

# Raport z przebiegu projektu

---

Kacper Kordalski 20192

## Temat: Etykietowanie zdjęć

### Wymagania projektowe:

1. Zapoznaj się ze strukturą przykładowego pliku JSON zawierającym informacje o oznaczonych obiektach znajdujących się na skojarzonym z plikiem zdjęciu
2. Uruchom skrypt rysujący wielokąt okalający wybrane obiekty. Tak można ocenić rezultat etykietowania zdjęcia.
3. W Twoim projekcie danymi wejściowymi będzie zbiór obrazów. W ramach projektu dla każdego obrazu powinieneś przygotować plik JSON zawierający etykiety obiektów wybranej klasy (np. osoba, znak drogowy, adres budynku itd.)
4. W rezultacie dla zbioru 300 zdjęć należy stworzyć 300 plików JSON, każdy zawierający zbiór współrzędnych - etykietę obiektów danej klasy znajdujących się na konkretnym zdjęciu.
5. Przykładowy udostępniony plik JSON zawiera ustaloną organizację danych. Zauważ, że nazwane są tam klasy obiektów 'shape' - czyli kształt, jako 'label:'person', 'label':'bottle' itd. Przyjmij taki sam schemat w twoich plikach JSON.
6. W przydzielonym zbiorze zdjęć samodzielnie wyodrębnij klasę obiektów, które będą oznaczane - czyli etykietowane.

- W trakcie prób poradzenia sobie z zadaniem projektowym wykorzystaliśmy następujące biblioteki**

**imgaug** – biblioteka wykorzystywana w augmentacji obrazów w nauczaniu maszynowym

**Dużą inspiracją dla nas była wspomniana na wykładzie metoda segmentacji. Przeszukiwaliśmy sieć w celu znalezienia informacji na ten temat, w celu zaimplementowania takiej metody pod nasze potrzeby**

Po okresie poszukiwań udało nam się odnaleźć metodę, która sprawdziła się dla nas bardzo dobrze i nie była aż tak skomplikowana do dostosowania pod nasze potrzeby

I tak zaczęliśmy naszą przygodę z pixellibem, w której naszym celem było sprawienie żeby osiągnął dla nas zmierzony cel

---

**Pierwszym sposobem, z którego skorzystaliśmy była segmentacja semantyczna**

Modyfikując nieco kod źródłowy by wykorzystać pod nasze potrzeby stworzyliśmy skrypt, który przeszukując zdjęcia w danym folderze oznaczał wszystkie obiekty i odznaczał je osobnym kolorem

**Użyty kod:**

```
import os
from os import listdir
from PIL import Image
import pixellib
from pixellib.semantic import semantic_segmentation

folder_dir = "C:/Users/zkaak/Documents/proj/zdjecia"
for filename in os.listdir(folder_dir):
    segment_image = semantic_segmentation()
    segment_image.load_ade20k_model("deeplabv3_xception65_ade20k.h5")

    segment_image.segmentAsAde20k("zdjecia/"+filename, output_image_name =filename, overlay = True)
```

**Użyty przetrenowany model danych:**  
 deeplabv3\_xception\_tf\_dim\_ordering\_tf\_kernels.h5

Deeplabv3 to już 3 wersja architektury służącej do semantycznej segmentacji znalazła swoje zastosowanie w wykrywaniu wielu obiektów, i głównie służyła na przykład do zmiany tła w trybie rzeczywistym czy znajdowania określonych obiektów

Tasks

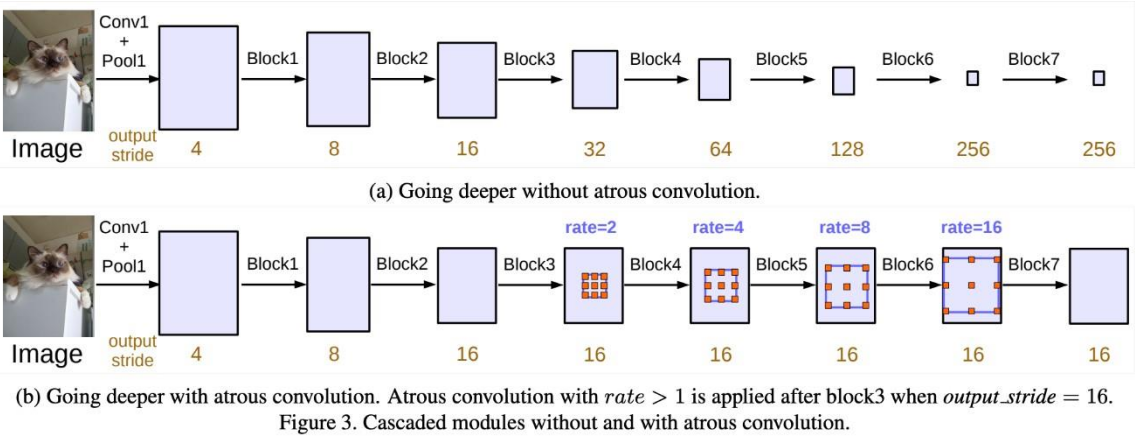
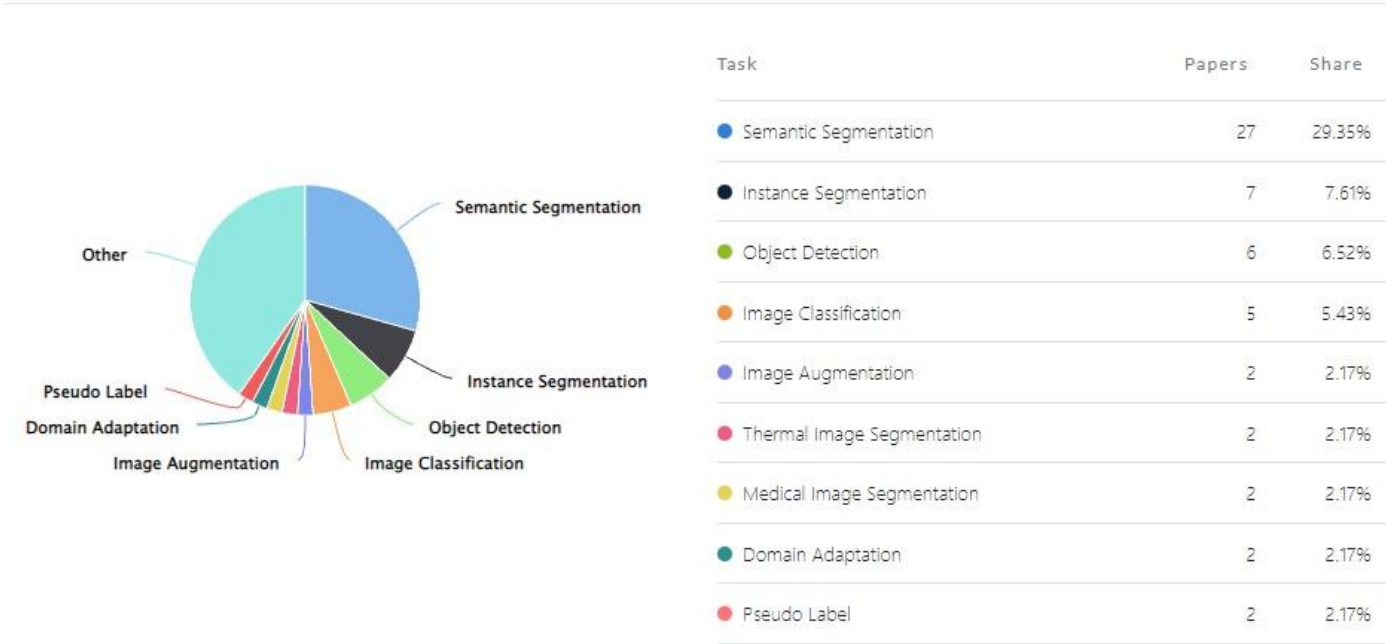


Figure 3. Cascaded modules without and with atrous convolution.

## Otrzymane efekty na zdjęciach testowych:

Program poprawnie odnalazł człowieka oraz oddzielił pozostałe elementy w tle

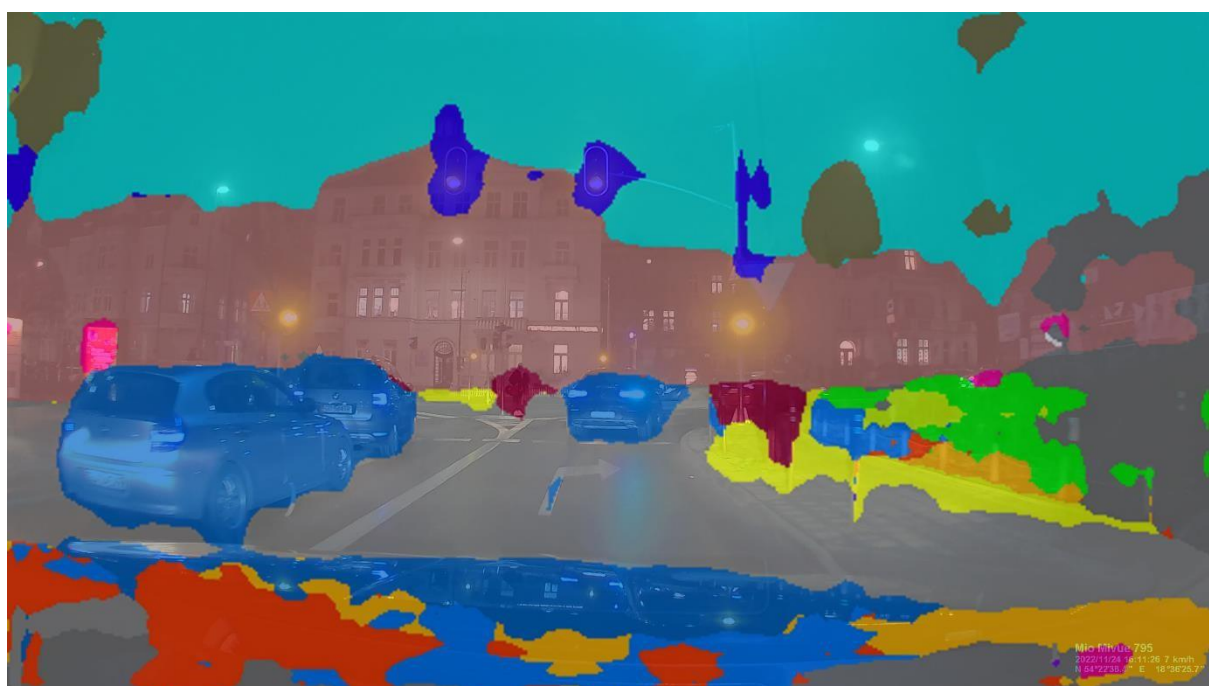


Program odnalazł jedną osobę, ale niestety omyłkowo zaznaczył też kawałek znaku





Na podanym zdjęciu algorytm się całkowicie rozjechał. Wynika to najprawdopodobniej z tego, że zdjęcie zostało wykonane w nocy, co nie było łatwym zadaniem dla programu



Dla pozostałych zdjęć program nie działał aż tak źle, chociaż nie, kiedy miał problemy z zdjęciami, na których ludzie byli w dalekiej odległości



**Ze względu na porażkę algorytmu oraz modelu na dużej ilości zdjęć zrezygnowaliśmy z dalszej modyfikacji semantycznej segmentacji, ponieważ nie dostarczyła nam oczekiwanych efektów**

Na szczęście pixellib oferował również inne sposoby segmentacji, dzięki czemu byliśmy ruszyć dalej z naszym projektem

---

**Wtedy sięgnęliśmy po metodę „Instance segmentation” która okazała się dla nas strzałem w dziesiątkę**

**Zaledwie 5 linijek kodu wystarczyło żeby samemu oznaczyć zdjęcie, co po modyfikacji pozwoliło nam na przetestowanie tego nowego sposobu z nową maską przetrenowaną w celu wykrywania ludzi**

**Użyty kod:**

```

import pixellib
from pixellib.instance import instance_segmentation
import os
from os import listdir
from PIL import Image

folder_dir = "C:/Users/zkaak/Documents/proj/zdjecia"
for filename in os.listdir(folder_dir):

    segment_image = instance_segmentation()
    segment_image.load_model("mask_rcnn_cocov2.h5")
    target_classes = segment_image.select_target_classes(person=True)
    segvoutput = segment_image.segmentImage("zdjecia/"+filename, segment_target_classes= target_classes , mask_points_values=True, output_image_name -filename)

```

**Użyty przetrenowany model danych:** mask\_rcnn\_cocov2.h5

COCO to zestaw danych do wykrywania obiektów na dużą skalę, segmentacji i tworzenia napisów.

COCO oferuje wiele funkcjonalności i jest stale rozwijany poniżej prezentujemy jego najnowsze osiągnięcia i możliwości

- ✓ Object segmentation
- ✓ Recognition in context
- ✓ Superpixel stuff segmentation
- ✓ 330K images (>200K labeled)
- ✓ 1.5 million object instances
- ✓ 80 object categories
- ✓ 91 stuff categories
- ✓ 5 captions per image
- ✓ 250,000 people with keypoints

Do teraz został przetrenowany na ponad 200 tysiącach zdjęć pod kątem poszukiwania poszczególnych obiektów

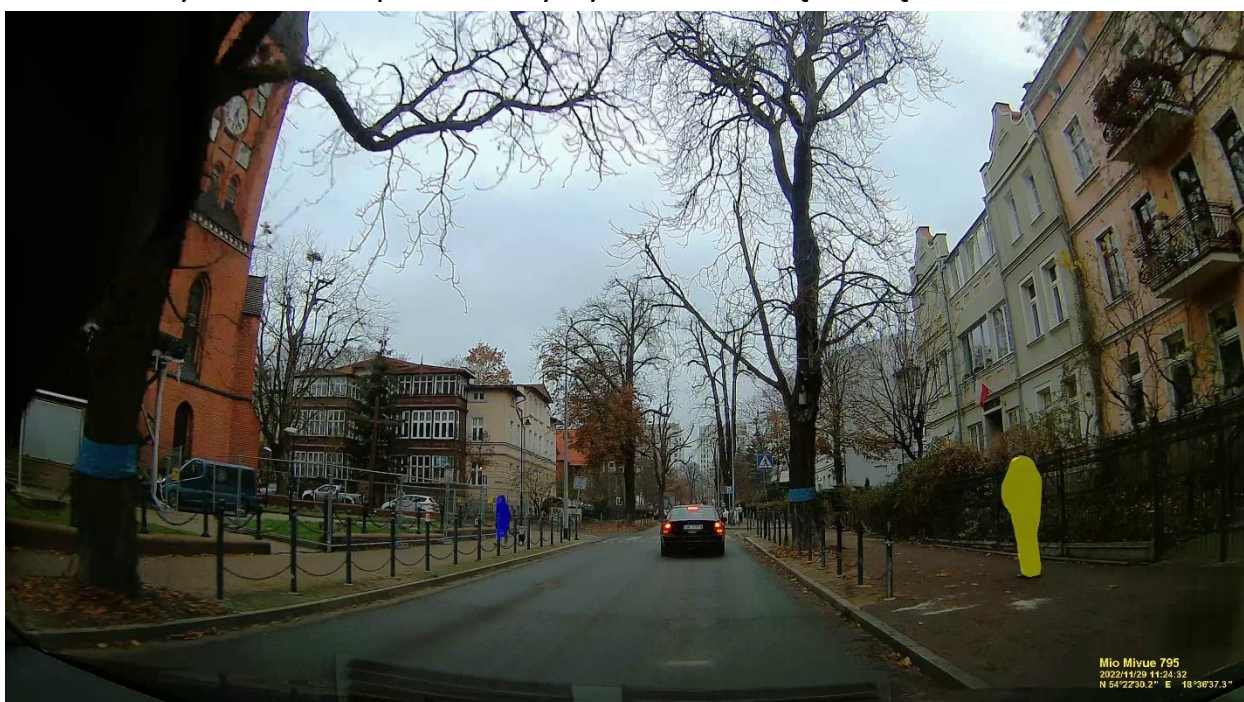
**Otrzymane efekty na zdjęciach testowych:**



Podstawowe zdjęcie testowe nie stanowiło większego wyzwania dla algorytmu nawet pod względem dokładności zaznaczenia

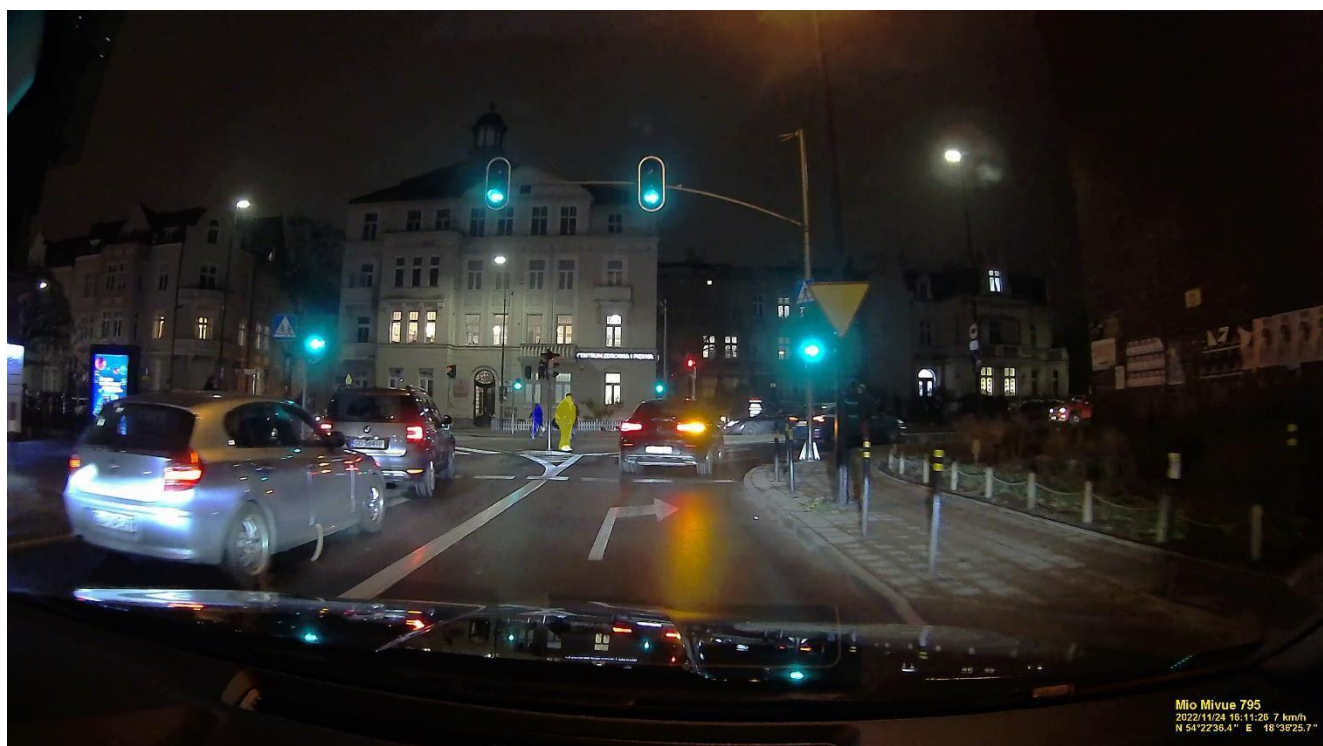


Kolejne zdjęcie testowe również przyniosło lepsze efekty, tym razem nie zaznaczyło drzewa a ponadto wykryło dodatkową osobę w oddali



Kolejne zdjęcie, z którym poprzedni sposób nie poradził sobie kompletnie dla nowego sposobu nie stanowiło większej przeszkody





I tak dla kolejnych zdjęć testowych algorytm przynosił lepsze wyniki niż poprzednia wersja. Była to najprawdopodobniej zasługa nowej maski, która była wielokrotnie trenowana jak i samego sposobu algorytmu.









Powyższy sposób sprostął naszym oczekiwaniom w zadowalającym stopniu na zestawie testowym. Postanowiliśmy, więc edytując ten algorytm dobrać się do siatki punktów zaznaczonych obiektów w celu wygenerowanie potrzebnych jsonów

**Modyfikując kod do działania praktycznie w każdym przypadku otrzymaliśmy następującą wersję:**

```

import pixellib
from pixellib.instance import instance_segmentation
import os
from os import listdir
from PIL import Image

folder_dir = "C:/Users/zkaak/Documents/proj/zdjecia"
for filename in os.listdir(folder_dir):

    segment_image = instance_segmentation()
    segment_image.load_model("mask_rcnn_cocov2.h5")
    target_classes = segment_image.select_target_classes(person=True)
    segvalues, output = segment_image.segmentImage("zdjecia/"+filename, segment_target_classes= target_classes , mask_points_values=True, output_image_name =filename)
    tab=(segvalues['masks'])

    if(len(tab)<=0):
        print("puste")
    else:
        filepath = "zdjecia/"+filename
        img = Image.open(filepath)
        width,height = img.size
        stripped_f = filename.strip(".png")

        with open("json_files/"+stripped_f+".json", 'w') as f:
            f.write('{')
            f.write('\n  "version": "4.0.0",' )
            f.write('\n  "flags": {},')
            f.write('\n  "shapes": [')
            li2=0

            for i in tab:
                lz=0
                f.write('\n    {')
                f.write('\n      "label": "person",' )
                f.write('\n      "points": [')
                for j in i:
                    for x in j:
                        lz=lz+1
                        f.write('\n        [')
                        f.write('\n          '+str(x[0])+',')
                        f.write('\n          '+str(x[1]))
                        if lz==len(j):
                            f.write('\n        ]')
                        else:
                            f.write('\n        ],')
                f.write('\n      ],')
                f.write('\n      "group_id": null,')
                f.write('\n      "shape_type": "polygon",' )
                f.write('\n      "flags": {}')
                li2=li2+1
                if li2==len(tab):
                    f.write('\n    }')
                else:
                    f.write('\n    },')
            f.write('\n  ],')
            f.write('\n  "imagePath": "' +filename+'",' )
            f.write('\n  "imageData": null,')
            f.write('\n  "imageHeight": '+str(height)+',')
            f.write('\n  "imageWidth": '+str(width))
            f.write('\n}')
            f.close()

```

Przykładowo wygenerowany json dla 1 osoby:

```

1 ▼ {
2     "version": "4.0.0",
3     "flags": {},
4     "shapes": [
5     {
6         "label": "person",
7         "points": [
8         [
9             1111,
10            889
11        ],
12        [
13            1110,
14            890
15        ],

```

.....

```

2237            1148,
2238            891
2239        ],
2240 ▼    [
2241            1147,
2242            890
2243        ],
2244 ▼    [
2245            1137,
2246            890
2247        ],
2248 ▼    [
2249            1136,
2250            889
2251        ]
2252    ],
2253    "group_id": null,
2254    "shape_type": "polygon",
2255    "flags": {}
2256    }
2257 ],
2258 "imagePath": "00e8cabf3daf106507644053a7adbd5a.png",
2259 "imageData": null,
2260 "imageHeight": 1600,
2261 "imageWidth": 2848
2262 }

```

Kiedy skrypt był gotowy przyszła pora na przetestowanie otrzymanego wcześniej w materiałach skryptu generującego oznaczenie wokół zdjęcia



**Dokonałiśmy modyfikacji otrzymanego skryptu by oszczędzić pracy ręcznej i nieco zautomatyzować pracę:**

Algorytm przegląda folder ze zdjęciami i jeżeli istnieje dla takiego zdjęcia plik json oraz jeszcze nie ma takiego zdjęcia w folderze z zapisanymi konturowanymi zdjęciami zapisuje go tam

```
from PIL import Image, ImageDraw
import json
import os
from os import listdir
from os.path import exists

def read_points(name):
    f = open('json_files/'+name+'.json')
    print(f)
    data = json.load(f)
    f.close()
    zbior_punktow = []
    for i in data['shapes']:
        punkty = []
        #print(i["Label"])
        for j in i["points"]:
            #print(j)
            punkty.append((j[0], j[1]))
        zbior_punktow.append(punkty)
    return zbior_punktow

def read_img(name):
    img = Image.open("zdjecia_zrobione/"+name+".png")
    w,h = img.size
    print("Format: ", img.format, "Rozmiar: ", img.size, img.mode)
    img1 = ImageDraw.Draw(img)
    xy = read_points(name)
    #img1.polygon(xy, outline="red", fill="#eeeeff")
    for p in xy:
        img1.polygon(p, outline="red")
    img.save("zdjecia_konturowane/"+name+".PNG")

folder_dir = "C:/Users/zkaak/Documents/proj/zdjecia_zrobione"
for filename in os.listdir(folder_dir):
    stripped_f = filename.strip(".png")
    file_exists = os.path.exists("zdjecia_konturowane/"+stripped_f+".PNG")
    #if(file_exists!=True):
    read_img(stripped_f)
```

**Rezultat działania skryptu:**







**Mieliśmy już gotowy skrypt do generowania jsonów oraz konturowanych zdjęć przyszła pora na przeprowadzenie przez niego całego zestawu 301 wcześniej przygotowanych zdjęć w celu sprawdzenia dokładności oraz jakości zastosowanego algorytmu w połączeniu z użytą maską**

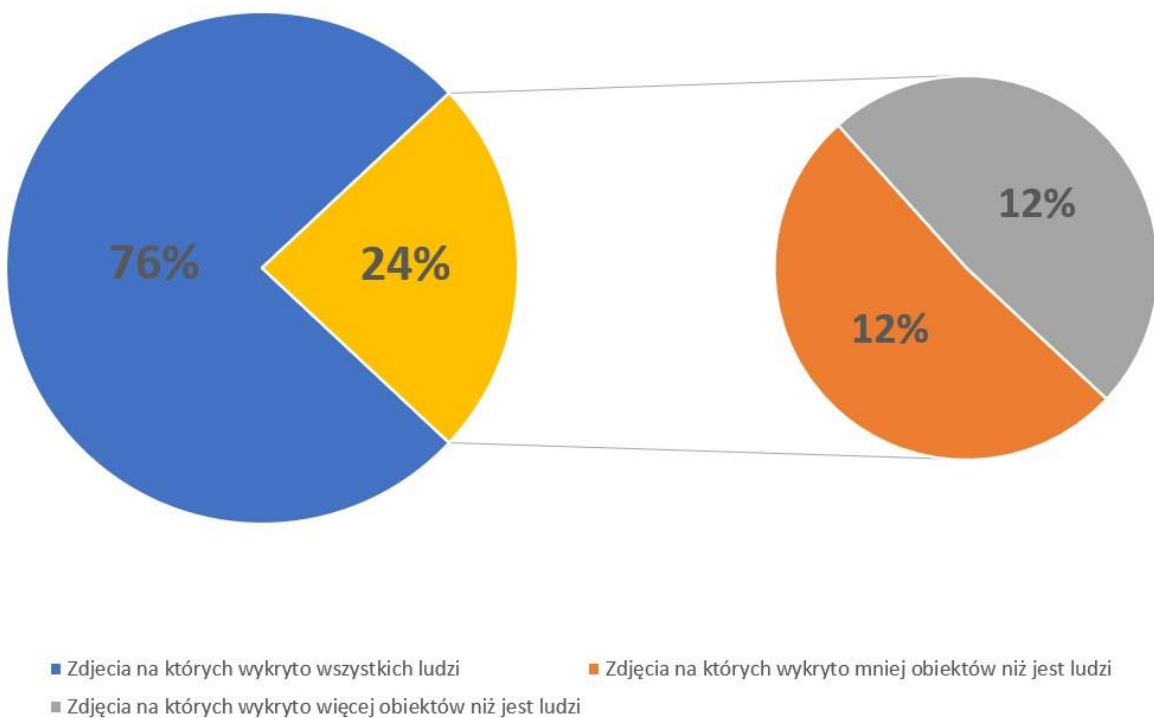
Po przejrzeniu wszystkich wynikowych zdjęć podaliśmy je segregacji w celu wyznaczenia statystyk

Podzieliliśmy je na takie, na których znalazła wszystkie szukane obiekty

Takie, na których znalazło ich za dużo (np. znaki, drzewa, krzewy)

Oraz takie, na których znalazło ich za mało

**Z tego podziału otrzymaliśmy następujące wyniki:**



**Przykładowe zdjęcia, dla których algorytm znalazł wszystkie osoby:**



Jedna osoba w świetle dziennym i nie zbyt dużym ruchu



Matka z córką w oddali stojące w dzień również nie stanowią problemu



Jedna osoba poruszająca się w jednym kierunku w dzień



**Przykładowe zdjęcia, dla których algorytm znalazł dodatkowe obiekty niekoniecznie będące osobą:**

Na zdjęciu po za ludźmi znalazło dodatkowo ciemne krzaki w tle



Algorytm dodatkowo wykrył rozmyty cień na schodach





Algorytm dodatkowo wykrył ciemny obiekt za szybą stacji

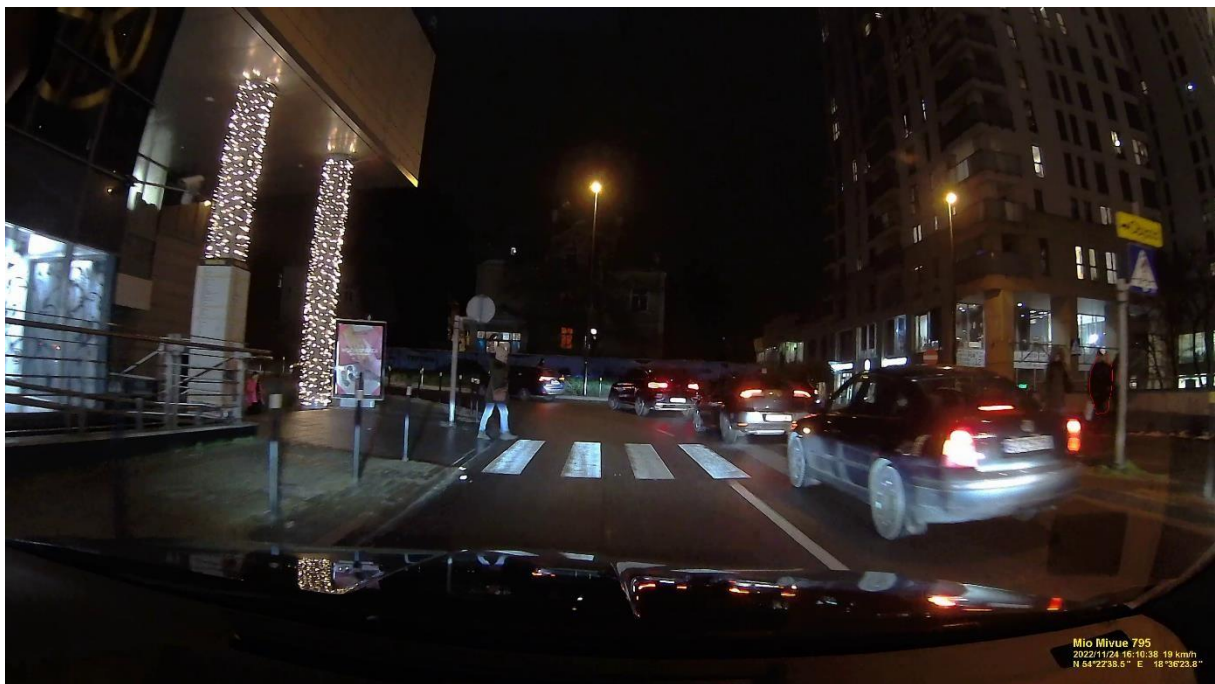


Przykładowe zdjęcia, dla których algorytm nie znalazł wszystkich ludzi:

Jedna z osób w bardzo dalekiej odległości nie została oznaczona



Jedna z osób tuż przed samochodem nie została wykryta



Nie wychwycono osoby trzymającej rower



## PODSUMOWANIE

Podsumowując zastosowany przez nas sposób w ostateczności działa w 100% dokładnie dla 76% zdjęć z bazy wybranych daje to dość dobry wynik. Dla 24 % algorytm nie był w stanie znaleźć wszystkich osób lub też znajdował obiekty niebędące ludźmi. Teraz rozważymy, dlaczego mogło tak być i z czego to wynika

Nasze obserwacje pozwoliły nam zwrócić uwagę na kilka podstawowych czynników wpływających efekty działania algorytmu:

- Po pierwsze i najważniejsze **poza, w jakiej** znajduje się osoba. Algorytm ma z tym nie lada problem, ponieważ człowiek może znajdować się w najróżniejszych pozycjach i przyjmować przeróżne kształty w przeciwieństwie do aut, okien, sygnalizacji, które zazwyczaj są w kilku określonych kształtach.
- Drugim ważnym czynnikiem jest **ruch** ludzie na ulicy zwykle są w ruchu, co często powoduje ich rozmycie na zdjęciach. Prowadzi to do nie wychwycenia ich przez algorytm

- Kolejnym istotnym elementem jest **pora dnia** dla przykładu, kiedy jest jasno ludzie się wyróżniają i dość łatwo jest ich oznaczyć a kiedy jest ciemno ludzie zlewają się z otoczeniem, przez co ciężko jest ich oznaczyć czy odróżnić od otoczenia, przez co algorytm się nie, kiedy myli i oznacza na przykład krzewy czy ciemne cienie
- **Odległość** również wpływa na to czy osoba zostanie wykryta dla przykładu na jednym zdjęciu ten sam człowiek jest wykrywany a gdy odejdzie trochę dalej już nie
- **Zastłonięcie** to znaczy, kiedy osoba nie jest przesłonięta na przykład przez barierkę, samochód czy inną osobę to zostaje w większości przypadków oznaczona prawidłowo, gdy jednak zostanie przesłonięta jest oznaczenie wtedy się zdeformuje w niektórych przypadkach

Podsumowując osiągnęliśmy zamierzony cel i otagowaliśmy wybrane zdjęcia w większości przypadków 100% poprawnie. Co do dokładności zaznaczenia to ma ona margines błędu do kilku milimetrów wynika to głównie z wyżej wymienionych czynników.

Zadanie, które sobie obraliśmy nie należało do najłatwiejszych, ponieważ ludzie nie są najłatwiejszym obiektem do otagowania pomimo tego, że wiele osób czy nawet firm od lat stara się znaleźć idealny sposób na ich oznaczanie, to do dzisiaj nie został on wymyślony, a każdy z dotąd wymyślonych ma swoje plusy i minusy.

Nie należy się jednak podawać, ponieważ cały czas powstają to nowe sposoby, które są rozwijane a także doszkalane są maski, które te wyszukiwanie usprawniają. Chociaż sami nie zdecydowaliśmy się na doszkalanie naszej maski (pomimo istnienia takiej możliwości), ponieważ została ona już wcześniej przeszkolona na setkach tysięcy zdjęć pod kątem znajdowania ludzi, co pozwoliło nam już na niej uzyskać zadowalający efekt a nasze 300 zdjęć nie wiele by do niej wniosło.

Zamiast tego skorzystaliśmy z innych wersji już przeszkolonych wersji wcześniej wspomnianych masek, które umieszczamy dodatkowo w plikach projektowych



Do projektu dołączamy:

- Folder z 10 zdjęciami testowymi(zdjecia\_oznaczone)
- Folder z semantycznym segmentowaniem tych 10 zdjęć(zdjecia\_segmanctic\_deeplab)
- Folder z segmentowaniem instancji tych 10 zdjęć (zdjecia\_oznaczone)
- Folder z bazą 301 wybranych zdjęć (zdjecia\_zrobione)
- Folder z 301 json'ami (json\_files)
- Folder z 301 okonturowanymi zdjęciami (zdjecia\_konturowane)
- Oznaczenie.ipynb – programy oznaczające zdjęcia na podstawie wygenerowanych json'ów
- Projekt.ipynb – programy generujące zdjęcia i pliki json przy pomocy pixelliba
- Różne maski użyte w celu wyłonienia tej najlepszej pod nasz potrzeby

## Bonusowy materiał

Na zakończenie chcieliśmy zaprezentować, że dzisiejsze badania systemów wykrywania ludzi w tak wielkiej firmie, jaką jest TESLA również nie zawsze w 100% dają oczekiwany skutek. Poniżej materiał z nie tak dawnych testów autopilota:

[https://www.tiktok.com/@realdanodowd/video/7187030300667464966?is\\_from\\_webapp=1&sender\\_device=pc&web\\_id=7133619734554510854](https://www.tiktok.com/@realdanodowd/video/7187030300667464966?is_from_webapp=1&sender_device=pc&web_id=7133619734554510854)