zamiennie szerokości i wysokości. Niestety nie udało się znaleźć położenia punktu orientacyjnego dla modelu Tesli Model 3 w Carli. Dlatego nie da się zweryfikować dokładnie czy odległość 6,37 m jest poprawna. Szacunkowo da można stwierdzić iż wynik jest względnie poprawny. Zakładając, iż punkty lokalizacyjne są w połowie pojazdu, a pojazd ma długość 4,694m([źródło](#)) wówczas oszacowana rzeczywista odległość wynosi 10m-4,694m = 5,306m. Wówczas różnica pomiędzy odległością wyznaczoną przez program, a rzeczywistą oszacowaną wynosi około 1m. Należy jednak pamiętać, że nie mamy dokładnych informacji na temat modelu Tesli w Carli i wcale nie oznacza to błędnego wyniku.

Skrypt yolo_sterowanie.py:

Skrypt yolo_sterowanie.py jest rozwinięciem skryptu yolo_odleglosc.py o dodanie drzewa decyzyjnego sterującym pojazdem i włączenie autopilota w pojeździe znajdującym się przed kamerą. Jest to pojazd, za którym ma podążać pojazd, którym sterujemy. W tym celu funkcja proces_img został wzbogacona o dodatkowy argument, którym jest pojazd, którym skrypt ma sterować. Dodatkowo została zdefiniowana funkcja sterowanie, która liniowo steruje prędkością w zależności od odległości od wykrytego pojazdu. A także skręcaniem w zależności od jego położenia w osi poziomej względem środka obrazu wychwytywanego kamery. Jest ona właśnie wywoływana w funkcji process_img w momencie wykrycia samochodu. Jak było wcześniej już wspomniane funkcja wykonuje się za mało razy aby móc stwierdzić jakość dobrego algorytmu sterowania. Bardzo możliwe, że problem jest z wielowątkowością Pythona lub trzeba przetestować to na sprzęcie o wyższej mocy obliczeniowej lub napisać to inaczej.

Skrypt reinforcement_projekt.py:

Jest to plik zupełnie inny, w którym wykorzystywany jest koncept reinforcement learningu. Jest to praca pochodząca ze strony: <https://pythonprogramming.net/reinforcement-learning-self-driving-autonomous-cars-carla-python/> wzbogacona o funkcję process_img wraz z obliczaniem odległości od najbliższego obiektu. Zostaje to wykorzystane w procesie przyznawania nagrody.

3. Opis prac

Prowadziłem zespół kiedy jeszcze on był zespołem to znaczy kiedy był jakiś kontakt i chęć zrobienie czegokolwiek ze strony pozostałych członków. Napisałem kody do skryptów: yolov1.py, yolo_odleglosc.py i yolo_sterowanie.py. Ponadto zainstalowałem Carlę i yolo na swoim komputerze co jest czasochłonne i nie takie łatwe jakby się mogło to wydawać. Ponadto zastanowiłem się nad problemem stworzenia inteligentnego agenta.

4. Opinia na temat pracy zespołowej

Na początku działaliśmy w miarę efektywnie spotykaliśmy się co tydzień w ciągu pierwszego miesiąca współpracy. Niestety wówczas zapoznawaliśmy się ze środowiskiem Carli, natomiast kontakt i zainteresowanie projektem urwało się w momencie podziału prac. Później brak zaangażowania i kontaktu ze strony pozostałych członków zespołu spowodował dezorganizację pracy. W zasadzie zaangażowanie w pracę zespołu było tylko z mojej strony pozostali członkowie albo się nie odzywali albo nie realizowali postawionych im zadań. Natomiast kontakt z opiekun zespołu i jego zaangażowanie uważam za bardzo dobre.

5. Podsumowanie i możliwość dalszego rozwoju

Wyniki z pomiarami odległości i zastosowaniem sieci yolo można uznać za satysfakcjonujące. Natomiast porażką okazały się próby stworzenia agentów poruszających się w świecie Carli jak i prostego sterowania w celu podążania za samochodem.

Projekt można rozwinąć poprzez zastosowanie sieci yolo w pliku domyślnym Carli `manual_control.py`. Ponadto można spróbować zastosować plik `reinforcement_projekt.py` na komputerze z dobrej jakości podzespołami i sprawnie działającymi bibliotekami.

Cześć Arsentiego Plotavchenko:

Szkolenie sieci neuronowej YOLOv4 na podstawie rzeczywistych obrazów

Aby wytrenować sieć neuronową, wykonano następujące kroki:

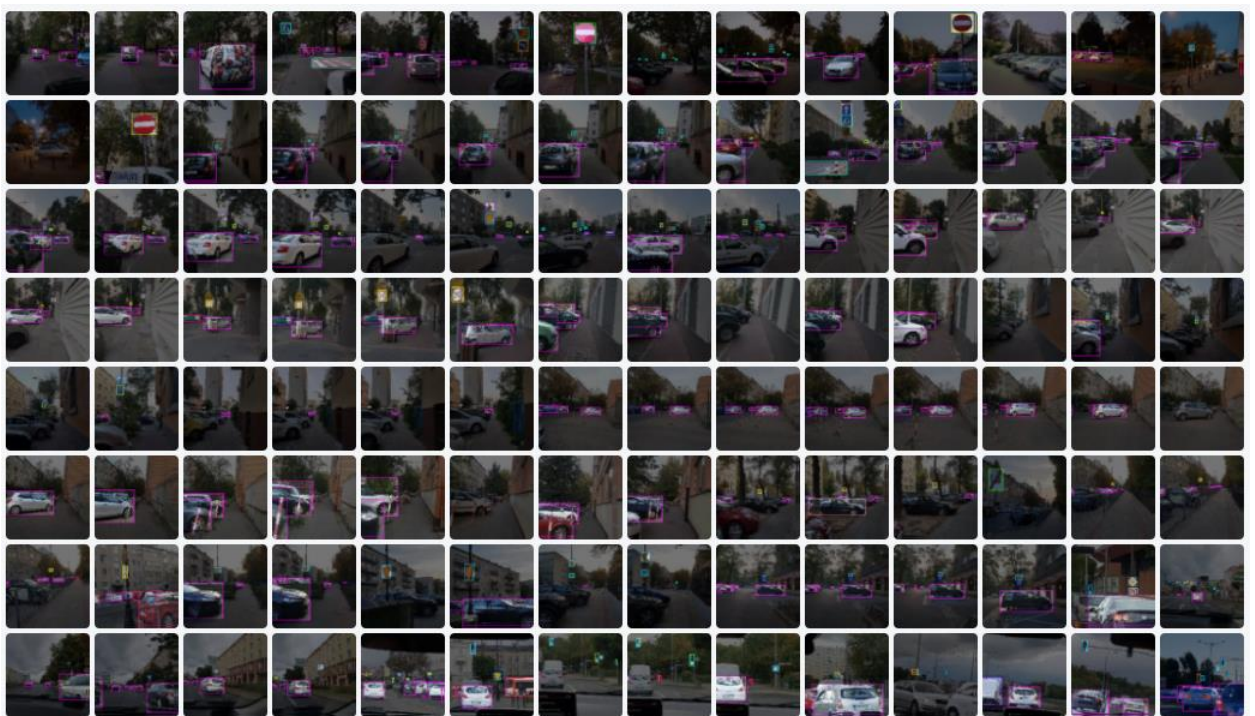
Zbieranie obrazów rzeczywistych obiektów

Adnotacja obiektów na zdjęciach i ich dodatkowa obróbka za pomocą platformy Roboflow

Szkolenie sieci neuronowej za pomocą udostępnionych narzędzi na platformie Jupiter Google collaboration.

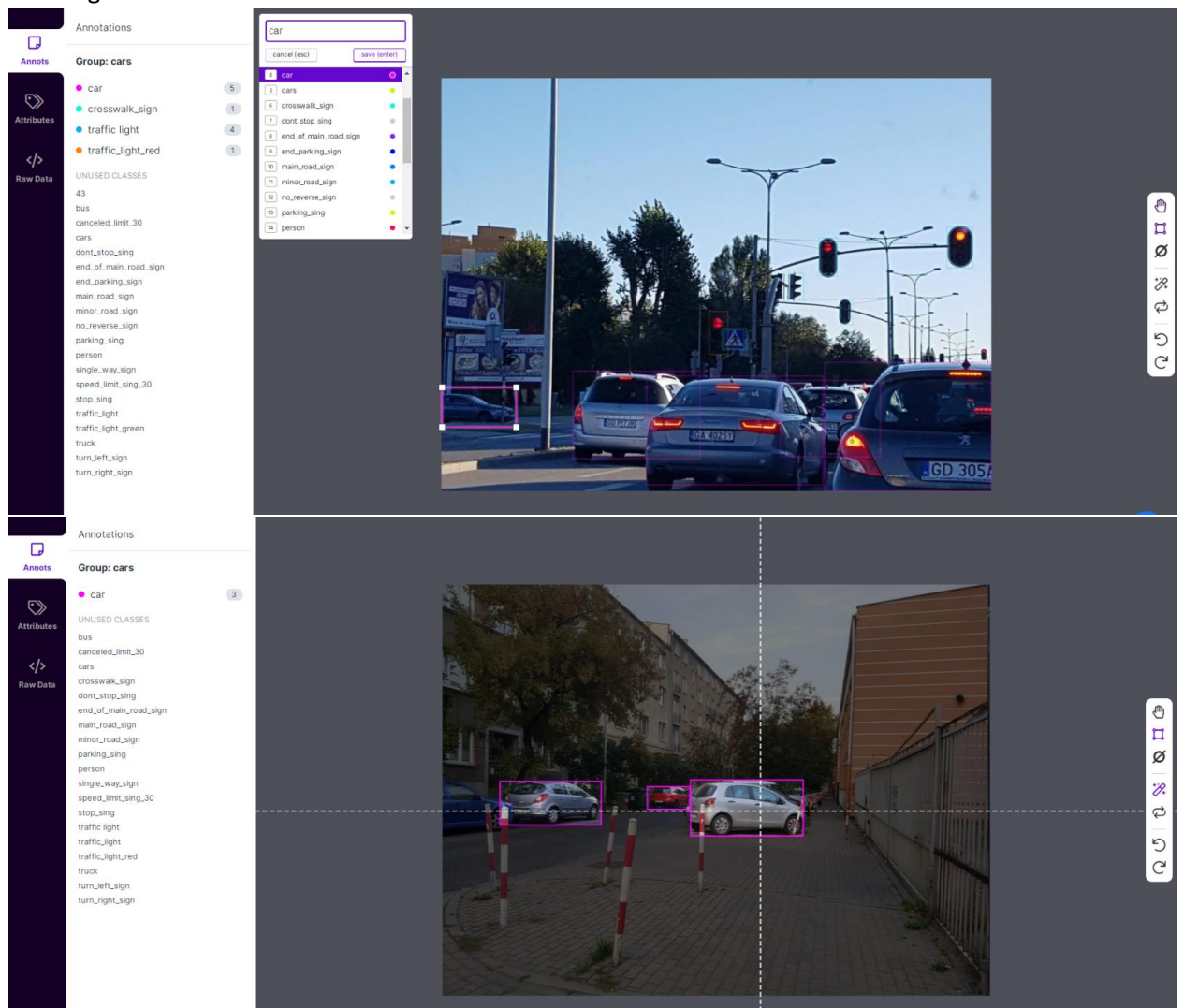
Przygotowanie materiałów

Przed wykonaniem tego zadania przygotowano bibliotekę zdjęć wykonanych na ulicy, w tym: pojazdy, znaki drogowe, różne przeszkody itp.

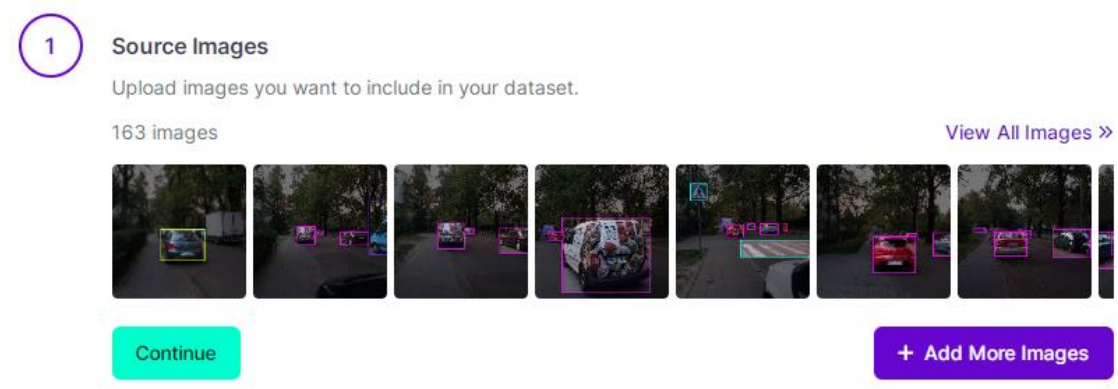


Anotacja obrazów

Na tym etapie należy dokonać selekcji i klasyfikacji obiektów na obrazach zgodnie z zasadą Bouding Box.



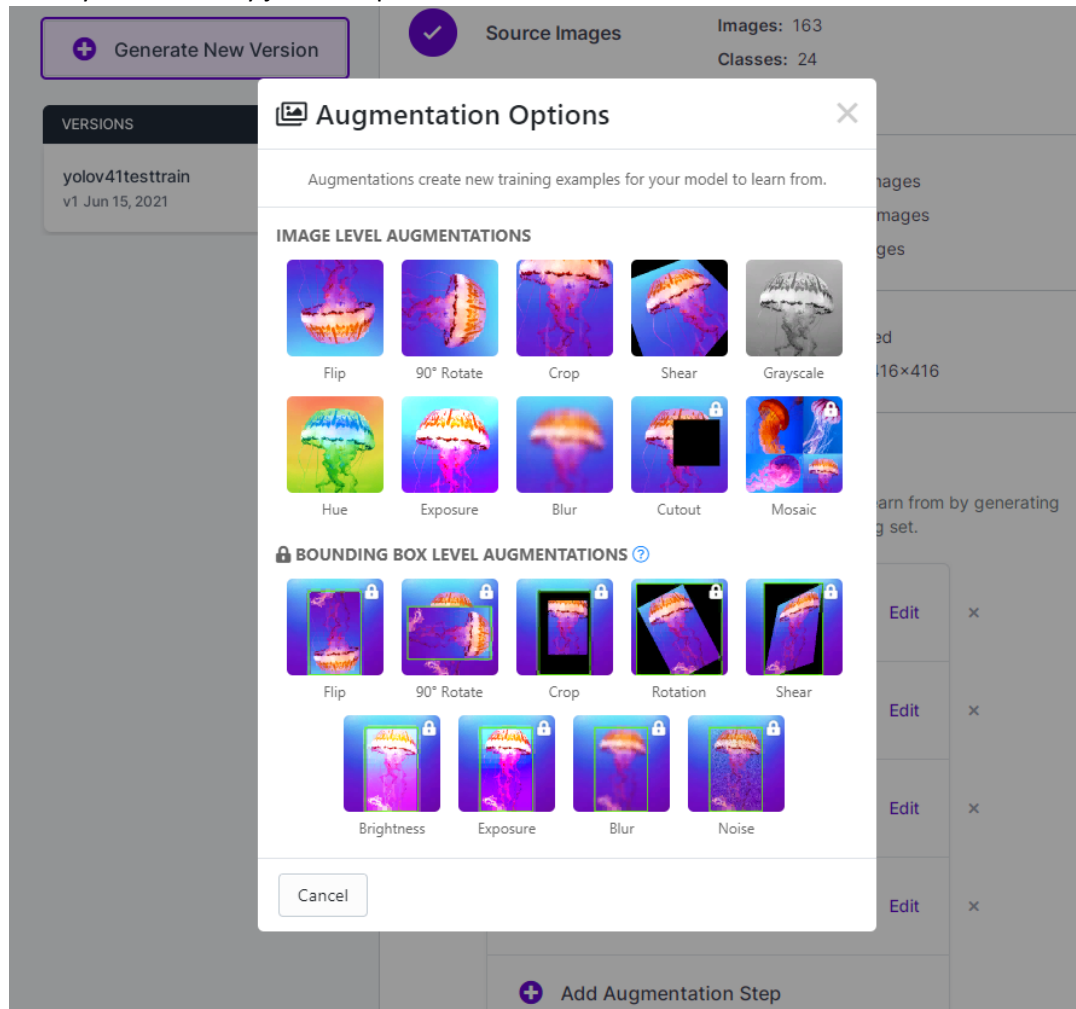
W trakcie tego procesu zaznaczamy prostokątami, co znajduje się w wybranych obszarach, jednocześnie klasyfikujemy obiekty, przy wyborze zaznaczamy, do której klasy należy ten lub inny obiekt na obrazie. Tę procedurę należy wykonać ze wszystkimi obrazami, im wyraźniej i dokładniej określone zostaną granice obiektów, tym dokładniejszy będzie trening sieci neuronowej.



W rezultacie otrzymujemy wiele obrazów z adnotacjami.

Przetwarzanie obrazów

Po zaadnotowaniu obrazów, przed skompilowaniem biblioteki adnotacji, platforma Roboflow oferuje obróbkę obrazów w celu wykonania na ich podstawie wielu zmodyfikowanych kopii z takimi efektami jak: rozmycie, obrót, zmiana perspektywy, przypadkowe przerwy w obrazach, zmiany kontrastu czy jasność itp.



Niektóre z tych funkcji są płatne, więc w tej pracy wykorzystano tylko bezpłatne opcje.

Po procesie przetwarzania ich zmodyfikowane kopie wraz z pozostałymi adnotacjami odpowiednich klasyfikacji są dodane do oryginalnej biblioteki obrazów.

TRAINING OPTIONS

Use Roboflow Train

Let us train your model and get results within 24 hours along with a hosted API endpoint for making predictions. [Learn More >](#)

[Start Training](#)

Available Credits: 0

Train Outside Roboflow

Export your data to use a model from [our model library >](#) with Google Colab or your own machine.

Format

YOLO Darknet

[Export](#)

IMAGES



391 images

[View All Images >](#)

TRAIN / TEST SPLIT

Training Set

87%

342 images

Validation Set

8%

33 images

Testing Set

4%

16 images

Na etapie tworzenia biblioteki adnotacji otrzymujemy zestawy obrazów do szkolenia, sprawdzania i testowania sieci.

Na obrazach uczących sieć przeprowadzi uczenie, obrazy testowe służą jako znaczniki kontrolne do samodzielnego sprawdzania sieci, czy jest ona prawidłowo uczona. Zestaw obrazów testowych jest niezbędny do późniejszej weryfikacji poprawności definiowania obiektów przez osobę, jako elementów, których sieć nigdy nie widziała.

Z oryginalnych 163 zdjęć uzyskaliśmy 391 zdjęcie do uczenia sieci, uwzględniając oryginalne zdjęcia oraz ich zmodyfikowane kopie, które wyeksportujemy do dalszego uczenia sieci neuronowej.

Szkolenie sieci neuronowej

Sieć neuronowa została przeszkolona na platformie Jupiter.

Umożliwiło to trenowanie sieci z wykorzystaniem mocy obliczeniowej dostarczanej przez Google.

```
! /usr/local/cuda/bin/nvcc --version

nvcc: NVIDIA (R) Cuda compiler driver
Copyright (c) 2005-2020 NVIDIA Corporation
Built on Wed_Jul_22_19:09:09_PDT_2020
Cuda compilation tools, release 11.0, V11.0.221
Build cuda_11.0_bu.TC445_37.28845127_0

[2] !nvidia-smi

Tue Jun 15 18:35:02 2021
+-----+
| NVIDIA-SMI 465.27      Driver Version: 460.32.03      CUDA Version: 11.2      |
+-----+-----+-----+
| GPU   Name           Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf    Pwr:Usage/Cap|      Memory-Usage | GPU-Util  Compute M. |
|                                           MIG M.         |
+-----+-----+-----+
|  0   Tesla T4             Off   | 00000000:00:04.0 Off  |             0        |
| N/A   45C    P8      9W / 70W | 0MiB / 15109MiB |      0%    Default   |
|                                           N/A              |
+-----+-----+-----+

+-----+
| Processes:                                     GPU Memory |
|  GPU   GI    CI          PID    Type    Process name                  Usage       |
|-----+-----+-----+
| No running processes found                       |
+-----+
```

Przed rozpoczęciem uczenia sieci neuronowej zostały wykonane polecenia instalacji bibliotek YOLOv4, Python i Darknet. Zainstalowane są również sterowniki i biblioteki dla GPU do przetwarzania dużych ilości danych: CUDA, CUDNN, OPECV. Następnie obrazy z adnotacjami są przygotowywane do procesu szkolenia.

Następnie - wypełnienie pliku konfiguracyjnego, są to ustawienia według których będzie trenowana sieć neuronowa.

```

#we build config dynamically based on number of classes
#we build iteratively from base config files. This is the same file shape as cfg/yolo-obj.cfg
def file_len(fname):
    with open(fname) as f:
        for i, l in enumerate(f):
            pass
        return i + 1

num_classes = file_len('train/_darknet.labels')
print("writing config for a custom YOLOv4 detector detecting number of classes: " + str(num_classes))

#Instructions from the darknet repo
#change line max_batches to (classes*2000 but not less than number of training images, and not less than 6000),
#f.e. max_batches=6000 if you train for 3 classes
#change line steps to 80% and 90% of max_batches, f.e. steps=4800,5400
if os.path.exists('./cfg/custom-yolov4-detector.cfg'): os.remove('./cfg/custom-yolov4-detector.cfg')

with open('./cfg/custom-yolov4-detector.cfg', 'a') as f:
    f.write('[net]' + '\n')
    f.write('batch=64' + '\n')
    f.write('subdivisions=24' + '\n')
    f.write('width=416' + '\n')
    f.write('height=416' + '\n')
    f.write('channels=3' + '\n')
    f.write('momentum=0.949' + '\n')
    f.write('decay=0.0005' + '\n')
    f.write('angle=0' + '\n')
    f.write('saturation = 1.5' + '\n')
    f.write('exposure = 1.5' + '\n')
    f.write('hue = .1' + '\n')
    f.write('\n')
    f.write('learning_rate=0.001' + '\n')
    f.write('[burn_in=1000' + '\n']')
    # max_batches = num_classes*2000
    max_batches = 2000
    f.write('max_batches=' + str(max_batches) + '\n')
    f.write('policy=steps' + '\n')
    steps1 = .8 * max_batches
    steps2 = .9 * max_batches
    f.write('steps='+str(steps1)+','+str(steps2) + '\n')

```

Ustawienia wskazują, jak sieć neuronowa będzie zmieniać zdjęcia z każdym cyklem treningowym, nasyceniem, rozmiarami, rotacjami, kątami pochylenia. Wskazane są również parametry czasu trwania cykli i ich liczba, w tym przypadku mamy 2000 cykli.

Po dokonaniu niezbędnych ustawień możesz przejść do rozpoczęcia treningu.

To dość długi proces, trwał 5 godzin.

```

Saving weights to backup//custom-yolov4-detector_last.weights
Resizing, random_coef = 1.40

416 x 416
try to allocate additional workspace_size = 124.60 MB
CUDA allocate done!
Loaded: 0.191554 seconds - performance bottleneck on CPU or Disk HDD/SSD

(next mAP calculation at 1000 iterations)
101: 369.451324, 686.249573 avg loss, 0.000000 rate, 10.918978 seconds, 6060 images, 4.231296 hours 1
Loaded: 0.000050 seconds

(next mAP calculation at 1000 iterations)
102: 360.486023, 653.673218 avg loss, 0.000000 rate, 10.948299 seconds, 6120 images, 4.216729 hours 1
Loaded: 0.000048 seconds

(next mAP calculation at 1000 iterations)
103: 336.907043, 621.996582 avg loss, 0.000000 rate, 10.916450 seconds, 6180 images, 4.201872 hours 1
Loaded: 0.000081 seconds

(next mAP calculation at 1000 iterations)
104: 316.587769, 591.455688 avg loss, 0.000000 rate, 10.913285 seconds, 6240 images, 4.187053 hours 1
Loaded: 0.000047 seconds

(next mAP calculation at 1000 iterations)
105: 302.809753, 562.591064 avg loss, 0.000000 rate, 10.897140 seconds, 6300 images, 4.172345 hours 1
Loaded: 0.000061 seconds

(next mAP calculation at 1000 iterations)
106: 284.697052, 534.801636 avg loss, 0.000000 rate, 10.909470 seconds, 6360 images, 4.157713 hours 1
Loaded: 0.000062 seconds

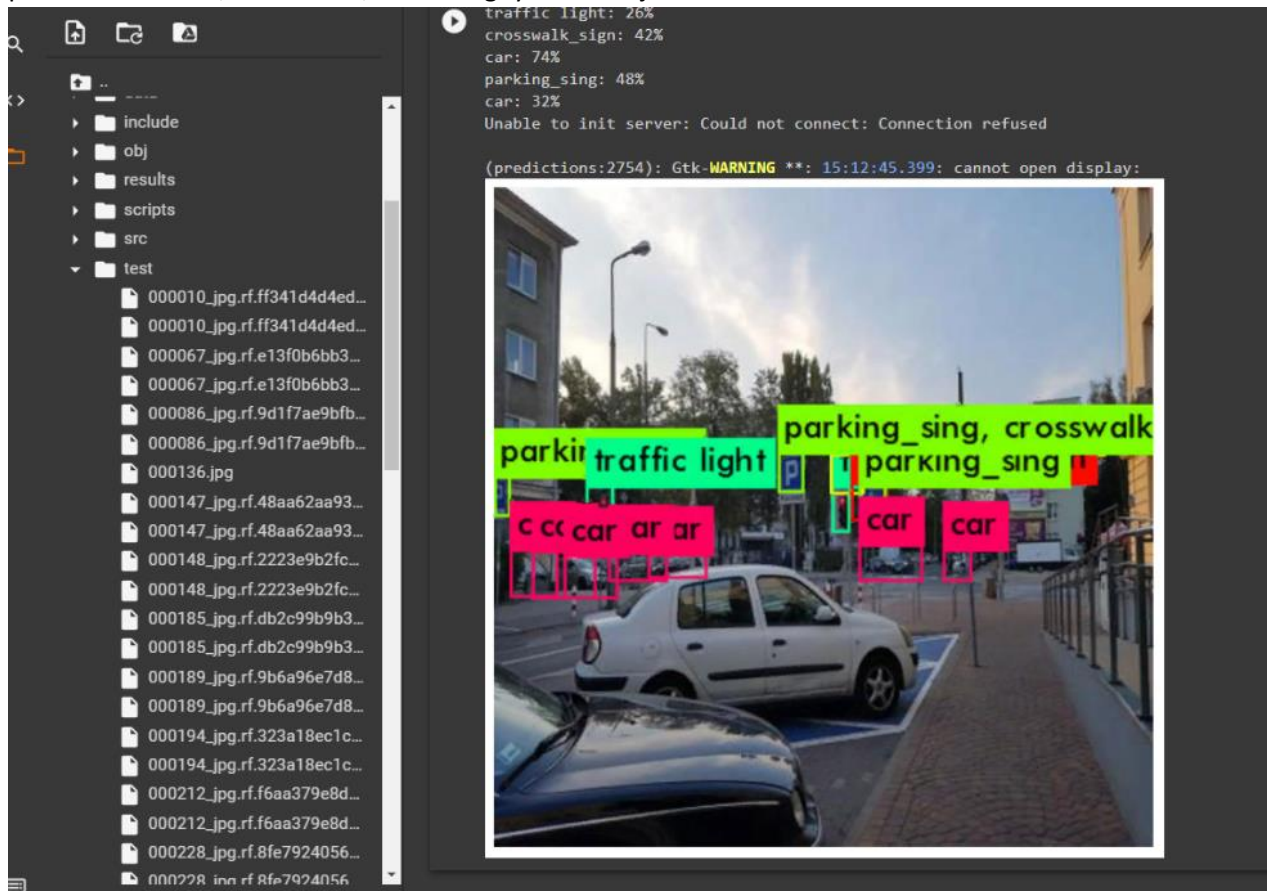
(next mAP calculation at 1000 iterations)
107: 276.521545, 508.973633 avg loss, 0.000000 rate, 10.918530 seconds, 6420 images, 4.143228 hours 1
Loaded: 0.000089 seconds

(next mAP calculation at 1000 iterations)
108: 259.696747, 484.045959 avg loss, 0.000000 rate, 10.889337 seconds, 6480 images, 4.128880 hours 1
Loaded: 0.000039 seconds

```

(proces uczenia sieci ze wskazaniem strat, błędów, czasu trwania 1 cyklu, ilości użytych obrazów)

Po ukończeniu szkolenia sprawdzamy na obrazach testowych, które nie zostały wykorzystane podczas szkolenia, co oznacza, że sieć nigdy wcześniej ich nie widziała.



Po zakończeniu procesu wynikowy plik z wytrenowaną siecią neuronową można pobrać na swój komputer i wykorzystać w dalszych projektach.

Wnioski

W trakcie pracy uzyskano sieć neuronową YOLOv4 wytrenowaną na rzeczywistych obrazach. Jego dokładność w określaniu obiektów jest gorsza od podobnych sieci szkolonych na bardziej produktywnym sprzęcie, ponieważ bardzo trudno było przeprowadzić głębsze szkolenie przy dostępnych możliwościach.

Powstałą sieć neuronową można wykorzystać do identyfikacji obiektów na obrazach i filmach.