

# Metody Rozpoznawania Obrazów

## Zadanie 04 - [Oczenurowano]

Wojciech Michaluk

12.11.2025

### Opis kolejnych etapów zadania

Do pierwszej części zadania wykorzystuję model `yolov8m.pt`. Z kolei później, w fazie *fine-tuningu* wykorzystam mniejszy model `yolov8n.pt`. Pobierając odpowiednie pliki ze strony HuggingFace (tutaj przykładowo dla większego modelu, dla mniejszego analogicznie), mogę je łatwo wczytać z wykorzystaniem biblioteki `ultralytics`:

```
from ultralytics import YOLO
```

```
model = YOLO('..../models/yolov8m.pt')
```

Listing 1: Wczytanie modelu z pobranego pliku

Zgodnie z poleceniem zadania sprawdzam działanie wczytanego modelu na obrazie zawierającym psa i kota obok siebie:



Rysunek 1: Wybrany oryginalny obraz (a) oraz obraz z oznaczeniami wykrytych obiektów (b) - klasa i pewność

Wydaje się, że model działa jak należy. W głównej części zadania korzystam z otwartego zbioru danych **Open Images Dataset V7**. Wykorzystuję bezpośrednio 2 pliki .csv z tej strony (źródła):

1. Annotacje i bounding boxy dla zbioru treningowego
2. Annotacje i bounding boxy dla zbioru walidacyjnego

Z kolei informację o odpowiedniej etykiecie dla *ogólnej* klasy "jedzenie" (bowiem w używanym zbiorze występuje hierarchia klas, a nas interesuje ta najmniej szczegółowa) mam pośrednio, poprzez plik z opisem klas. Ta etykieta to `/m/02wbm`.

Na początku wczytuję wspomniane pliki w postaci DataFrame'ów z wykorzystaniem biblioteki pandas:

```
train_csv_df = pd.read_csv("./csvs/oidv6-train-annotations-bbox.csv")
val_csv_df = pd.read_csv("./csvs/validation-annotations-bbox.csv")

# etykieta klasy "jedzenie"
food_label = "m/02wbm"
```

Listing 2: Wczytanie plików i ustawienie etykiety jedzenia

Skrypt `downloader.py` (źródło) służący do pobierania odpowiednich plików (obrazów) wymaga informacji o *splice* - w tym przypadku jest to **train** dla obrazów treningowych i **validation** dla obrazów walidacyjnych oraz identyfikatorów obrazów. Przygotowane DataFrame'y zawierają taką informację. Najpierw dokonuję filtrowania, żeby pozostawić jedynie wiersze odpowiadające obrazom z etykietą "jedzenie":

```
food_train_df = train_csv_df[train_csv_df['LabelName'] == food_label]
food_val_df = val_csv_df[val_csv_df['LabelName'] == food_label]
```

Listing 3: Filtrowanie DataFrame'ów

Jako że w obecnym kształcie wśród identyfikatorów obrazów mogą występować duplikaty, tworzę ich *zbiory* (`set` w Pythonie).

```
train_images_ids_set = set(food_train_df['ImageID'])
val_images_ids_set = set(food_val_df['ImageID'])
```

Listing 4: Przygotowanie zbiorów identyfikatorów

Okazuje się, że unikalnych obrazów jest dokładnie 25712 w zbiorze treningowym oraz 1696 w zbiorze walidacyjnym. Ze względów czasowych oraz zajętości miejsca na dysku, postanawiam ograniczyć nieco liczbę pobieranych obrazów. W tym celu wybieram losowo 5000 obrazów treningowych i 500 obrazów walidacyjnych do pobrania. W przypadku kolejnego uruchamiania programu nie wykonuję tego procesu ponownie.

```
# zapiszmy identyfikatory obrazów do pliku
train_output_filename = "./images_ids/train_images_ids.txt"
val_output_filename = "./images_ids/val_images_ids.txt"

if not os.path.exists(train_output_filename):
    # ograniczmy nieco liczbę obrazów do zapisania i pobrania
    train_images_ids = sample(sorted(train_images_ids_set), k=5000)

    with open(train_output_filename, "w") as f:
        for image_id in train_images_ids:
            f.write(f"train/{image_id}\n")
else:
    print("Plik z ID obrazów treningowych już istnieje!")

# pobierzmy listę identyfikatorów z pliku
with open(train_output_filename, "r") as f:
    lines = f.readlines()
    train_images_ids = []

    for line in lines:
        train_images_ids.append(line[6:]) # pominiecie przedrostka "train/"

if not os.path.exists(val_output_filename):
    # ograniczmy nieco liczbę obrazów do zapisania i pobrania
    val_images_ids = sample(sorted(val_images_ids_set), k=500)

    with open(val_output_filename, "w") as f:
        for image_id in val_images_ids:
            f.write(f"validation/{image_id}\n")
```

```

else:
    print("Plik z ID obrazów walidacyjnych już istnieje!")

# pobierzmy listę identyfikatorów z pliku
with open(val_output_filename, "r") as f:
    lines = f.readlines()
    val_images_ids = []

for line in lines:
    val_images_ids.append(line[11:]) # pominięcie przedrostka "validation/"

```

Listing 5: Przygotowanie plików z ID obrazów do pobrania

Mając przygotowane odpowiednie pliki z ID obrazów, korzystam ze skryptu i pobieram obrazy (zachowując wymaganą strukturę plików do późniejszego treningu sieci):

```

# wskazuję ścieżki do zapisu obrazów treningowych/walidacyjnych
os.makedirs("./dataset/train", exist_ok=True)
os.makedirs("./dataset/val", exist_ok=True)
train_images_directory = "./dataset/train/images"
val_images_directory = "./dataset/val/images"

# uruchamiam skrypt 'downloader.py' z odpowiednimi argumentami
if not os.path.exists(train_images_directory):
    os.mkdir(train_images_directory)
    os.system(f"py_downloader.py {train_output_filename} --download_folder=\
              {train_images_directory}")
else:
    print("Obrazy do treningu już zostały pobrane!")

if not os.path.exists(val_images_directory):
    os.mkdir(val_images_directory)
    os.system(f"py_downloader.py {val_output_filename} --download_folder=\
              {val_images_directory}")
else:
    print("Obrazy do walidacji już zostały pobrane!")

```

Listing 6: Pobieranie obrazów treningowych i walidacyjnych

Teraz przystępuję do stworzenia pliku .yaml, potrzebnego do treningu sieci. Po wyszukaniu informacji w Internecie, co taki plik powinien zawierać, postanawiam stworzyć go ręcznie - w naszym przypadku jest on bardzo prosty. Poniżej przedstawiam go w pełnej okazałości:

```

# Ścieżki do zbiorów obrazów
train: dataset/train/images
val: dataset/val/images

```

```

# Nazwy klas
names: [ 'food' ]

```

```

# Liczba klas
nc: 1

```

Listing 7: Plik dataset.yaml do treningu sieci

Ostatnią rzeczą, którą trzeba zrobić przed rozpoczęciem treningu, jest przygotowanie plików tekstowych dla każdego obrazu, zawierających informacje o bounding boxach. Na szczęście wszystkie wymagane dane są zawarte we wcześniej przygotowanych DataFrame'ach. Należy je tylko odpowiednio przetworzyć, żeby zgadzały się z formatem wymaganym do treningu YOLO:

```
<nr klasy> <wsp. x środka bboxa> <wsp. y środka bboxa> <szerokość bboxa> <wysokość bboxa>
```

Realizuję to następująco:

```
# tworzę DataFrame'y z wartościami używanymi w formacie YOLO
train_bbs_df = food_train_df.copy()
val_bbs_df = food_val_df.copy()

# format YOLO wymaga współrzędnych środka oraz wymiarów bounding boxa
train_bbs_df['x_center'] = (food_train_df['XMin'] + food_train_df['XMax']) / 2
train_bbs_df['y_center'] = (food_train_df['YMin'] + food_train_df['YMax']) / 2
train_bbs_df['width'] = food_train_df['XMax'] - food_train_df['XMin']
train_bbs_df['height'] = food_train_df['YMax'] - food_train_df['YMin']

val_bbs_df['x_center'] = (food_val_df['XMin'] + food_val_df['XMax']) / 2
val_bbs_df['y_center'] = (food_val_df['YMin'] + food_val_df['YMax']) / 2
val_bbs_df['width'] = food_val_df['XMax'] - food_val_df['XMin']
val_bbs_df['height'] = food_val_df['YMax'] - food_val_df['YMin']

# wskazuję ścieżki do zapisu informacji o bounding boxach w formacie YOLO
train_labels_directory = "./dataset/train/labels"
val_labels_directory = "./dataset/val/labels"

if not os.path.exists(train_labels_directory):
    os.mkdir(train_labels_directory)

    for image_id in train_images_ids:
        image_bbs = train_bbs_df[train_bbs_df['ImageID'] == image_id]

        with open(f"{train_labels_directory}/{image_id}.txt", "w") as f:
            for _, bb in image_bbs.iterrows():
                f.write(f"0,{bb['x_center']},{bb['y_center']} +\n"
                       f",{bb['width']},{bb['height']}\n")

else:
    print("Pliki z bounding boxami dla obrazów treningowych już zostały stworzone!")

if not os.path.exists(val_labels_directory):
    os.mkdir(val_labels_directory)

    for image_id in val_images_ids:
        image_bbs = val_bbs_df[val_bbs_df['ImageID'] == image_id]

        with open(f"{val_labels_directory}/{image_id}.txt", "w") as f:
            for _, bb in image_bbs.iterrows():
                f.write(f"0,{bb['x_center']},{bb['y_center']} +\n"
                       f",{bb['width']},{bb['height']}\n")

else:
    print("Pliki z bounding boxami dla obrazów walidacyjnych już zostały stworzone!")
```

Listing 8: Zapis informacji o bounding boxach dla wybranych obrazów

Warto zauważyć, że jako numer klasy zawsze podaję 0, gdyż mamy jedną klasę (jedzenie), a indeksowanie jest od 0.

Wreszcie mogę przystąpić do treningu:

```
# po treningu zapisuję najlepszy model, żeby nie powtarzać procedury
trained_model_path = "./models/yolov8n_finetuned.pt"

if not os.path.exists(trained_model_path):
    # ładuję mniejszy model z pobranego pliku
    model = YOLO("./models/yolov8n.pt")

    # trening modelu przez 5 epok
    results = model.train(
        data="./dataset.yaml",
        epochs=5,
        imgsz=640,
        batch=16,
        workers=8,
        verbose=False,
        name="custom_yolo_run"
    )

    # ścieżka do najlepszych wag modelu
    best_model_path = "./runs/detect/custom_yolo_run/weights/best.pt"

    # wczytanie wytrenowanego modelu
    model = YOLO(best_model_path)

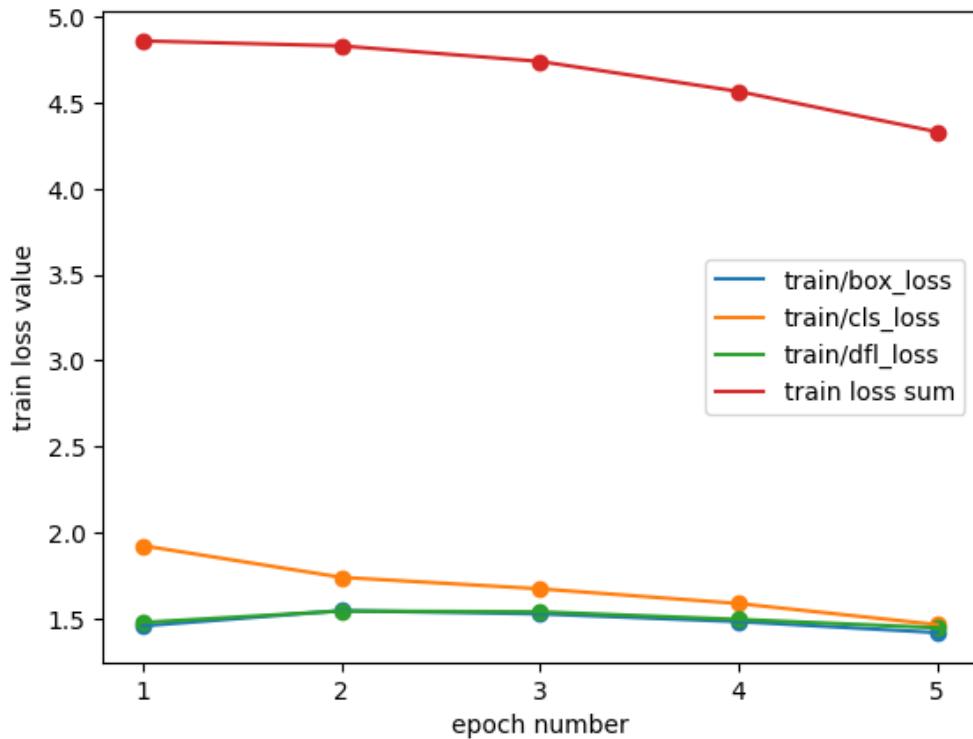
    # zapisanie modelu, co może przydać się później
    model.save(trained_model_path)
else:
    print(f"Model już został wytrenowany!\nWczytuje najlepszy model z {trained_model_path}")

model = YOLO(trained_model_path)
```

Listing 9: Trening mniejszego modelu z wykorzystaniem pobranych obrazów

Parametry treningu dobieram tak, żeby trening wykonał się optymalnie na moim sprzęcie. Podczas treningu zapisywane jest wiele istotnych informacji, między innymi plik .csv ze szczegółowymi informacjami dotyczącymi wartości funkcji kosztu czy wagi modelu z ostatniej epoki treningu oraz najlepszego (w rozumieniu metryki *mAP50-95*), które wykorzystuję przy następnych uruchomieniach programu - zapisuję je po wykonaniu treningu, żeby nie powtarzać go za każdym razem (u mnie całość trwa około 2 godzin).

Podczas treningu zapisywane są także "gotowe" wykresy różnych metryk - ze względu na czytelność samodzielnie skonstruuję wykres z tylko interesującymi nas wartościami funkcji kosztu.



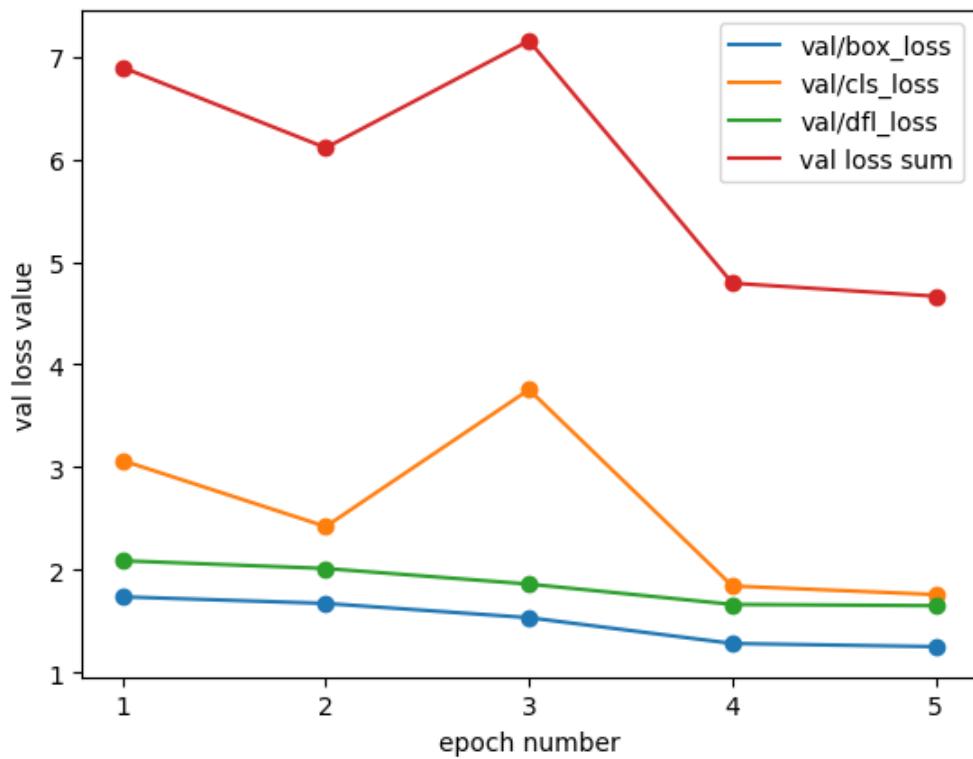
Rysunek 2: Wykres funkcji kosztu dla danych treningowych

Należy zwrócić uwagę, że na funkcję kosztu składają się 3 składowe:

1. **box\_loss**, czyli strata związana z dokładnością predykcji *bounding boxów*,
2. **cls\_loss**, czyli strata związana z klasyfikacją (ciekawa kwestia dla jednej klasy),
3. **dfl\_loss**, czyli tzw. *Distribution Focal Loss* - Strata Ogniskowa Rozkładu - pełni ważną rolę w zwiększeniu dokładności predykcji.

Powyżej przedstawiam także sumę wszystkich tych składowych. Jak widać, każda z nich powoli maleje (jedynie *dfl\_loss* na początku wzrasta), ale ich suma stale maleje, co wydaje się być dobrym prognozykiem.

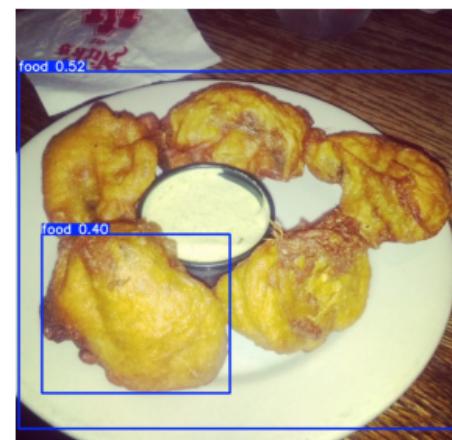
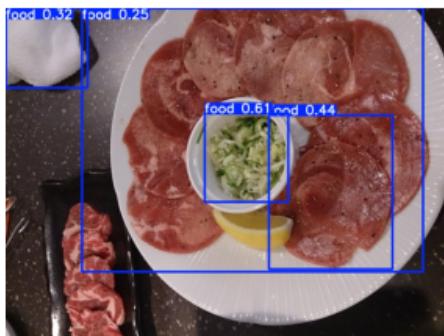
Analogicznie przedstawiam wykres dla danych walidacyjnych.



Rysunek 3: Wykres funkcji kosztu dla danych walidacyjnych

W tym przypadku także widać tendencję malejącą, zatem model nie powinien doświadczyć overfittingu (tylko w 3. epoce następuje niepokojący skok składowej związanej z klasifikacją, ale w kolejnej epoce jest "odbiecie"). Finalnie wartości bezwzględne wszystkich składowych są porównywalne do tych dla treningu, nieco wyższe.

Teraz przetestuję model na 4 losowo wybranych obrazach ze zbioru testowego.



Rysunek 4: Obrazy ze zbioru testowego z oznaczonymi obiektami jedzenia wraz z pewnością wykrycia

Wytrenowany model w miarę poprawnie rozpoznaje jedzenie na obrazie. Zdarzają się "zagnieżdżone" bounding boxy oraz takie z niską pewnością wykrycia (tutaj nie ustawiam żadnego progu pewności), ale zasadniczo wszelkie przejawy jedzenia na obrazie są wykryte (poza elementem w lewym dolnym rogu pierwszego obrazu).

Na zestawie innych 6 obrazów ze zbioru testowego dokonuję próby rozmycia jedzenia. W tym celu wykorzystuję wynik zwrócony przez predykcję modelu, który zawiera przewidywane dla obrazu bounding-boxy. W ramach pojedynczego bounding-boxa podane są współrzędne lewego górnego rogu oraz prawego dolnego rogu prostokąta. Wykorzystuję je, aby działać na odpowiednim obszarze obrazu. Dokonuję na nim rozmycia gaussowskiego (*Gaussian blur*) z wykorzystaniem funkcji `cv2.GaussianBlur` z biblioteki OpenCV. Efekt końcowy prezentuje się następująco:



Rysunek 5: Obrazy ze zbioru testowego z rozmazanymi obszarami tam, gdzie model wykrywa jedzenie

Wydaje mi się, że algorytm spełnia swoje zadanie. Większość z tych obrazów jest co prawda rozpoznawana wręcz w całości jako jedzenie, ale na przykładzie drugiego obrazu widać wyraźnie rozmazaną środkową część przy pozostawieniu reszty obrazu nienaruszonej, zatem algorytm nie cenzuruje "bezmyślnie" całego obrazu. W pozostałych przypadkach akurat tak się składa, że obiekt jedzenia zajmuje znaczną część obrazu.