Cyfrowe przetwarzanie obrazów

Projekt

„Wykrywanie piłki siatkowej na obrazie”

Wykonali:
Przemysław Kobylański, 297253

Grzegorz Ołdakowski, 297313

Warszawa 2023

1. Cel projektu

Celem projektu było wykonanie za pomocą Pytorcha algorytmu uczenia maszynowego którego zadaniem będzie rozpoznawanie na zdjęciu piłek siatkowych wraz z ich położeniem.

2. Wykonanie projektu

Podstawowymi krokami do wykonania projektu było:

- wykonanie datasetu na którym siec będzie trenowana
- stworzenie kodu który dotrenuje zaimportowaną sieć na zadanym zbiorze danych
- tuning parametrów tak aby trenowanie było optymalne
- walidacja na przesłanych przez prowadzącego zdjęciach sekwencji z kamer na których rejestrowane były piłki siatkowe

2.1. Wykonanie datasetu

Do wykonania datasetu użyto zdjęć z ogólnodostępnego datasetu COCO
Pobraliśmy zdjęcia validacyjne z roku 2017 (5 tysięcy zdjęć) z czego następnie wybraliśmy 1000
Dodatkowo w Internecie znaleźliśmy 9 zdjęć piłki Mikasy, które w programie Paint2D zostały przycięte tak aby ich boki były styczne do kwadratu zdjęcia. Piłki wkleiliśmy na zdjęcia z datasetu COCO wraz z zapisaniem w pliku JSON dane bounding boxów położenia tych piłek oraz zapisując zdjęcia binarne masek na których obaszar piłek przedstawiają jedynki, a tło zera. Cała operacja została wykonana za pomocą kodu dataset.py załączonego do folderu z projektem (screeny z kodem poniżej).

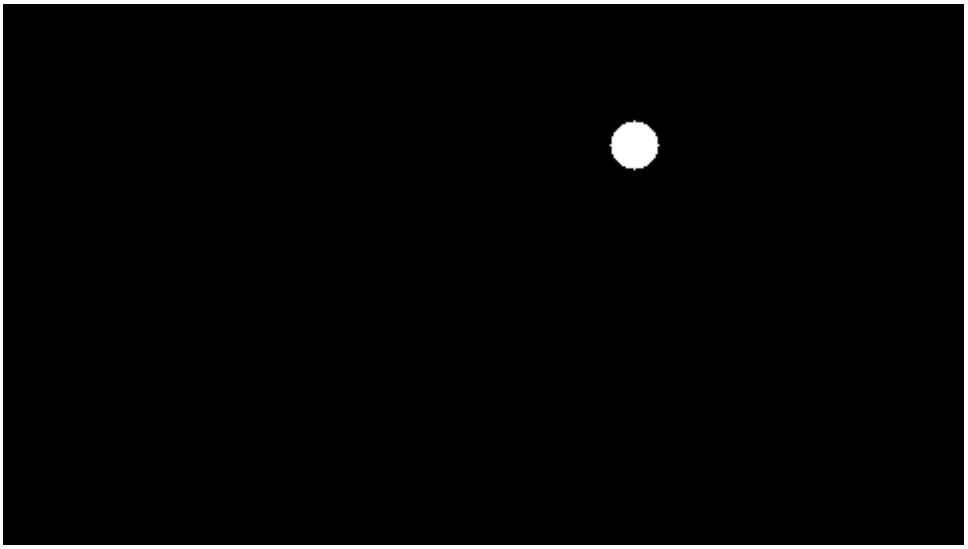
```python
# Ścieżki do folderów i plików
coco_images_dir = 'D:/Studia/Studia/Magisterskie/Sem2/CPO/Sieci/Dataset'
ball_images_dir = 'D:/Studia/Studia/Magisterskie/Sem2/CPO/Sieci/Pilki_dataset'
output_dir = 'D:/Studia/Studia/Magisterskie/Sem2/CPO/Sieci/FinalDataset3'

# Lista plików w folderze COCO
coco_images = [f for f in os.listdir(coco_images_dir) if f.endswith('.jpg')]

# Lista plików z piłkami
ball_images = [f for f in os.listdir(ball_images_dir) if f.endswith('.jpg')]

# Tworzenie folderu wyjściowego
if not os.path.exists(output_dir):
    os.makedirs(output_dir)

# Inicjalizacja słownika do przechowywania metadanych o obrazach i bounding boxach
metadata = {}

# Przechodzenie przez obrazy COCO
for coco_image in coco_images:
    # Wczytanie obrazu COCO
    coco_image_path = os.path.join(coco_images_dir, coco_image)
    coco_img = cv2.imread(coco_image_path)

    # Resize obrazu COCO do rozmiaru 480x270
    coco_img = cv2.resize(coco_img, (480, 270))

    # Losowy wybór obrazu piłki
    ball_image_name = np.random.choice(ball_images)
    ball_image_path = os.path.join(ball_images_dir, ball_image_name)
    ball_img = cv2.imread(ball_image_path)

    # Randomizacja wielkości piłki
    ball_size = np.random.randint(9, 34)
    ball_img = cv2.resize(ball_img, (ball_size, ball_size))

    # Generowanie losowych współrzędnych dla wklejenia obrazu piłki na obraz COCO
    x = np.random.randint(0, coco_img.shape[1] - ball_img.shape[1])
    y = np.random.randint(0, coco_img.shape[0] - ball_img.shape[0])

    # Tworzenie maski okręgu o rozmiarze piłki
    mask = np.zeros((ball_img.shape[0], ball_img.shape[1]), dtype=np.uint8)
    center = (ball_img.shape[1] // 2, ball_img.shape[0] // 2)
    radius = ball_img.shape[1] // 2
    cv2.circle(mask, center, radius, 255, -1)

    # Wklejenie obrazu piłki na obraz COCO z uwzględnieniem maski
    masked_ball_img = cv2.bitwise_and(ball_img, ball_img, mask=mask)
    mask = cv2.bitwise_not(mask)
    roi = coco_img[y:y+ball_img.shape[0], x:x+ball_img.shape[1]]
    roi = cv2.bitwise_and(roi, roi, mask=mask)
    roi = cv2.bitwise_or(roi, masked_ball_img)
    coco_img[y:y+ball_img.shape[0], x:x+ball_img.shape[1]] = roi

    # Zapis współrzędnych bounding boxa
    bbox = [x, y, x + ball_img.shape[1], y + ball_img.shape[0]]

    # Dodanie metadanych do słownika
    metadata[coco_image] = bbox
```

```
    # Zapis obrazu COCO z wklejoną piłką
    output_image_path = os.path.join(output_dir, coco_image)
    cv2.imwrite(output_image_path, coco_img)

    # Tworzenie maski okręgu o rozmiarze piłki
    mask = np.zeros((coco_img.shape[0], coco_img.shape[1]), dtype=np.uint8)
    center = (x + ball_img.shape[1] // 2, y + ball_img.shape[0] // 2)
    radius = ball_img.shape[1] // 2
    cv2.circle(mask, center, radius, 255, -1)

    # Zapis zbinaryzowanego obrazu z zaznaczonym położeniem piłki jako okręg
    binary_image = np.zeros((coco_img.shape[0], coco_img.shape[1]), dtype=np.uint8)
    binary_image[mask > 0] = 255
    binary_output_path = os.path.splitext(output_image_path)[0] + '_binary.png'
    cv2.imwrite(binary_output_path, binary_image)

# Zapis metadanych do pliku JSON
metadata_path = os.path.join(output_dir, 'metadata.json')
with open(metadata_path, 'w') as f:
    json.dump(metadata, f)
```

Początkowo przeskalowano wszystkie zdjęcia do formatu FullHD (1920x1080) jednak potem je zmniejszono 4 krotnie w każdym wymiarze, więc łącznie 16 krotnie do formatu 480x270 aby uczenie przebiegało szybciej

Przykładowe zdjęcie z piłką:

Oraz maska



2.2. Stworzenie notatnika

Następnym krokiem było stworzenie notatnika w google colab na podstawie notatnika przykładowego ze strony Pytorcha do wykonania treningu z maską. Na jego bazie stworzyliśmy notatnik załączony w plikach w folderze projektu (CPO2D_torch.ipynb). W notatniku można wyróżnić następujące kroki:
Zaimportowanie niezbędnych rzeczy (pobranie repo z Pytorchem, zainstalowanie starszej wersji Pytorcha ponieważ najnowsza dawała błędy, mountowanie dysku itp.)
Ważnymi z programistycznego punktu widzenia były kroki tworzenia datasetu oraz dobranie parametrów treningowych – reszta przebiegała jak w tutorialu
Definiowanie datasetu:

```
#@title Defining dataset
import os
import numpy as np
import torch
import glob
import json
import ntpath
import torch.utils.data
from PIL import Image


class volleyBallsDataset(torch.utils.data.Dataset):
    def __init__(self, root, transforms=None):
        self.root = root
        self.transforms = transforms
        # load all image files, sorting them to
        # ensure that they are aligned
        self.imgs =list(glob.glob(root+"/*.jpg"))
        self.masks =list(glob.glob(root+"/*.png"))
```

```
    def __getitem__(self, idx):
        # load images
        img_path = self.imgs[idx]
        mask_path = self.masks[idx]
        # print(img_path)
        img = Image.open(img_path).convert("RGB")
        mask=Image.open(mask_path)


        mask = np.array(mask)
        # instances are encoded as different colors
        obj_ids = np.unique(mask)
        # first id is the background, so remove it
        obj_ids = obj_ids[1:]

        # split the color-encoded mask into a set
        # of binary masks
        masks = mask == obj_ids[:, None, None]

        # import bboxes from json file
        boxes = []
        with
open('/content/gdrive/MyDrive/CPO2D/FinalDataset2/metadata.json') as
json_file:
          data = json.load(json_file)
          filename = ntpath.basename(img_path)
          boxes.append(data[filename])

        # convert bboxes to pytorch tensor
        boxes = torch.as_tensor(boxes, dtype=torch.float32)
        # # there is only one class
        num_objs = len(obj_ids)
        labels = torch.ones((num_objs,), dtype=torch.int64)
        masks=torch.as_tensor(masks, dtype=torch.uint8)

        image_id = torch.tensor([idx])
        area = (boxes[:, 3] - boxes[:, 1]) * (boxes[:, 2] - boxes[:,
0])
        # suppose all instances are not crowd
        iscrowd = torch.zeros((num_objs,), dtype=torch.int64)

        target = {}
        target["boxes"] = boxes
        target["labels"] = labels
        target["masks"] = masks
        target["image_id"] = image_id
        target["area"] = area
        target["iscrowd"] = iscrowd
```

```
        if self.transforms is not None:
            img, target = self.transforms(img, target)

        return img, target

    def __len__(self):
        return len(self.imgs)
```

W przypadku definiowania datasetu należało wczytać obrazy z piłkami wraz z maskami, odpowiednio je przekonwertować oraz wczytać współrzędne bounding boxów z pliku JSON. Boxy i maski zostały przekonwertowane do tensorów a cała klasa została wyposażona w interesujące nas parametry takie jak współrzędne boxa, maski, obszar boxa, id zdjęcia, etykieta oraz informacja czy wykryte obiekty na siebie zachodzą (nie spodziewamy się takich przypadków).

Następnymi krokami było:

- Wczytanie przetrenowanego modelu resnet50 oraz określenie naszej liczby klas (2 – tło i piłka).
- Zaimportowanie torchvision w celu segmentacji (która koniec końców nie zadziałała tak jak liczyliśmy ale całość treningu się powiodła)
- Zdefiniowanie funkcji transform dla datasetu (wykorzystywana dla danych treningowych).
- Testowanie na jednym batchu danych zużycia RAMu oraz zapoznanie się z danymi wchodzącym i wychodzącymi z sieci

```
#@title Training
import utils
model =
torchvision.models.detection.fasterrcnn_resnet50_fpn(pretrained=True)
dataset =
volleyBallsDataset('/content/gdrive/MyDrive/CPO2D/FinalDataset2',
get_transform(train=True))
data_loader = torch.utils.data.DataLoader(
    dataset, batch_size=8, shuffle=True, num_workers=2,
    collate_fn=utils.collate_fn
)
# For Training
images,targets = next(iter(data_loader))
# images = list(torch.FloatTensor([image, 1080, 1920]) for image in
images)
images = list(image for image in images)
targets = [{k: v for k, v in t.items()} for t in targets]
output = model(images,targets)   # Returns losses and detections
# For inference
model.eval()
x = [torch.rand(3, 300, 400), torch.rand(3, 500, 400)]
predictions = model(x)           # Returns predictions
```

```
print(predictions)
```

W tym kroku ustalono wielkość batch_size (ile zdjęć na raz jest analizowanych) w oparciu o dostępny w środowiku google colab RAM. W przypadku batchy powyżej 8 RAM-u brakowało.

Następnie podzielono dataset na część treningową i testową (pominięto dane walidacyjne)

```
#@title Split dataset into training and verification sets
# use our dataset and defined transformations
dataset =
volleyBallsDataset("/content/gdrive/MyDrive/CPO2D/FinalDataset2",
get_transform(train=True))
dataset_test =
volleyBallsDataset("/content/gdrive/MyDrive/CPO2D/FinalDataset2",
get_transform(train=False))

# split the dataset in train and test set
torch.manual_seed(1)
indices = torch.randperm(len(dataset)).tolist()
dataset = torch.utils.data.Subset(dataset, indices[:-100])
dataset_test = torch.utils.data.Subset(dataset_test, indices[-100:])

# define training and validation data loaders
data_loader = torch.utils.data.DataLoader(
    dataset, batch_size=8, shuffle=True, num_workers=2,
    collate_fn=utils.collate_fn)

data_loader_test = torch.utils.data.DataLoader(
    dataset_test, batch_size=8, shuffle=False, num_workers=2,
    collate_fn=utils.collate_fn)
```

Dane zostały podzielone w proporcji 9:1 na rzecz danych testowych

Następnie zdefiniowano parametry treningu

```
# our dataset has two classes only - background and person
num_classes = 2

# get the model using our helper function
model = get_instance_segmentation_model(num_classes)
# move model to the right device
model.to(device)

# construct an optimizer
params = [p for p in model.parameters() if p.requires_grad]
optimizer = torch.optim.SGD(params, lr=0.005,
                            momentum=0.9, weight_decay=0.0005)

# and a learning rate scheduler which decreases the learning rate by
```

```
# 10x every 3 epochs
lr_scheduler = torch.optim.lr_scheduler.StepLR(optimizer,
                                               step_size=3,
                                               gamma=0.1)
```

Najważniejszym parametrem tutaj jest lr – learning rate która określa szybkość uczenia się modelu. U nas zostawiono domyślną wartość 0.005 która jest zwiększana 10 krotnie co każde 3 epoki.

Ostatnim z kroków było wytrenowanie modelu

```
# let's train it for 5 epochs
from torch.optim.lr_scheduler import StepLR
num_epochs = 5

for epoch in range(num_epochs):
    # train for one epoch, printing every 40 iterations
    train_one_epoch(model, optimizer, data_loader, device,
epoch,print_freq=40)
    # update the learning rate
    lr_scheduler.step()
    # evaluate on the test dataset
    evaluate(model, data_loader_test, device=device)
```

Wyniki dla modelu następują się następująco:

```
Epoch: [0]  [  0/113]  eta: 0:22:15  lr: 0.000050  loss: 4.4503 (4.4503)  loss_classifier: 0.8621
(0.8621)  loss_box_reg: 0.1164 (0.1164)  loss_mask: 3.4350 (3.4350)  loss_objectness: 0.0361 (0.0361)
loss_rpn_box_reg: 0.0005 (0.0005)  time: 11.8175  data: 2.8878  max mem: 10945
Epoch: [0]  [ 40/113]  eta: 0:03:00  lr: 0.001834  loss: 0.2730 (0.8296)  loss_classifier: 0.0920
(0.2309)  loss_box_reg: 0.1231 (0.1052)  loss_mask: 0.0201 (0.4585)  loss_objectness: 0.0202 (0.0325)
loss_rpn_box_reg: 0.0025 (0.0026)  time: 2.2843  data: 0.0298  max mem: 10945
Epoch: [0]  [ 80/113]  eta: 0:01:20  lr: 0.003617  loss: 0.2157 (0.5588)  loss_classifier: 0.0495
(0.1461)  loss_box_reg: 0.1492 (0.1239)  loss_mask: 0.0077 (0.2656)  loss_objectness: 0.0039 (0.0206)
loss_rpn_box_reg: 0.0015 (0.0026)  time: 2.3951  data: 0.0309  max mem: 10945
Epoch: [0]  [112/113]  eta: 0:00:02  lr: 0.005000  loss: 0.0969 (0.4523)  loss_classifier: 0.0223
(0.1125)  loss_box_reg: 0.0633 (0.1145)  loss_mask: 0.0106 (0.2077)  loss_objectness: 0.0012 (0.0154)
loss_rpn_box_reg: 0.0010 (0.0022)  time: 2.3400  data: 0.0301  max mem: 10945
Epoch: [0] Total time: 0:04:32 (2.4113 s / it)
creating index...
index created!
Test:  [ 0/13]  eta: 0:00:20  model_time: 1.0633 (1.0633)  evaluator_time: 0.0300 (0.0300)  time: 1.5693
data: 0.4674  max mem: 10945
Test:  [12/13]  eta: 0:00:01  model_time: 1.0013 (0.9684)  evaluator_time: 0.0156 (0.0184)  time: 1.0551
data: 0.0622  max mem: 10945
Test: Total time: 0:00:13 (1.0702 s / it)
Averaged stats: model_time: 1.0013 (0.9684)  evaluator_time: 0.0156 (0.0184)
Accumulating evaluation results...
DONE (t=0.01s).
Accumulating evaluation results...
DONE (t=0.01s).
IoU metric: bbox
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.763
 Average Precision  (AP) @[ IoU=0.50      | area=   all | maxDets=100 ] = 0.993
 Average Precision  (AP) @[ IoU=0.75      | area=   all | maxDets=100 ] = 0.972
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = 0.763
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = -1.000
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = -1.000
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=  1 ] = 0.802
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets= 10 ] = 0.802
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.802
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = 0.802
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = -1.000
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = -1.000
IoU metric: segm
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.000
 Average Precision  (AP) @[ IoU=0.50      | area=   all | maxDets=100 ] = 0.000
 Average Precision  (AP) @[ IoU=0.75      | area=   all | maxDets=100 ] = 0.000
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = 0.000
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = -1.000
```

```
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = -1.000
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=  1 ] = 0.000
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets= 10 ] = 0.000
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.000
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = 0.000
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = -1.000
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = -1.000
Epoch: [1]  [  0/113]  eta: 0:05:17  lr: 0.005000  loss: 0.0684 (0.0684)  loss_classifier: 0.0167
(0.0167)  loss_box_reg: 0.0472 (0.0472)  loss_mask: 0.0038 (0.0038)  loss_objectness: 0.0003 (0.0003)
loss_rpn_box_reg: 0.0005 (0.0005)  time: 2.8135  data: 0.4448  max mem: 10945
Epoch: [1]  [ 40/113]  eta: 0:02:56  lr: 0.005000  loss: 0.0581 (0.1002)  loss_classifier: 0.0159
(0.0178)  loss_box_reg: 0.0309 (0.0354)  loss_mask: 0.0086 (0.0458)  loss_objectness: 0.0003 (0.0005)
loss_rpn_box_reg: 0.0005 (0.0007)  time: 2.3768  data: 0.0313  max mem: 10945
Epoch: [1]  [ 80/113]  eta: 0:01:19  lr: 0.005000  loss: 0.0602 (0.1046)  loss_classifier: 0.0132
(0.0162)  loss_box_reg: 0.0256 (0.0311)  loss_mask: 0.0202 (0.0561)  loss_objectness: 0.0003 (0.0006)
loss_rpn_box_reg: 0.0005 (0.0006)  time: 2.3908  data: 0.0306  max mem: 10945
Epoch: [1]  [112/113]  eta: 0:00:02  lr: 0.005000  loss: 0.0440 (0.0971)  loss_classifier: 0.0124
(0.0151)  loss_box_reg: 0.0222 (0.0286)  loss_mask: 0.0061 (0.0523)  loss_objectness: 0.0001 (0.0005)
loss_rpn_box_reg: 0.0004 (0.0006)  time: 2.3340  data: 0.0293  max mem: 10945
Epoch: [1] Total time: 0:04:30 (2.3924 s / it)
creating index...
index created!
Test:  [ 0/13]  eta: 0:00:23  model_time: 1.0277 (1.0277)  evaluator_time: 0.0176 (0.0176)  time: 1.7844
data: 0.7317  max mem: 10945
Test:  [12/13]  eta: 0:00:01  model_time: 0.9875 (0.9558)  evaluator_time: 0.0117 (0.0124)  time: 1.0522
data: 0.0786  max mem: 10945
Test: Total time: 0:00:14 (1.0795 s / it)
Averaged stats: model_time: 0.9875 (0.9558)  evaluator_time: 0.0117 (0.0124)
Accumulating evaluation results...
DONE (t=0.02s).
Accumulating evaluation results...
DONE (t=0.01s).
IoU metric: bbox
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.862
 Average Precision  (AP) @[ IoU=0.50      | area=   all | maxDets=100 ] = 1.000
 Average Precision  (AP) @[ IoU=0.75      | area=   all | maxDets=100 ] = 1.000
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = 0.862
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = -1.000
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = -1.000
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=  1 ] = 0.882
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets= 10 ] = 0.882
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.882
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = 0.882
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = -1.000
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = -1.000
IoU metric: segm
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.000
 Average Precision  (AP) @[ IoU=0.50      | area=   all | maxDets=100 ] = 0.000
 Average Precision  (AP) @[ IoU=0.75      | area=   all | maxDets=100 ] = 0.000
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = 0.000
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = -1.000
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = -1.000
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=  1 ] = 0.000
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets= 10 ] = 0.000
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.000
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = 0.000
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = -1.000
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = -1.000
Epoch: [2]  [  0/113]  eta: 0:05:54  lr: 0.005000  loss: 0.0308 (0.0308)  loss_classifier: 0.0109
(0.0109)  loss_box_reg: 0.0143 (0.0143)  loss_mask: 0.0055 (0.0055)  loss_objectness: 0.0001 (0.0001)
loss_rpn_box_reg: 0.0001 (0.0001)  time: 3.1337  data: 0.6980  max mem: 10945
Epoch: [2]  [ 40/113]  eta: 0:02:56  lr: 0.005000  loss: 0.0469 (0.0814)  loss_classifier: 0.0109
(0.0113)  loss_box_reg: 0.0208 (0.0215)  loss_mask: 0.0137 (0.0479)  loss_objectness: 0.0002 (0.0003)
loss_rpn_box_reg: 0.0005 (0.0004)  time: 2.3816  data: 0.0314  max mem: 10945
Epoch: [2]  [ 80/113]  eta: 0:01:19  lr: 0.005000  loss: 0.0392 (0.0848)  loss_classifier: 0.0102
(0.0108)  loss_box_reg: 0.0165 (0.0200)  loss_mask: 0.0132 (0.0533)  loss_objectness: 0.0001 (0.0003)
loss_rpn_box_reg: 0.0003 (0.0004)  time: 2.3890  data: 0.0306  max mem: 10945
Epoch: [2]  [112/113]  eta: 0:00:02  lr: 0.005000  loss: 0.0303 (0.0767)  loss_classifier: 0.0095
(0.0106)  loss_box_reg: 0.0159 (0.0192)  loss_mask: 0.0031 (0.0463)  loss_objectness: 0.0000 (0.0003)
loss_rpn_box_reg: 0.0002 (0.0004)  time: 2.3351  data: 0.0292  max mem: 10945
Epoch: [2] Total time: 0:04:30 (2.3958 s / it)
creating index...
index created!
Test:  [ 0/13]  eta: 0:00:19  model_time: 1.0322 (1.0322)  evaluator_time: 0.0104 (0.0104)  time: 1.4718
data: 0.4235  max mem: 10945
Test:  [12/13]  eta: 0:00:01  model_time: 0.9898 (0.9561)  evaluator_time: 0.0104 (0.0123)  time: 1.0333
data: 0.0588  max mem: 10945
Test: Total time: 0:00:13 (1.0496 s / it)
Averaged stats: model_time: 0.9898 (0.9561)  evaluator_time: 0.0104 (0.0123)
Accumulating evaluation results...
DONE (t=0.01s).
Accumulating evaluation results...
DONE (t=0.01s).
```

```
IoU metric: bbox
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.903
 Average Precision  (AP) @[ IoU=0.50      | area=   all | maxDets=100 ] = 1.000
 Average Precision  (AP) @[ IoU=0.75      | area=   all | maxDets=100 ] = 1.000
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = 0.903
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = -1.000
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = -1.000
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=  1 ] = 0.921
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets= 10 ] = 0.921
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.921
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = 0.921
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = -1.000
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = -1.000
IoU metric: segm
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.000
 Average Precision  (AP) @[ IoU=0.50      | area=   all | maxDets=100 ] = 0.000
 Average Precision  (AP) @[ IoU=0.75      | area=   all | maxDets=100 ] = 0.000
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = 0.000
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = -1.000
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = -1.000
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=  1 ] = 0.000
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets= 10 ] = 0.000
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.000
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = 0.000
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = -1.000
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = -1.000
Epoch: [3]  [  0/113]  eta: 0:05:19  lr: 0.000500  loss: 0.0269 (0.0269)  loss_classifier: 0.0082
(0.0082)  loss_box_reg: 0.0130 (0.0130)  loss_mask: 0.0054 (0.0054)  loss_objectness: 0.0000 (0.0000)
loss_rpn_box_reg: 0.0002 (0.0002)  time: 2.8262  data: 0.4521  max mem: 10945
Epoch: [3]  [ 40/113]  eta: 0:02:56  lr: 0.000500  loss: 0.0317 (0.0945)  loss_classifier: 0.0098
(0.0098)  loss_box_reg: 0.0143 (0.0143)  loss_mask: 0.0054 (0.0697)  loss_objectness: 0.0000 (0.0002)
loss_rpn_box_reg: 0.0003 (0.0004)  time: 2.3862  data: 0.0333  max mem: 10945
Epoch: [3]  [ 80/113]  eta: 0:01:19  lr: 0.000500  loss: 0.0295 (0.0766)  loss_classifier: 0.0092
(0.0095)  loss_box_reg: 0.0134 (0.0144)  loss_mask: 0.0060 (0.0521)  loss_objectness: 0.0000 (0.0002)
loss_rpn_box_reg: 0.0002 (0.0003)  time: 2.3895  data: 0.0316  max mem: 10945
Epoch: [3]  [112/113]  eta: 0:00:02  lr: 0.000500  loss: 0.0293 (0.0690)  loss_classifier: 0.0095
(0.0095)  loss_box_reg: 0.0133 (0.0143)  loss_mask: 0.0057 (0.0445)  loss_objectness: 0.0000 (0.0002)
loss_rpn_box_reg: 0.0003 (0.0004)  time: 2.3357  data: 0.0316  max mem: 10945
Epoch: [3] Total time: 0:04:30 (2.3973 s / it)
creating index...
index created!
Test:  [ 0/13]  eta: 0:00:24  model_time: 1.0430 (1.0430)  evaluator_time: 0.0180 (0.0180)  time: 1.8791
data: 0.8079  max mem: 10945
Test:  [12/13]  eta: 0:00:01  model_time: 0.9859 (0.9555)  evaluator_time: 0.0103 (0.0120)  time: 1.0694
data: 0.0958  max mem: 10945
Test: Total time: 0:00:14 (1.0850 s / it)
Averaged stats: model_time: 0.9859 (0.9555)  evaluator_time: 0.0103 (0.0120)
Accumulating evaluation results...
DONE (t=0.01s).
Accumulating evaluation results...
DONE (t=0.01s).
IoU metric: bbox
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.916
 Average Precision  (AP) @[ IoU=0.50      | area=   all | maxDets=100 ] = 1.000
 Average Precision  (AP) @[ IoU=0.75      | area=   all | maxDets=100 ] = 1.000
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = 0.916
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = -1.000
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = -1.000
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=  1 ] = 0.932
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets= 10 ] = 0.932
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.932
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = 0.932
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = -1.000
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = -1.000
IoU metric: segm
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.000
 Average Precision  (AP) @[ IoU=0.50      | area=   all | maxDets=100 ] = 0.000
 Average Precision  (AP) @[ IoU=0.75      | area=   all | maxDets=100 ] = 0.000
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = 0.000
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = -1.000
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = -1.000
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=  1 ] = 0.000
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets= 10 ] = 0.000
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.000
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = 0.000
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = -1.000
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = -1.000
Epoch: [4]  [  0/113]  eta: 0:05:22  lr: 0.000500  loss: 0.0838 (0.0838)  loss_classifier: 0.0137
(0.0137)  loss_box_reg: 0.0127 (0.0127)  loss_mask: 0.0571 (0.0571)  loss_objectness: 0.0001 (0.0001)
loss_rpn_box_reg: 0.0003 (0.0003)  time: 2.8503  data: 0.4684  max mem: 10945
Epoch: [4]  [ 40/113]  eta: 0:02:56  lr: 0.000500  loss: 0.0290 (0.0544)  loss_classifier: 0.0086
(0.0089)  loss_box_reg: 0.0136 (0.0138)  loss_mask: 0.0054 (0.0313)  loss_objectness: 0.0000 (0.0001)
loss_rpn_box_reg: 0.0003 (0.0003)  time: 2.3867  data: 0.0345  max mem: 10945
```

```
Epoch: [4]  [ 80/113]  eta: 0:01:19  lr: 0.000500  loss: 0.0299 (0.0725)  loss_classifier: 0.0091
(0.0092)  loss_box_reg: 0.0138 (0.0139)  loss_mask: 0.0063 (0.0489)  loss_objectness: 0.0001 (0.0001)
loss_rpn_box_reg: 0.0003 (0.0003)  time: 2.3962  data: 0.0338  max mem: 10945
Epoch: [4]  [112/113]  eta: 0:00:02  lr: 0.000500  loss: 0.0316 (0.0674)  loss_classifier: 0.0096
(0.0092)  loss_box_reg: 0.0139 (0.0139)  loss_mask: 0.0061 (0.0438)  loss_objectness: 0.0001 (0.0001)
loss_rpn_box_reg: 0.0003 (0.0003)  time: 2.3299  data: 0.0286  max mem: 10945
Epoch: [4] Total time: 0:04:30 (2.3971 s / it)
creating index...
index created!
Test:  [ 0/13]  eta: 0:00:19  model_time: 1.0054 (1.0054)  evaluator_time: 0.0102 (0.0102)  time: 1.4654
data: 0.4421  max mem: 10945
Test:  [12/13]  eta: 0:00:01  model_time: 0.9914 (0.9541)  evaluator_time: 0.0104 (0.0125)  time: 1.0320
data: 0.0593  max mem: 10945
Test: Total time: 0:00:13 (1.0542 s / it)
Averaged stats: model_time: 0.9914 (0.9541)  evaluator_time: 0.0104 (0.0125)
Accumulating evaluation results...
DONE (t=0.01s).
Accumulating evaluation results...
DONE (t=0.01s).
IoU metric: bbox
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.912
 Average Precision  (AP) @[ IoU=0.50      | area=   all | maxDets=100 ] = 1.000
 Average Precision  (AP) @[ IoU=0.75      | area=   all | maxDets=100 ] = 1.000
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = 0.912
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = -1.000
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = -1.000
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=  1 ] = 0.926
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets= 10 ] = 0.926
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.926
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = 0.926
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = -1.000
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = -1.000
IoU metric: segm
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.000
 Average Precision  (AP) @[ IoU=0.50      | area=   all | maxDets=100 ] = 0.000
 Average Precision  (AP) @[ IoU=0.75      | area=   all | maxDets=100 ] = 0.000
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = 0.000
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = -1.000
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = -1.000
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=  1 ] = 0.000
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets= 10 ] = 0.000
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.000
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = 0.000
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = -1.000
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = -1.000
```

Jak można zaobserwować – precyzja modelu dla wykrywania bounding boxów rośnie bardzo szybko – mimo że 5 epok to dość niedużo to już nawet po 3 przekracza ona 0.9 IoU (stosunek pokrycia rzeczywistego bounding boxa z bounding boxem wykrytym przez sieć). Widać jednak że moduł segmentacji w ogóle nie działa i nie udało nam się znaleźć przyczyny tego faktu. Tak samo widać jak dla każdego batcha(wyświetlany jest co 40 w epoce) dla każdej epoki spada wartość loss – spada ona bardzo szybko do niskiego poziomu poniżej 0.1 już w pierwszej epoce spada z każdą epoką do wartości 0.0316 (chociaż minimalna wartość wynosi 0.290 dla tej samej epoki ale wcześniejszego batcha)

Na bazie tych danych można uznać że model został dobrze wytrenowany, należy go jeszcze tylko przetestować na zdjęciach spoza datasetu

2.3. Sprawdzenie skuteczności na zdjęciach

Aby sprawdzić czy wyniki dawane przez model są poprawne, wgrano na dysk część klatek wysłanych nam w pliku sekwencje i wypróbowano działanie modelu na kilku z nich. W tym celu dalej rozbudowano notatnik o kroki:
Ustalenie predykcji

```
#@title predykcje
import torchvision.transforms as transforms
from PIL import Image
```

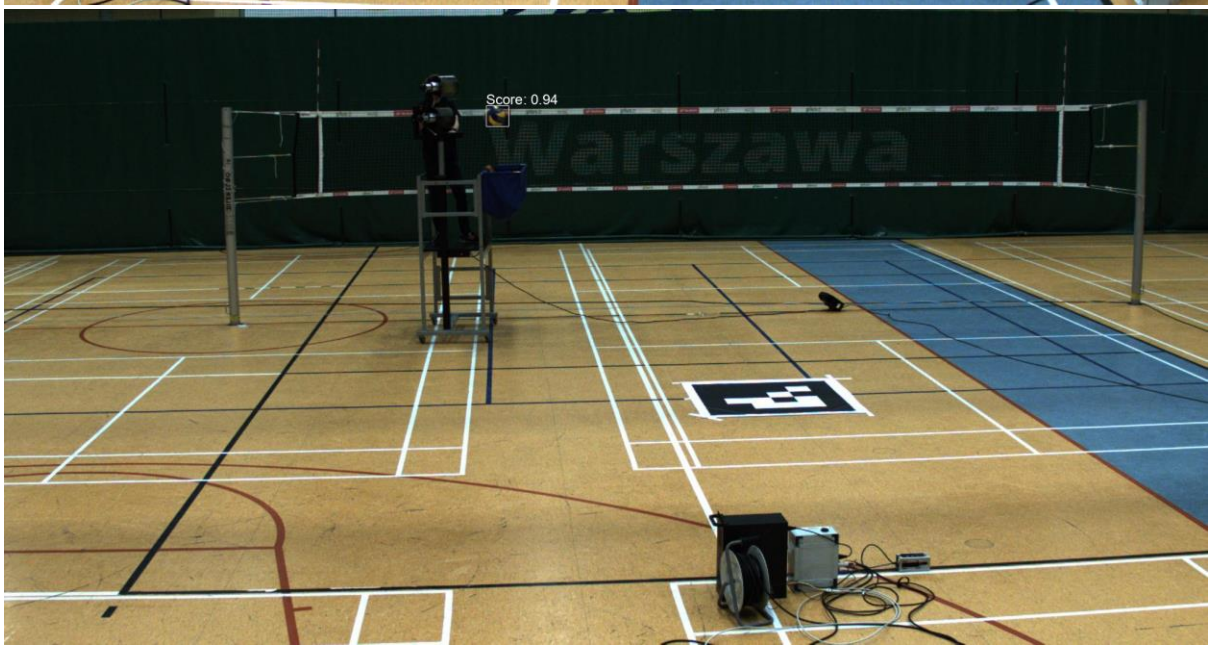
```
# pick one image from the test set
img = Image.open("/content/gdrive/MyDrive/CPO2D/frames/frame_138.jpg")
# Definiuj transformacje
transform = transforms.Compose([
    transforms.ToTensor()
])

# Zastosuj transformacje do obrazu
img_tensor = transform(img)
# put the model in evaluation mode
model.eval()
with torch.no_grad():
    prediction = model([img_tensor.to(device)])
```

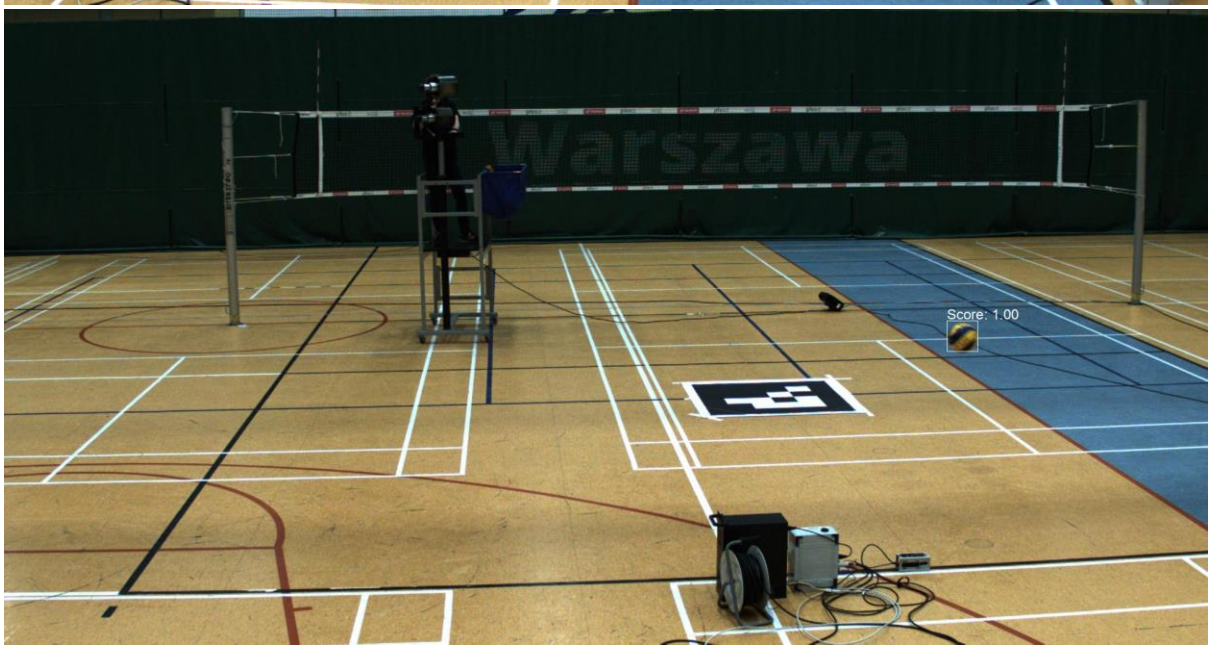
Sprawdzenie I wyświetlenie zdjęcia oraz zapisanie obrazu wynikowego

```
from PIL import Image, ImageDraw, ImageFont

# Konwertuj tensor na obiekt PIL.Image
img1 = Image.fromarray(img_tensor.mul(255).permute(1, 2,
0).byte().numpy())

# Pobierz wartość score z predykcji
score = prediction[0]['scores'].cpu().numpy()[0]

# Pobierz bbox
boxes = prediction[0]['boxes'].cpu().numpy()
bbox = boxes[0]

# Przygotuj kształt bbox
shape = [(int(bbox[0]), int(bbox[1])), (int(bbox[2]), int(bbox[3]))]

# Rysuj bbox i dodaj napis z score
draw = ImageDraw.Draw(img1)
draw.rectangle(shape, outline="white")
font = ImageFont.truetype("/content/gdrive/MyDrive/CPO2D/arial.ttf",
size=24)
draw.text((shape[0][0], shape[0][1] - 24), f"Score: {score:.2f}",
fill="white",font=font)

# Wyświetl obraz z bbox i napisem
img1.show()
img1.save('/content/gdrive/MyDrive/CPO2D/zdj4.jpg')
```

Dodatkowym krokiem było pobranie i wgranie na dysk pliku arial.tff tak aby zwiększyć rozmiar tej czcionki na obrazie aby całość lepiej wyglądała
Kilka zdjęć wynikowych z zaznaczonymi piłkami i podpisaną wartością score (oznaczającą pewność modelu co do położenia piłki) znajdują się poniżej:


Warszawa
Score: 1.00


Score: 0.94
Warszawa



3. Wnioski

- Sieć została poprawnie wytrenowana na stworzonym datasecie zapewniając możliwość skutecznego rozpoznawania piłek siatkowych.
- Możliwość doszkolenia pretrenowanej sieci neuronowej do realizacji konkretnej funkcji jest bardzo przydatną funkcją, ponieważ znacząco obniża próg wejścia rozwiązań opierających się na sieciach neuronowych do zastosowań pozaakademickich.
- Nie udało się jednak poprawnie odtwarzać masek z rozkładem prawdopodobieństwa występowania poszukiwanego obiektu. Wynikać to może zarówno z błędu w definicji którejś z funkcji powiązanych z trenowaniem modelu, lub błędu wynikającego z błędów/zmian pochodzących od konieczności skorzystania z starszej wersji torchvision.