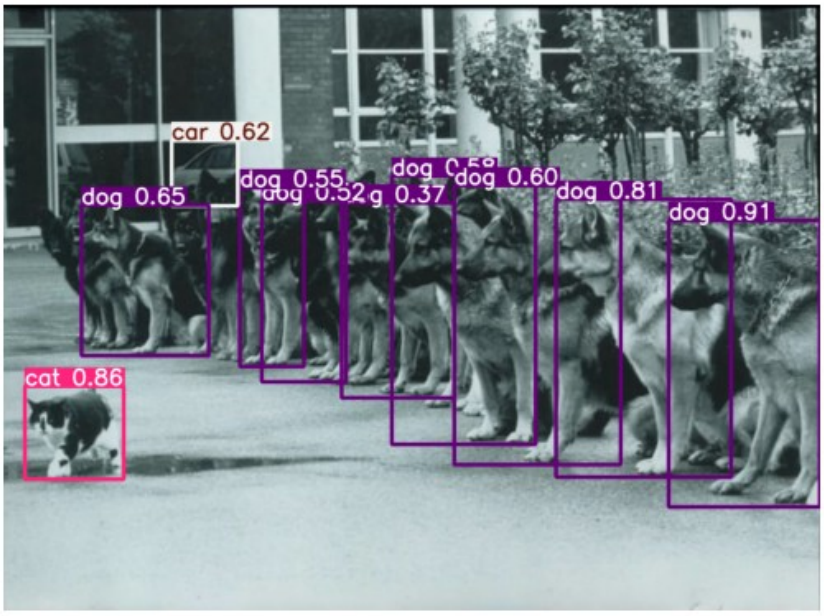
Metody Rozpoznawania Obrazów
Dominik Szot
Zadanie 04

1. Korzystanie z YOLO

Pierwszym zadaniem w tym laboratorium było przetestowanie gotowego modelu YOLO do detekcji obiektów. Testowanym modelem YOLO jest **yolo10m**, który jest znacznie lepszy w detekcji od swoich poprzedników.

Obraz 1: Wybrane zdjęcie psów i kota



Obraz 2: Predykcje modelu yolo10m

2. Przystosowanie YOLO do wykrywania jedzenia

W celu przystosowania modelu to wykrywania jedzenia wprowadzamy dodatkowe trenowanie. Zaczynamy od przgotowania danych treningowych, testowych I walidacyjnych.
Piewszym krokiem będzie przygotowanie pliku tekstowego którego będzie można użyć do pobrania zdjęć przy użyciu skryptu.

Generowanie listy obrazów

```
import pandas as pd
import os

def create_image_list(
        dataset_image_ids_path: str,        # Path to the CSV file with image IDs
        data_type: str,                     # Type of dataset ('test', 'train', or 'validation')
        class_name: str                     # Class name 'Food'
):

        # Columns of dataset
        column_names = ['ImageID','Source','LabelName','Confidence']
        class_descriptions = pd.read_csv(dataset_image_ids_path, names
            =   column_names, header=None, low_memory=False)

        # Filter collumns with given label name
        filtered_images = class_descriptions[class_descriptions['LabelName'] == class_name]
        num_images_needed = DATASET_SPLIT.get(data_type, 0)

        # Get ID's of extracted rows
        filtered_images_ids = filtered_images['ImageID'].unique()[:num_images_needed]
        DATASET_IDS[data_type] = filtered_images_ids

        # Create list-file
        path = './image-lists/' + data_type + '_image.list.txt'

        with open(path, 'w') as f:
                for image_id in filtered_images_ids:
                        f.write(f'{data_type}/{image_id}\n')

        print(f"{data_type} image list created! No. entities '{len(filtered_images_ids)}'")
```

Oprócz tego potrzebne będą jeszcze odpowiednie etykiety do prawidłowego stworzenia datasetu

```
def create_labels_list(
        dataset_bbox_path: str,
        dataset: str,
        data_type: str,
        class_name: str,
        DATASET_IDS: dict
):
        # Columns to use
        use_columns = ['ImageID', 'LabelName', 'XMin', 'XMax', 'YMin', 'YMax']
        directory = f"./{dataset}/labels/{data_type}/"
        os.makedirs(directory, exist_ok=True)

        df = pd.read_csv(
                dataset_bbox_path,
                header=0,
                usecols=use_columns,
                engine='python',
                on_bad_lines='skip'
        )

        mask = (
                (df['LabelName'] == class_name) &
                (df['ImageID'].isin(DATASET_IDS[data_type]))
        )

        filtered_df = df.loc[mask].copy()
        for col in ['XMin', 'XMax', 'YMin', 'YMax']:
                filtered_df[col] = pd.to_numeric(filtered_df[col], errors='coerce')

        filtered_df['width'] = filtered_df['XMax'] - filtered_df['XMin']
        filtered_df['height'] = filtered_df['YMax'] - filtered_df['YMin']
        filtered_df['center_x'] = (filtered_df['XMin'] + filtered_df['XMax']) / 2
        filtered_df['center_y'] = (filtered_df['YMin'] + filtered_df['YMax']) / 2

        for _, row in filtered_df.iterrows():
                file_path = os.path.join(directory, f"{row['ImageID']}.txt")
                with open(file_path, 'w') as f:
                        f.write(f"0 {row['center_x']} {row['center_y']} {row['width']} {row['height']}\n")

total_labels = len(filtered_df)
print(f"{data_type} bounding boxes created. No. entities '{total_labels}'")
```

Posiadając przygotowane listy dla każdego zbioru, oraz przygotowane etykiety możemy użyć skryptu do pobrania zbioru danych.

W tym zadaniu wybrano dla zbiorów następujące rozmiary:

- Testowy: 2000 zdjęć
- Treningowy: 10000 zdjęć
- Walidacyjny: 2000 zdjęć

Liczba etykiet różniła się od liczby zdjęć I wyniosła

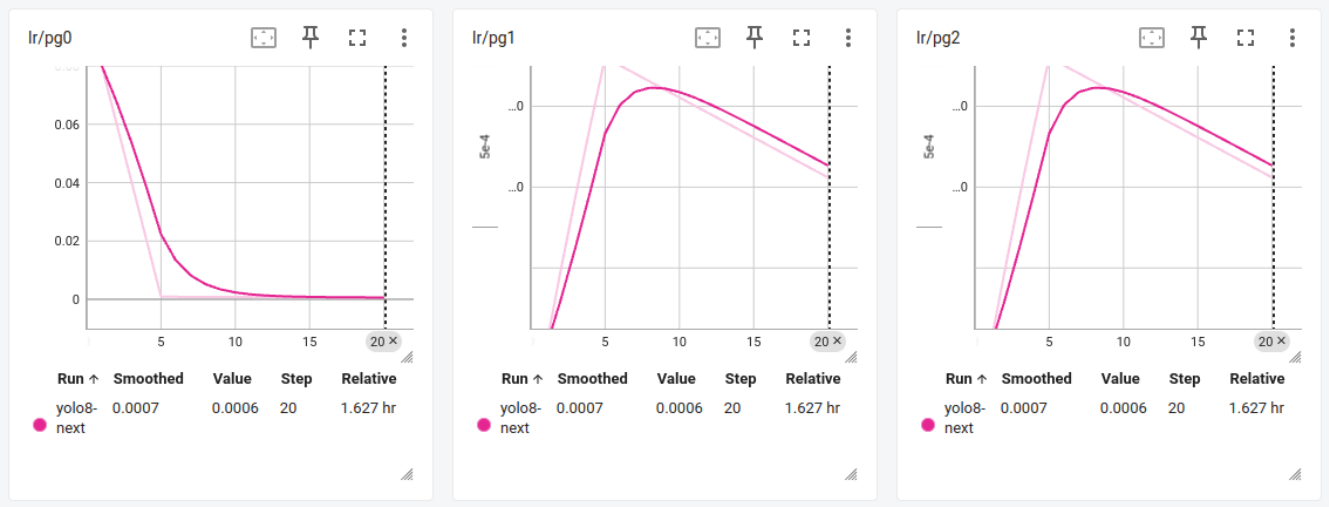
- 1413 etykiet dla zbioru testowego
- 4443 etykiet dla zbioru treningowego
- 1581 dla zbioru walidacyjnego

W kolejnym etapie należało przygotować plik .yaml dla YOLO z opisem zbioru danych.
Plik zawiera ścieżki do odpowiednich katalogów I można go stworzyć ręcznie.

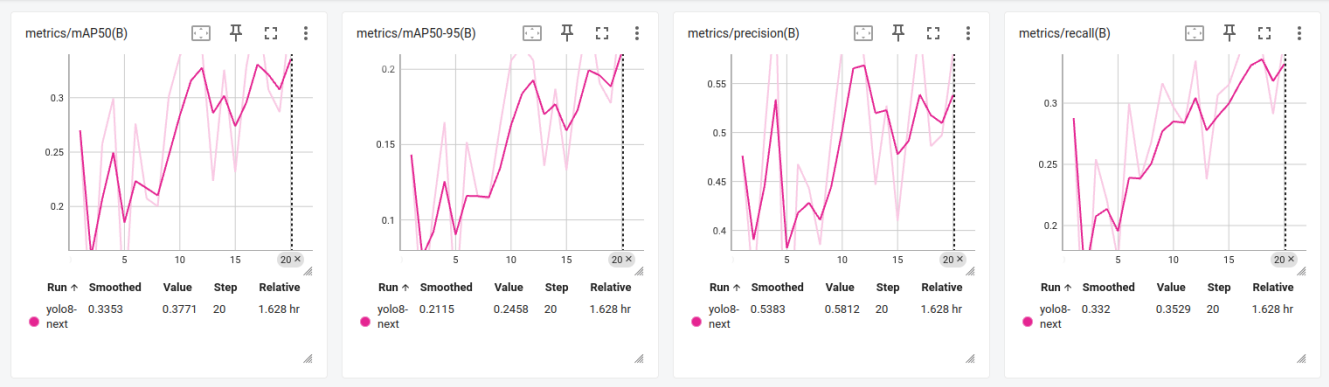
```
path: /content/dataset/
train: images/train
validation: images/val
test: images/test

names:
  0: food
```

Należało też zadbać o odpowiednią hierarchię obiektów w zbiorze danych do odpowiedniego przeprowadzenia treningu.

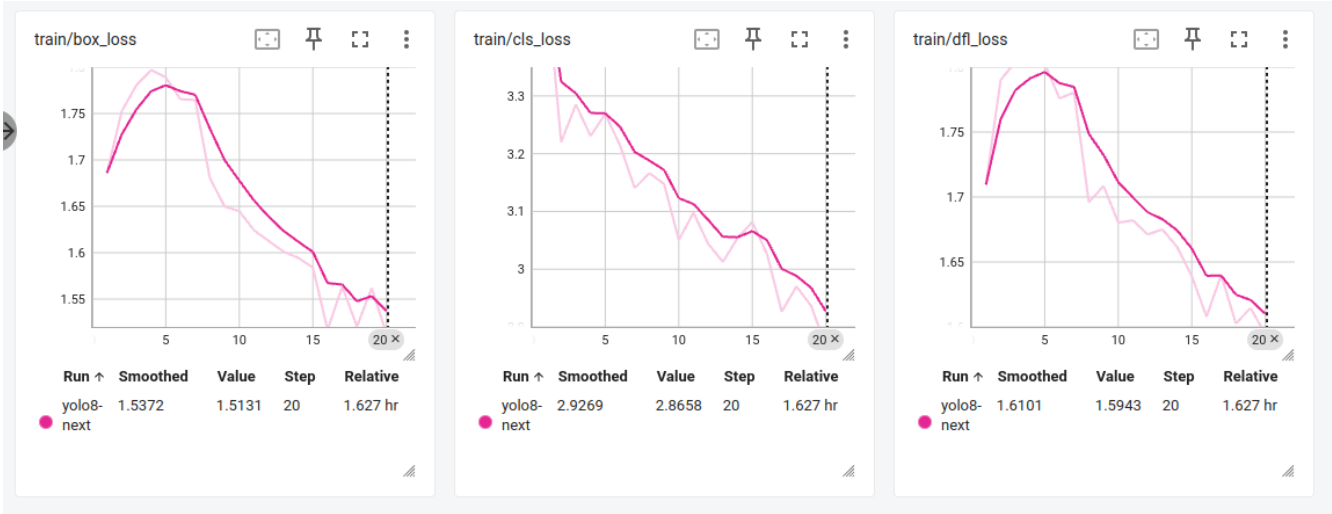
Przebieg treningu monitorowano przy użyciu Tensorboard’a



Obraz 3: Wykresy LR



Obraz 4: Wykresy metryk

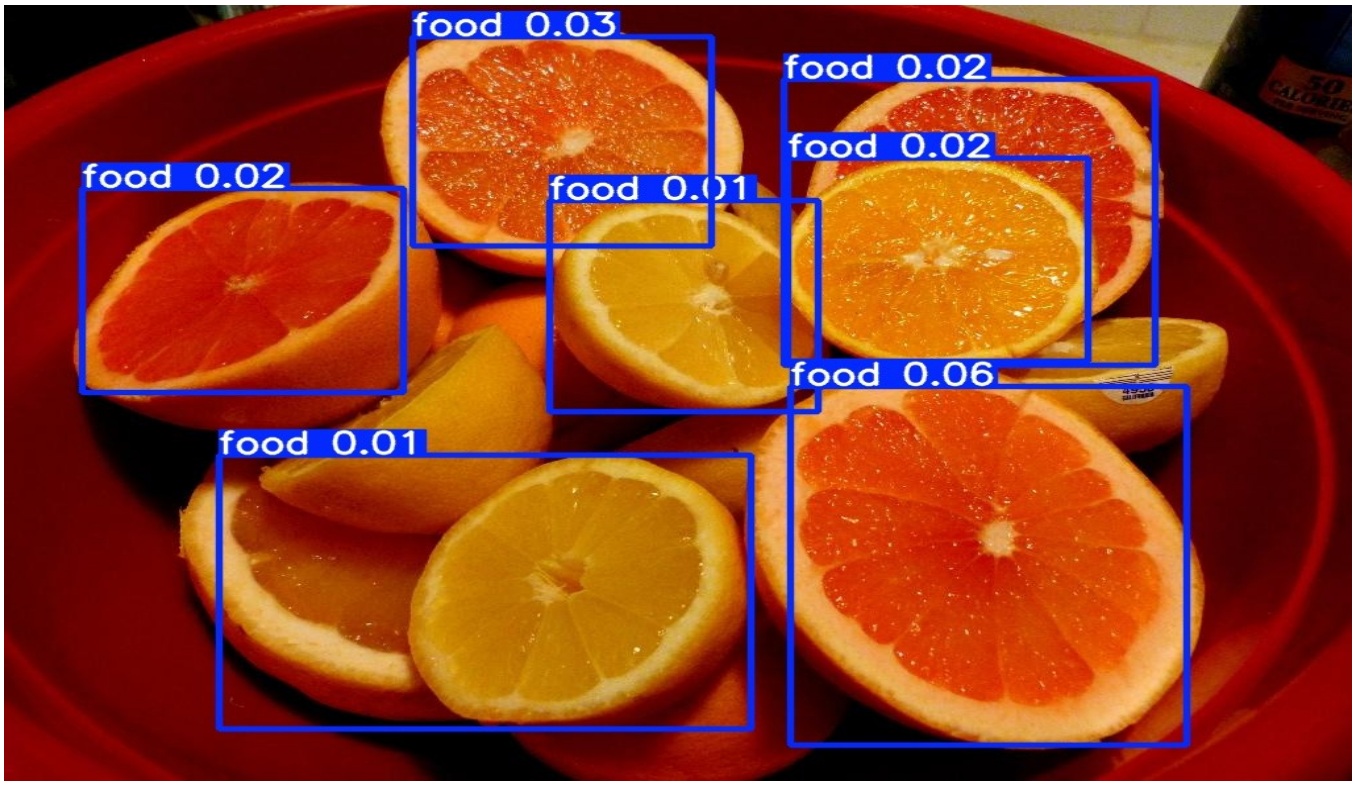


Obraz 5: Wykresy funkcji kosztu

3. Problemy

Po skończeniu treningu zauważyłem, że model nie przewiduje zbyt dobrze obiektów na obrazach. Mogło to być spowodowane niedostatecznym treningiem, zbyt dużą ilością danych treningowych lub też innymi, nieznanymi mi problemami. Z tego też powodu do detekcji obiektów w kolenych punktach ręcznie zmieniono próg detekcji *confidence threshold,* żeby też nie pozostać z pustym raportem.


food 0.05


food 0.03
food 0.02
food 0.02
food 0.02
food 0.01
food 0.06
food 0.01

W celu uzyskania efektu rozmycia zastosowałem prosty algorytm rozmycia gausowskiego [1]z bibiloteki OpenCV.

```
def blur_image(
        model: YOLO,
        image: str,
        conf: int
):
        results = model.predict(image, conf=conf)

        image_arr = cv2.imread(image)
        for result in results:
                for box in result.boxes:
                        x1, y1, x2, y2 = map(int, box.xyxy[0])

                        roi = image_arr[y1:y2, x1:x2]
                        image_arr[y1:y2, x1:x2] = cv2.GaussianBlur(roi, (99, 99), 0)

        filename = image.rsplit('.', 1)[0]
        output_path = f"{filename}_blurred.jpg"

cv2.imwrite(output_path, image_arr)
display(Image(output_path))
```

Obraz 6: Wynik rozmycia wykrytego obiektu

1 https://docs.opencv.org/4.x/d4/d86/group__imgproc__filter.html#gae8bdcd9154ed5ca3cbc1766d960f45c1

Obraz 7: Wynik rozmycia wykrytego obiektu

Obraz 8: Wynik rozmycia wykrytego obiektu