

# Custom Vision vs Own Model for object detection

Cel :

Celem naszego projektu jest wytrenowanie, zapoznanie się z modelami dla wykrywanie obiektów:

1. Azure Custom Vision
2. Mask R-CNN

oraz porównanie własnoręcznie wytrenowanych modeli z usługą udostępnioną przez Microsoft.

**Azure Custom Vision** dostarcza możliwość wykorzystania dwóch rodzajów modeli do widzenia komputerowego:

- modelu do klasyfikacji obrazów oraz
- modelu do detekcji obiektów.

W naszym projekcie wykorzystaliśmy drugi typ, ponieważ skupiamy się na wykrywaniu różnego rodzaju obiektów występujących na zdjęciach.

**Mask R-CNN ResNet-50 FPN.** Do trenowania tego modelu wykorzystaliśmy bibliotekę Pytorch. Zdecydowaliśmy się na wybór tego modelu ze względu na to, że w dokumentacji wspomnianej biblioteki Pytorch osiąga on najlepsze rezultaty z dostępnych modeli.

Network	box AP	mask AP	keypoint AP
Faster R-CNN ResNet-50 FPN	37.0	•	•
RetinaNet ResNet-50 FPN	36.4	•	•
Mask R-CNN ResNet-50 FPN	37.9	34.6	•

**Image 1 - Porównanie**

Zbiory danych:

W celu porówniania działania obu serwisów sprawdziliśmy trzy zbiory danych:

- Monkey, Cat and Dog detection (30 MB) - [Kaggle](#),
- Fruit Images for Object Detection (28 MB) - [Kaggle](#)
- Card detection (38,6 MB) - [Github](#)

Poniżej zostały dostarczone wytrenowane modele na wyżej omówionych zbiorach danych.

## Azure Machine Learning Configuration

Poniżej zostały dostarczone wytrenowane modele na wyżej omówionych zbiorach danych.

Podłączanie bibliotek i pakietów:

```
In [1]: import azureml.core
from azureml.core import Workspace
from dotenv import set_key, get_key, find_dotenv
from pathlib import Path
from utilities import get_auth
```

Pokazanie poprawnego połączenie do Workspace Azure. Wyświetlenie wersji:

```
In [2]: print("You are currently using version", azureml.core.VERSION, "of the Azure
```

You are currently using version 1.17.0 of the Azure ML SDK

Konfiguracja Azure oraz environment

```
In [3]: import os
subscription_id = os.environ['SUB_ID']
resource_group = "ProjektAzure"
workspace_name = "ProjektAzure"
workspace_region = "East US"
```

Znajdowanie zasobu oraz połączenie po przez kluczy oraz subskrypcje:

```
In [4]: env_path = find_dotenv()
if env_path == "":
    Path(".env").touch()
env_path = find_dotenv()
```

```
In [5]: set_key(env_path, "subscription_id", subscription_id)
set_key(env_path, "resource_group", resource_group)
set_key(env_path, "workspace_name", workspace_name)
set_key(env_path, "workspace_region", workspace_region)
```

```
Out[5]: (True, 'workspace_region', 'East US')
```

Stworzenie roboczego obszaru przy użyciu określonych parametrów i zapisanie szczegółów obszaru roboczego do pliku konfiguracyjnego:

```
In [6]: # Create the workspace using the specified parameters
ws = Workspace.create(
    name=workspace_name,
    subscription_id=subscription_id,
    resource_group=resource_group,
    location=workspace_region,
    create_resource_group=True,
    auth=get_auth(env_path),
    exist_ok=True,
)

# write the details of the workspace to a configuration file
ws.write_config()
```

Poniżej można zobaczyć informację dotyczące roboczego obszaru (jeżeli odkomentować ws.get\_details())

```
In [8]: # load workspace configuration
ws = Workspace.from_config(auth=get_auth(env_path))
#ws.get_details()
```

Poniżej jest importowanie bibliotek oraz połączenie ścieżki do plików, które wskazują na

moduł:

```
In [9]: import sys

sys.path.append("scripts")
sys.path.append("scripts/cocoapi/PythonAPI/")

import azureml.core
from azureml.core import Workspace, Experiment
from azureml.widgets import RunDetails
from azureml.train.dnn import PyTorch

from dotenv import set_key, get_key, find_dotenv
from utilities import get_auth, download_data

import torch
from scripts.XMLDataset import BuildDataset, get_transform
from scripts.maskrcnn_model import get_model

from PIL import Image, ImageDraw
from IPython.display import display

# check core SDK version number
print("Azure ML SDK Version: ", azureml.core.VERSION)
```

Azure ML SDK Version: 1.17.0

## Implementacja Mask R-CNN model

```
In [ ]: #UWAGA! UWAGA! UWAGA!
```

```
In [9]: %%writefile scripts/XMLDataset.py
import os
import xml.etree.ElementTree as ET
import torch
import transforms as T
from PIL import Image

class BuildDataset(torch.utils.data.Dataset):
    def __init__(self, root, dataset, transforms=None, train=True):
        self.root = root
        self.transforms = transforms
        #self.labels_dict = {'obama': 1}
        #self.labels_dict = {'ace': 1, 'king': 2, 'queen': 3, 'jack': 4, 'ten': 5}
        self.labels_dict = {'cat': 1, 'dog': 2, 'monkey': 3}
        self.labels_dict['cat'] = 1
        self.labels_dict['dog'] = 2
        self.labels_dict['monkey'] = 3
        # load all image files
        if dataset == "Fruit":
            self.labels_dict = {'apple': 1, 'orange': 2, 'banana': 3}
        elif dataset == "PlayCards":
            self.labels_dict = {'ace': 1, 'king': 2, 'queen': 3, 'jack': 4, 'ten': 5}
        else:
            self.labels_dict = {'cat': 1, 'dog': 2, 'monkey': 3}
    if train:
        print(f"Data/{dataset}/JPEGImages")
        print(f"Data/{dataset}/Annotations")
        self.imgs_path = os.path.join(root, f"Data/{dataset}/JPEGImages")
        print(type(dataset))
        self.imgs = list(sorted(os.listdir(self.imgs_path)))
        self.xls_path = os.path.join(root, f"Data/{dataset}/Annotations")
```

```

else:
    self.imgs_path = os.path.join(root, f"Data/{dataset}/JPEGImages")
    self.imgs = list(sorted(os.listdir(self.imgs_path)))
    self.xls_path = os.path.join(root, f"Data/{dataset}/Annotations")

# if train:
#     self.imgs_path = os.path.join(root, "Data/PlayCards/JPEGImages")
#     self.imgs = list(sorted(os.listdir(self.imgs_path)))
#     self.xls_path = os.path.join(root, "Data/PlayCards/Annotations")
# else:
#     self.imgs_path = os.path.join(root, "Data/PlayCards/JPEGImages")
#     self.imgs = list(sorted(os.listdir(self.imgs_path)))
#     self.xls_path = os.path.join(root, "Data/PlayCards/Annotations")
# if train:
#     self.imgs_path = os.path.join(root, "Data/monkeyCats/JPEGImages")
#     self.imgs = list(sorted(os.listdir(self.imgs_path)))
#     self.xls_path = os.path.join(root, "Data/monkeyCats/Annotations")
# else:
#     self.imgs_path = os.path.join(root, "Data/monkeyCats/JPEGImages")
#     self.imgs = list(sorted(os.listdir(self.imgs_path)))
#     self.xls_path = os.path.join(root, "Data/monkeyCats/Annotations")

def __getitem__(self, idx):
    img_path = os.path.join(self.imgs_path, self.imgs[idx])
    xml_path = os.path.join(
        self.xls_path, "{}.xml".format(self.imgs[idx].strip(".jpg")))
    )
    img = Image.open(img_path).convert("RGB")

    # parse XML annotation
    tree = ET.parse(xml_path)
    t_root = tree.getroot()

    # get bounding box coordinates
    boxes = []
    labels = []
    for obj in t_root.findall("object"):
        bnd_box = obj.find("bndbox")
        xmin = float(bnd_box.find("xmin").text)
        xmax = float(bnd_box.find("xmax").text)
        ymin = float(bnd_box.find("ymin").text)
        ymax = float(bnd_box.find("ymax").text)
        boxes.append([xmin, ymin, xmax, ymax])
        label_name = str(obj.find("name").text)
        if self.labels_dict:
            if label_name not in self.labels_dict.keys():
                self.labels_dict[label_name] = max(self.labels_dict.values())
            else:
                self.labels_dict[label_name] = 1
            print(label_name + ": " + str(self.labels_dict[label_name]))
        labels.append(self.labels_dict[label_name])
    num_objs = len(boxes)
    boxes = torch.as_tensor(boxes, dtype=torch.float32)

    # there is only one class
    labels = torch.as_tensor(labels)
    print(labels)
    labels = torch.ones((num_objs,), dtype=torch.int64)
    image_id = torch.tensor([idx])

    # area of the bounding box, used during evaluation with the COCO metric
    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["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)

def get_transform(train):
    transforms = []
    transforms.append(T.ToTensor())
    if train:
        transforms.append(T.RandomHorizontalFlip(0.5))
    return T.Compose(transforms)

```

Overwriting scripts/XMLDataset.py

```

In [10]: %%writefile scripts/maskrcnn_model.py
import torchvision
from torchvision.models.detection.faster_rcnn import FastRCNNPredictor
from torchvision.models.detection.rpn import AnchorGenerator
from torchvision.models.detection.rpn import RPNHead

def get_model(
    num_classes,
    anchor_sizes,
    anchor_aspect_ratios,
    rpn_nms_threshold,
    box_nms_threshold,
    box_score_threshold,
    num_box_detections,
):
    # load pre-trained mask R-CNN model
    model = torchvision.models.detection.maskrcnn_resnet50_fpn(
        pretrained=True,
        rpn_nms_thresh=rpn_nms_threshold,
        box_nms_thresh=box_nms_threshold,
        box_score_thresh=box_score_threshold,
        box_detections_per_img=num_box_detections,
    )
    # get number of input features for the classifier
    in_features = model.roi_heads.box_predictor.cls_score.in_features

    # replace the pre-trained head with a new one
    model.roi_heads.box_predictor = FastRCNNPredictor(in_features, num_classes)

    anchor_sizes = tuple([float(i) for i in anchor_sizes.split(",")])
    anchor_aspect_ratios = tuple([float(i) for i in anchor_aspect_ratios.split(",")])

    # create an anchor_generator for the FPN which by default has 5 outputs
    anchor_generator = AnchorGenerator(
        sizes=tuple([anchor_sizes for _ in range(5)]),
        aspect_ratios=tuple([anchor_aspect_ratios for _ in range(5)]),
    )

```

```
model.rpn.anchor_generator = anchor_generator

# get number of input features for the RPN returned by FPN (256)
in_channels = model.backbone.out_channels

# replace the RPN head
model.rpn.head = RPNAhead(
    in_channels, anchor_generator.num_anchors_per_location()[0]
)

# turn off masks since dataset only has bounding boxes
model.roi_heads.mask_roi_pool = None

return model
```

Overwriting scripts/maskrcnn\_model.py

```

        "--anchor_aspect_ratios", default="1.0", type=str, help="anchor aspect ratios",
    )
    parser.add_argument(
        "--rpn_nms_thresh",
        default=0.7,
        type=float,
        help="NMS threshold used for postprocessing the RPN proposals",
    )
    parser.add_argument(
        "--box_nms_thresh",
        default=0.5,
        type=float,
        help="NMS threshold for the prediction head. Used during inference",
    )
    parser.add_argument(
        "--box_score_thresh",
        default=0.05,
        type=float,
        help="during inference only return proposals with a classification score greater than box_score_thresh",
    )
    parser.add_argument(
        "--box_detections_per_img",
        default=100,
        type=int,
        help="maximum number of detections per image, for all classes",
    )
    parser.add_argument(
        "--num_classes",
        default=4,
        type=int,
        help="number of classes + 1",
    )
    parser.add_argument(
        "--dataset",
        default="Fruit",
        type=str,
        help="Path to dataset",
    )
)
args = parser.parse_args()

data_path = args.data_path

# use our dataset and defined transformations
dataset = BuildDataset(data_path, args.dataset, get_transform(train=True), transform=None)
dataset_test = BuildDataset(data_path, args.dataset, get_transform(train=False), transform=None)
# dataset_test = BuildDataset(data_path, get_transform(train=False), train=False)

# split the dataset in train and test set
# indices = torch.randperm(len(dataset)).tolist()
# dataset = torch.utils.data.Subset(dataset, range(0, len(dataset)))
# dataset_test = torch.utils.data.Subset(dataset_test, range(0, len(dataset_test)))
# dataset = torch.utils.data.Subset(dataset, indices[:-100])
# dataset_test = torch.utils.data.Subset(dataset_test, indices[-100:])

batch_size = args.batch_size
workers = args.workers

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

```

```
)  
  
data_loader_test = torch.utils.data.DataLoader(  
    dataset_test,  
    batch_size=2,  
    shuffle=False,  
    num_workers=workers,  
    collate_fn=utils.collate_fn,  
)  
  
# our dataset has two classes only - background and out of stock  
num_classes = args.num_classes  
  
model = get_model(  
    num_classes,  
    args.anchor_sizes,  
    args.anchor_aspect_ratios,  
    args.rpn_nms_thresh,  
    args.box_nms_thresh,  
    args.box_score_thresh,  
    args.box_detections_per_img,  
)  
  
  
# train on the GPU or on the CPU, if a GPU is not available  
device = torch.device("cuda") if torch.cuda.is_available() else torch.device()  
  
# move model to the right device  
model.to(device)  
  
learning_rate = args.learning_rate  
momentum = args.momentum  
weight_decay = args.weight_decay  
  
# construct an optimizer  
params = [p for p in model.parameters() if p.requires_grad]  
optimizer = torch.optim.SGD(  
    params, lr=learning_rate, momentum=momentum, weight_decay=weight_decay  
)  
  
lr_step_size = args.lr_step_size  
lr_gamma = args.lr_gamma  
  
# and a learning rate scheduler  
lr_scheduler = torch.optim.lr_scheduler.StepLR(  
    optimizer, step_size=lr_step_size, gamma=lr_gamma  
)  
  
# number of training epochs  
num_epochs = args.epochs  
print_freq = args.print_freq  
  
for epoch in range(num_epochs):  
    # train for one epoch, printing every 10 iterations  
    train_one_epoch(model, optimizer, data_loader, device, epoch, print_freq=  
  
    # update the learning rate  
    lr_scheduler.step()  
    # evaluate on the test dataset after every epoch  
    #print("EVALUATE!!!!!!!!!!!!!!")  
    evaluate(model, data_loader_test, device=device)
```

```

        evaluate(model, data_loader, device=device)

    #

    # save model
    if not os.path.exists(args.output_dir):
        os.makedirs(args.output_dir)
    torch.save(model.state_dict(), os.path.join(args.output_dir, args.dataset+"model.pt"))

    print("That's it!")

```

Overwriting scripts/train.py

## Trenowanie modelu

```

In [19]: import sys

sys.path.append("scripts")
sys.path.append("scripts/cocoapi/PythonAPI/")

import azureml.core
from azureml.core import Workspace, Experiment
from azureml.widgets import RunDetails
from azureml.train.dnn import PyTorch

from dotenv import set_key, get_key, find_dotenv
from utilities import get_auth, download_data

import torch
from scripts.XMLDataset import BuildDataset, get_transform
from scripts.maskrcnn_model import get_model

from PIL import Image, ImageDraw
from IPython.display import display

# check core SDK version number
print("Azure ML SDK Version: ", azureml.core.VERSION)

```

Azure ML SDK Version: 1.17.0

```
In [20]: env_path = find_dotenv.raise_error_if_not_found=True)
```

```

In [21]: # data_file = "Data.zip"
# data_url = ("https://bostondata.blob.core.windows.net/builddata/{}".format(
# download_data(data_file, data_url)

```

```

In [22]: ws = Workspace.from_config(auth=get_auth(env_path))
print(ws.name, ws.resource_group, ws.location, sep="\n")

```

ProjektAzure  
ProjektAzure  
eastus

```
In [23]: exp = Experiment(workspace=ws, name='torchvision')
```

```

In [17]: # with open("scripts/train.py", "r") as f:
#         print(f.read())

```

```

In [18]: num_epochs = 2
script_params = {
    "--data_path": ".",
    "--workers": 8,
    "--learning_rate": 0.005,
    "--epochs": num_epochs,
}

```

```

    "--anchor_sizes": "16,32,64,128,256,512",
    "--anchor_aspect_ratios": "0.25,0.5,1.0,2.0",
    "--rpn_nms_thresh": 0.5,
    "--box_nms_thresh": 0.3,
    "--box_score_thresh": 0.10,
    "--num_classes": 4,
    "--dataset": "Fruit",
}

estimator = PyTorch(
    source_directory=".scripts",
    script_params=script_params,
    compute_target="local",
    entry_script="train.py",
    use_docker=False,
    user_managed=True,
    use_gpu=True,
)

```

```

In [19]: def prepareEstimator (dataset,clas,num_epochs,outPut_fileName):
    script_params = {
        "--data_path": ".",
        "--workers": 8,
        "--learning_rate": 0.005,
        "--epochs": num_epochs,
        "--anchor_sizes": "16,32,64,128,256,512",
        "--anchor_aspect_ratios": "0.25,0.5,1.0,2.0",
        "--rpn_nms_thresh": 0.5,
        "--box_nms_thresh": 0.3,
        "--box_score_thresh": 0.10,
        "--num_classes": clas,
        "--dataset": dataset,
    }

    estimator = PyTorch(
        source_directory=".scripts",
        script_params=script_params,
        compute_target="local",
        entry_script="train.py",
        use_docker=False,
        user_managed=True,
        use_gpu=True,
    )
    estimator.run_config.environment.python.interpreter_path = ("/anaconda/er
estimator.run_config.history.snapshot_project = False
run = exp.submit(estimator)
RunDetails(run).show()
run.wait_for_completion(show_output=True)
metrics = run.get_metrics()
run.get_file_names()
run.register_model(model_name="torchvision_local_model", model_path="/out
run.download_file("outputs/" +outPut_fileName+"model_latest.pth")
return metrics

```

## Ładowanie datasetów

DataSet Fruit

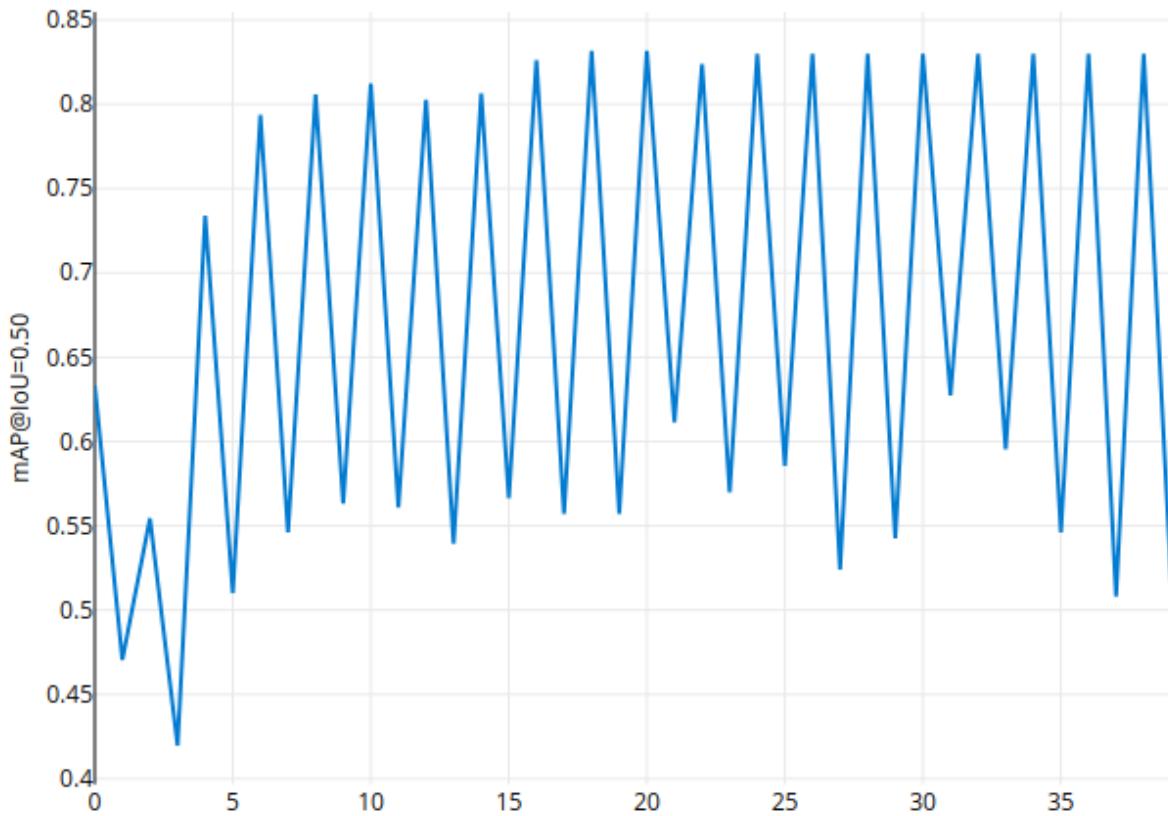
### Arguments

```
--data_path . --workers 8 --learning_rate 0.005 --epochs 20 --
anchor_sizes 16,32,64,128,256,512 --anchor_aspect_ratios
```

```
0.25,0.5,1.0,2.0 --rpn_nms_thresh 0.5 --box_nms_thresh 0.3 --
box_score_thresh 0.1 --num_classes 4 --dataset Fruit
```

Czas trwania: 1h 54m 18.30s

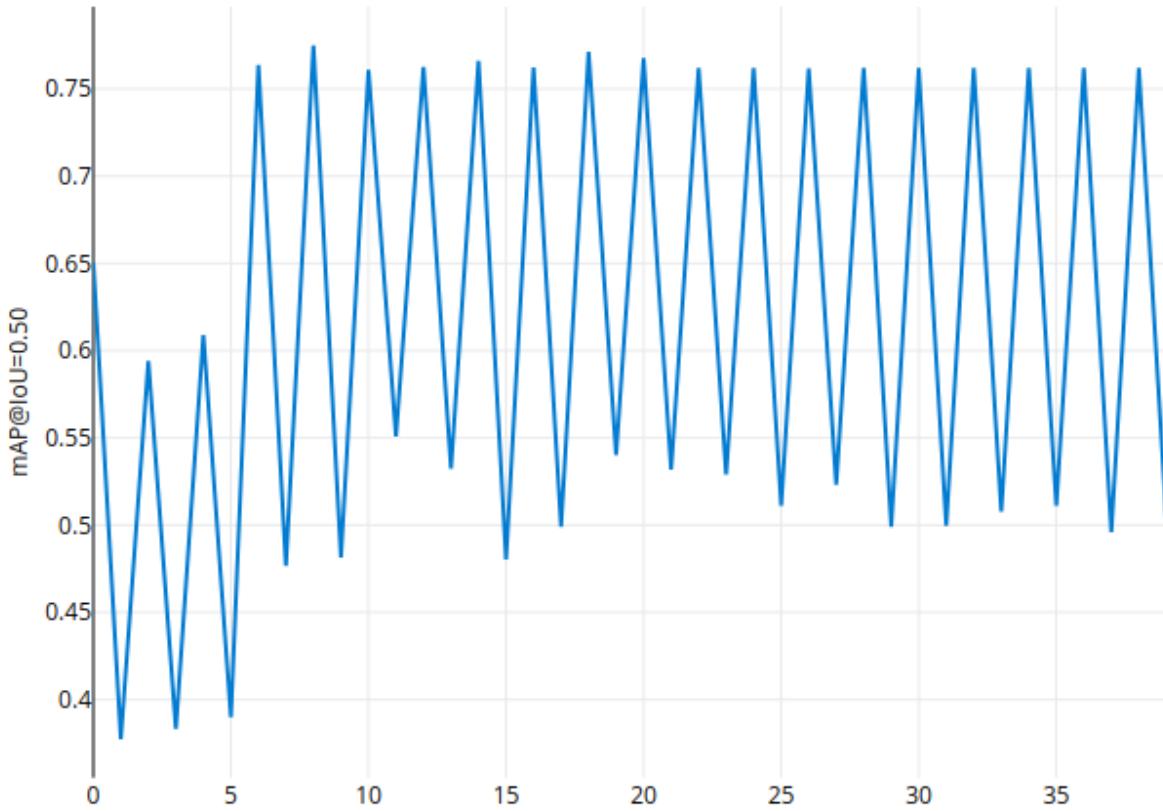
**mAP@IoU=0.50**



### DataSet MonkeyCats

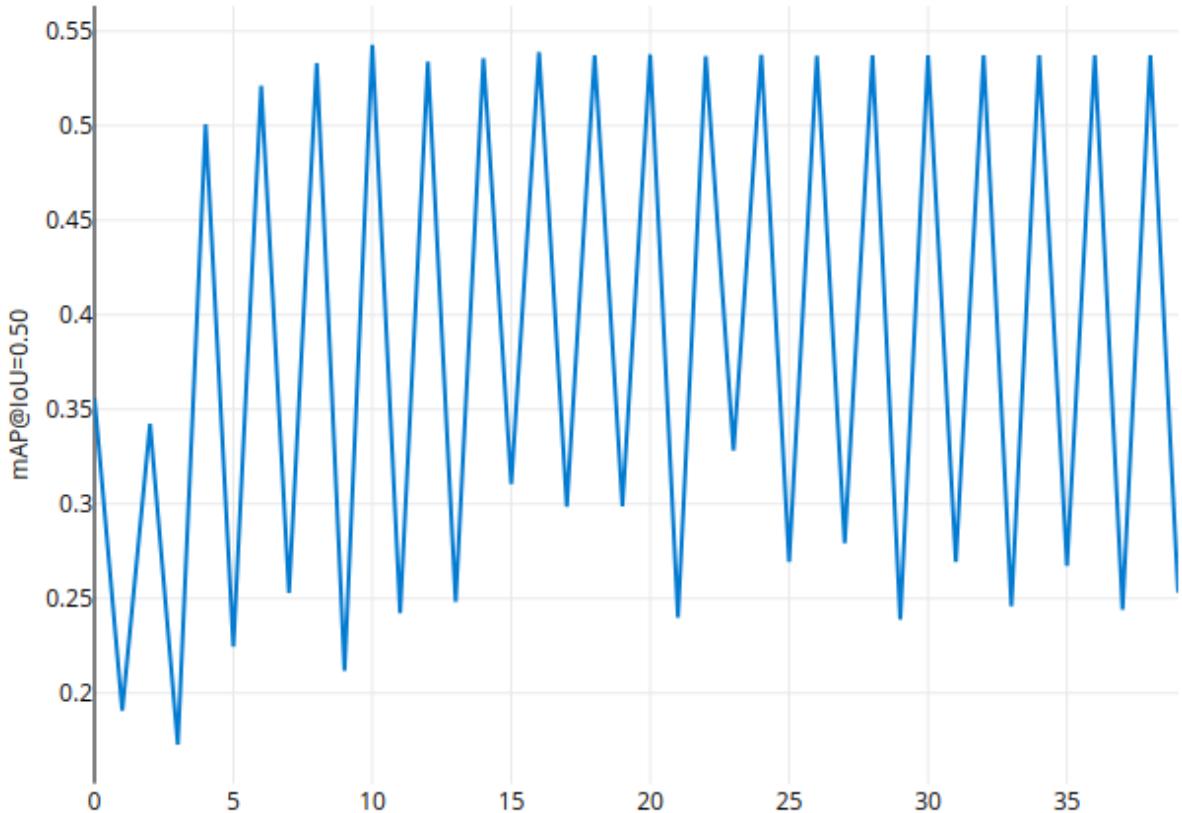
Arguments  
--data\_path . --workers 8 --learning\_rate 0.005 --epochs 20 --
anchor\_sizes 16,32,64,128,256,512 --anchor\_aspect\_ratios
0.25,0.5,1.0,2.0 --rpn\_nms\_thresh 0.5 --box\_nms\_thresh 0.3 --
box\_score\_thresh 0.1 --num\_classes 4 --dataset monkeyCats

Czas trwania: 3h 47m 4.478s

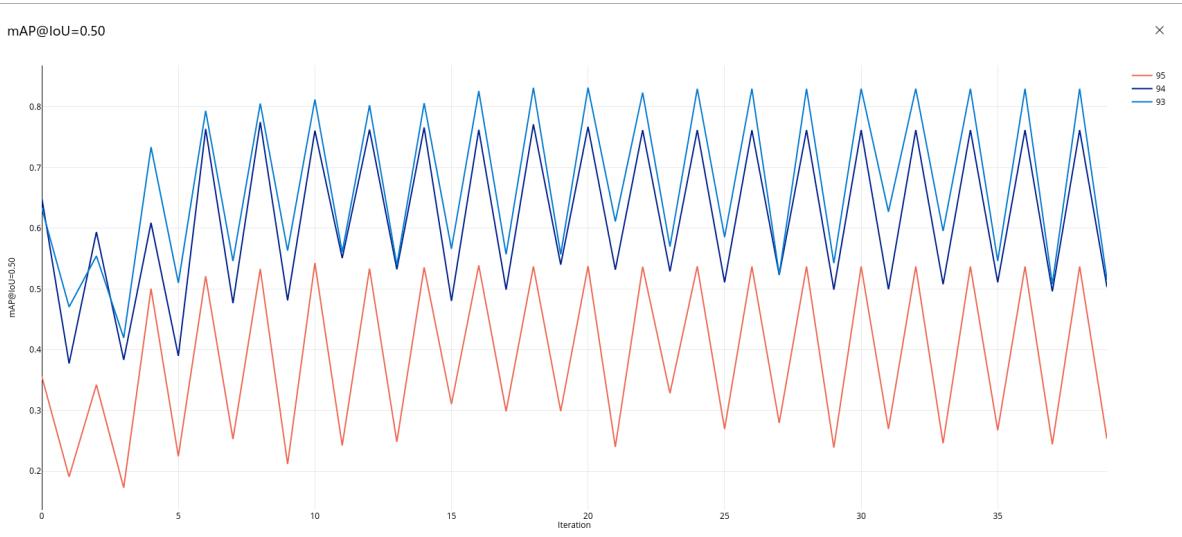
**mAP@IoU=0.50****DataSet PlayCards**

Arguments  
--data\_path . --workers 8 --learning\_rate 0.005 --epochs 20 --  
anchor\_sizes 16,32,64,128,256,512 --anchor\_aspect\_ratios  
0.25,0.5,1.0,2.0 --rpn\_nms\_thresh 0.5 --box\_nms\_thresh 0.3 --  
box\_score\_thresh 0.1 --num\_classes 7 --dataset PlayCards

Czas trwania: 2h 23m 4.841s

**mAP@IoU=0.50**

Poniżej pokazuje metryki wszystkich na jednym wykresie:

**93 - Fruit 94 - MonkeyCats 95 - PlayCards**

```
In [27]: datasetsList = ["Fruit", "monkeyCats", "PlayCards"]
classesList = [4, 4, 7]
num_epochs=20
metrics = []
for dataset,clas in zip(datasetsList,classesList):
    metrics.append(prepareEstimator (dataset,clas,num_epochs,dataset))
```

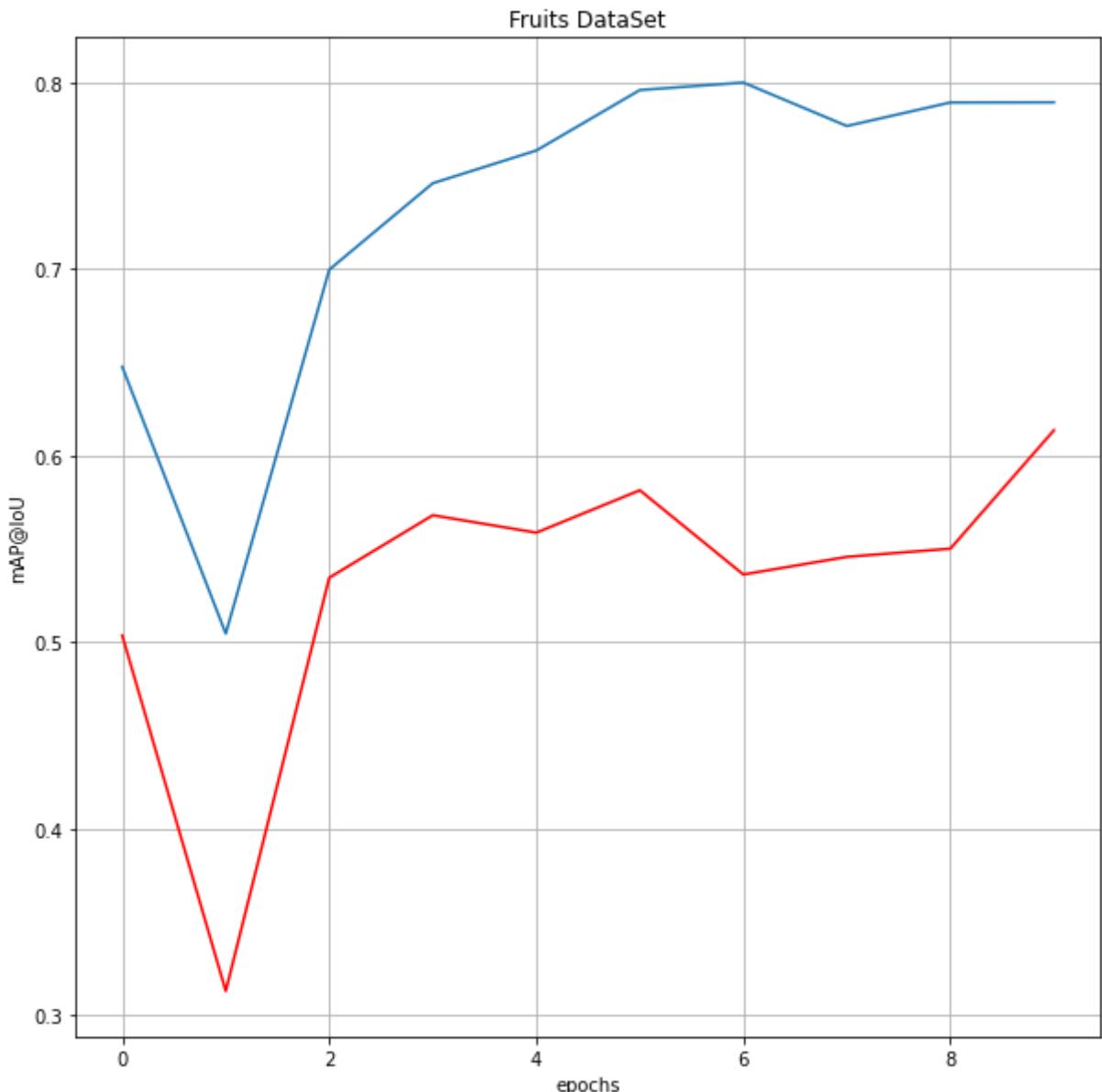
```
In [28]: with open('metrixs.txt', 'w') as f:
    for item in metrics:
        f.write("%s\n" % item)
```

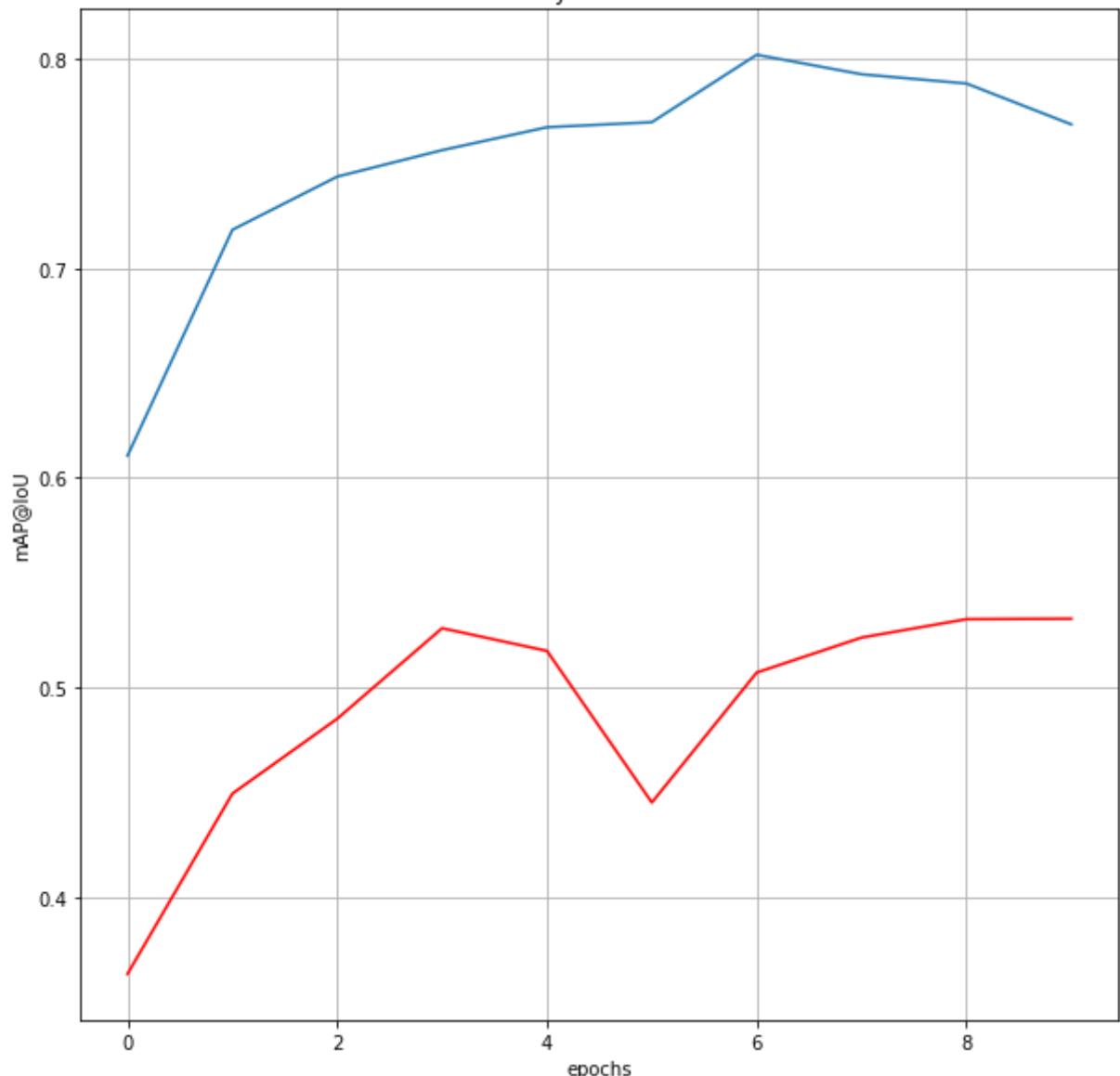
In [56]: #UWAGA! UWAGA! UWAGA!

## Dokładny opis datasetów

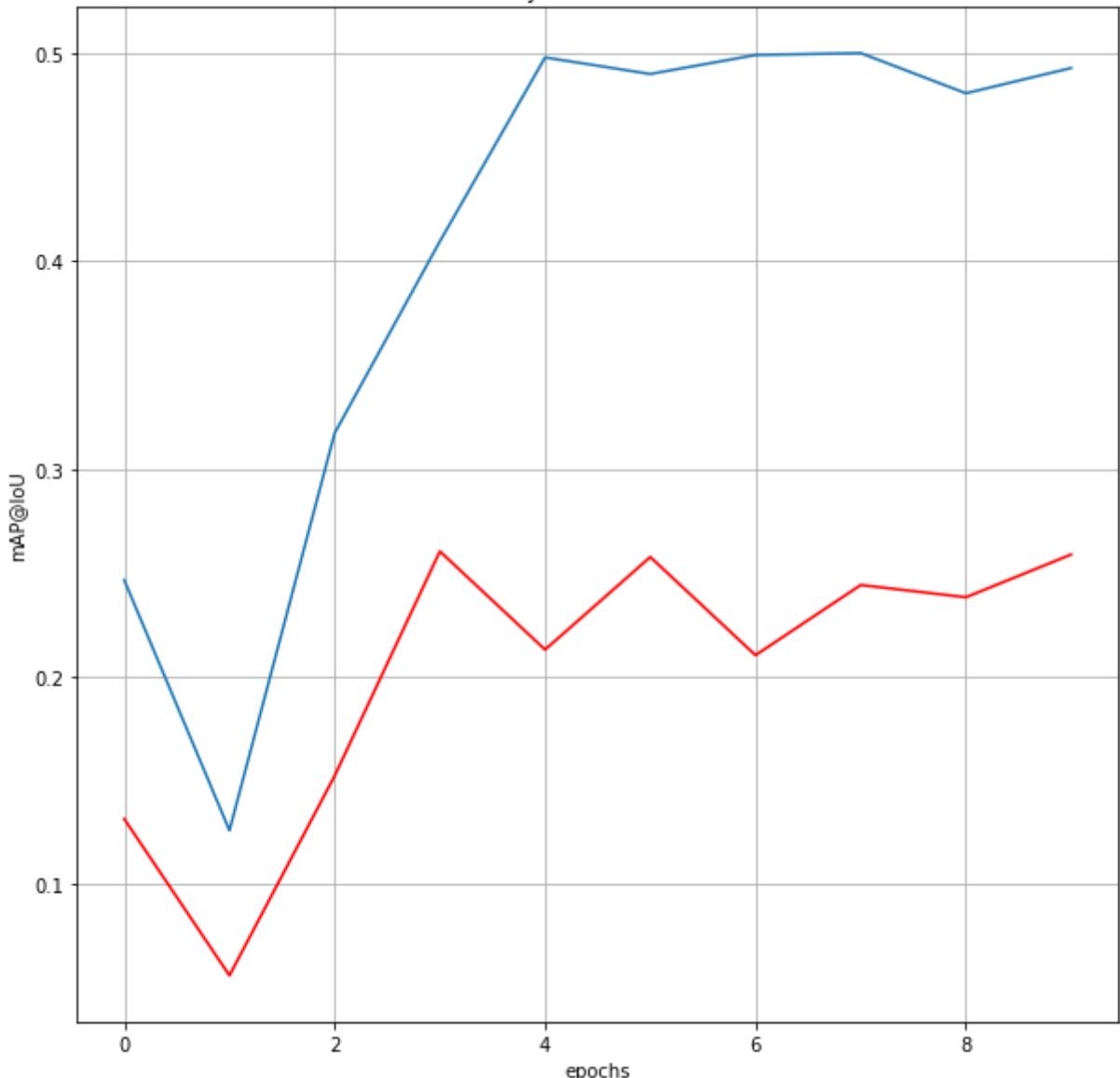
```
In [10]: from matplotlib import pyplot as plt
def plotmAP(fileName,dataSets):
    with open(fileName, 'r') as f:
        for line, title in zip(f.readlines(), dataSets):
            dic = eval(line)
            plt.figure(figsize=(10,10))
            plt.plot(dic['mAP@IoU=0.50'][::2])
            plt.plot(dic['mAP@IoU=0.50'][1::2], color="red")
            plt.title(title)
            plt.xlabel("epochs")
            plt.ylabel("mAP@IoU")
            plt.grid()
            plt.show()
```

```
In [11]: plotmAP("metrixs-1.txt",['Fruits DataSet', 'MonkeyCats DataSet', 'PlayCards'])
```



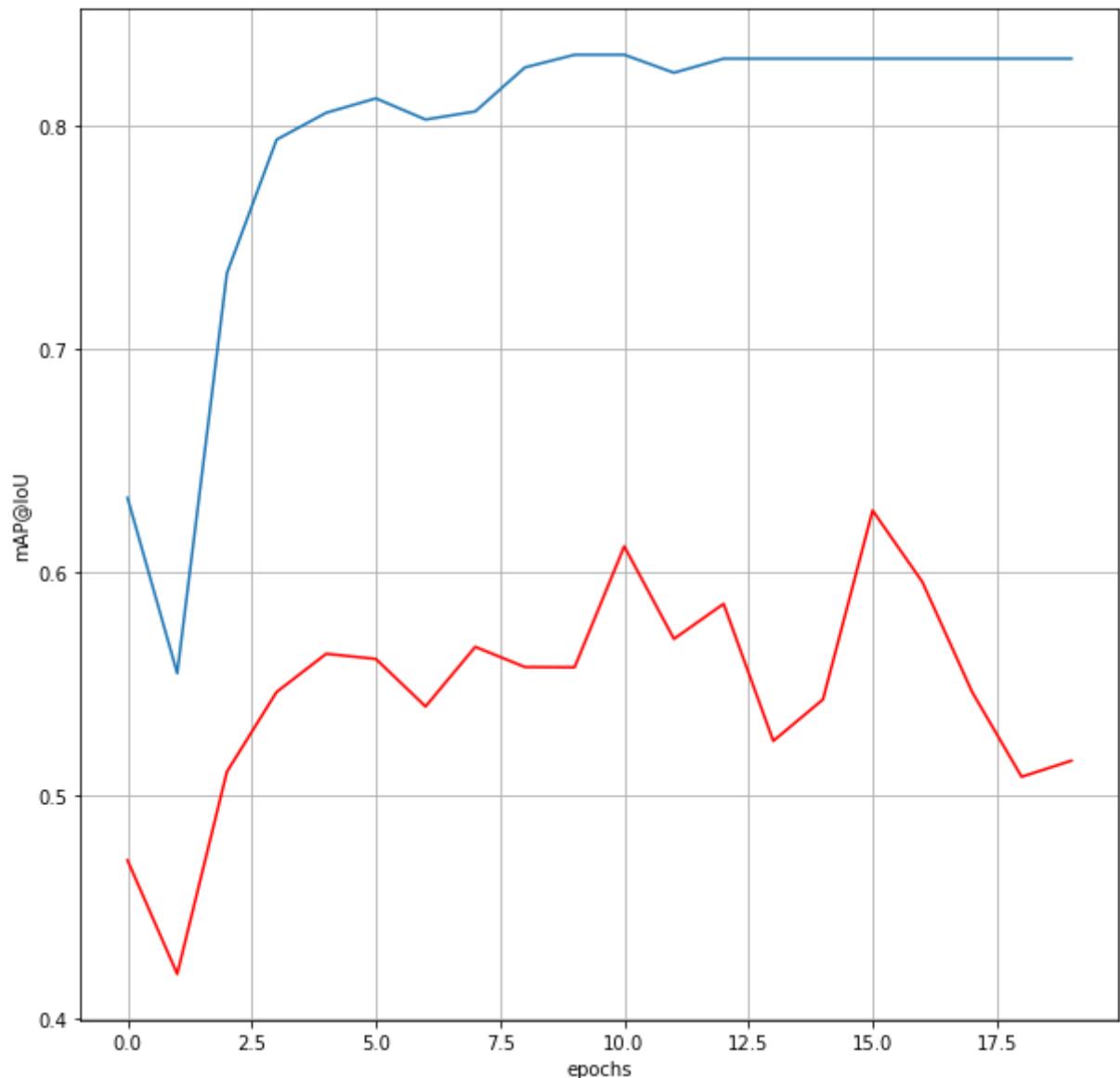
**MonkeyCats DataSet**

## PlayCards DataSet

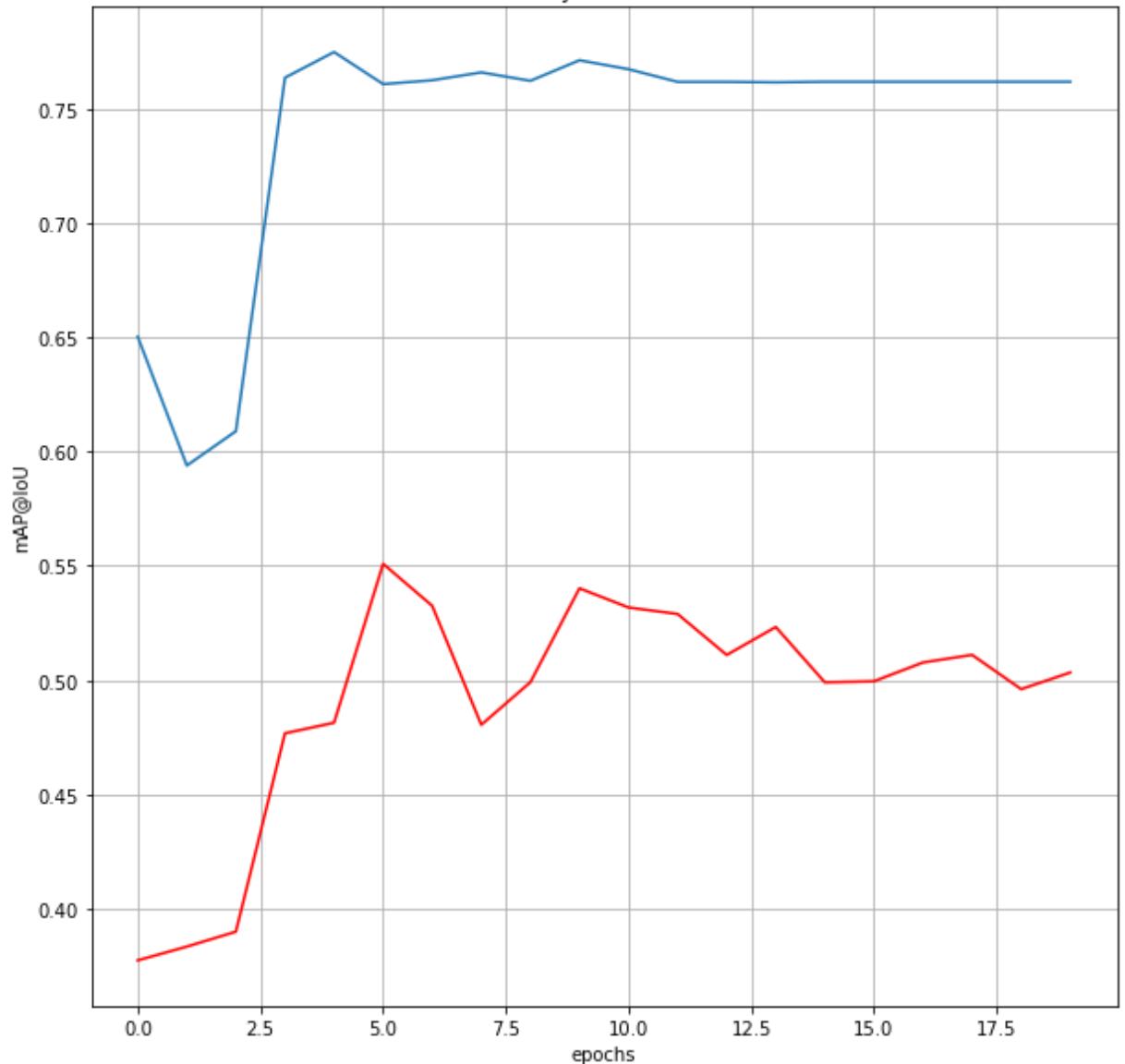


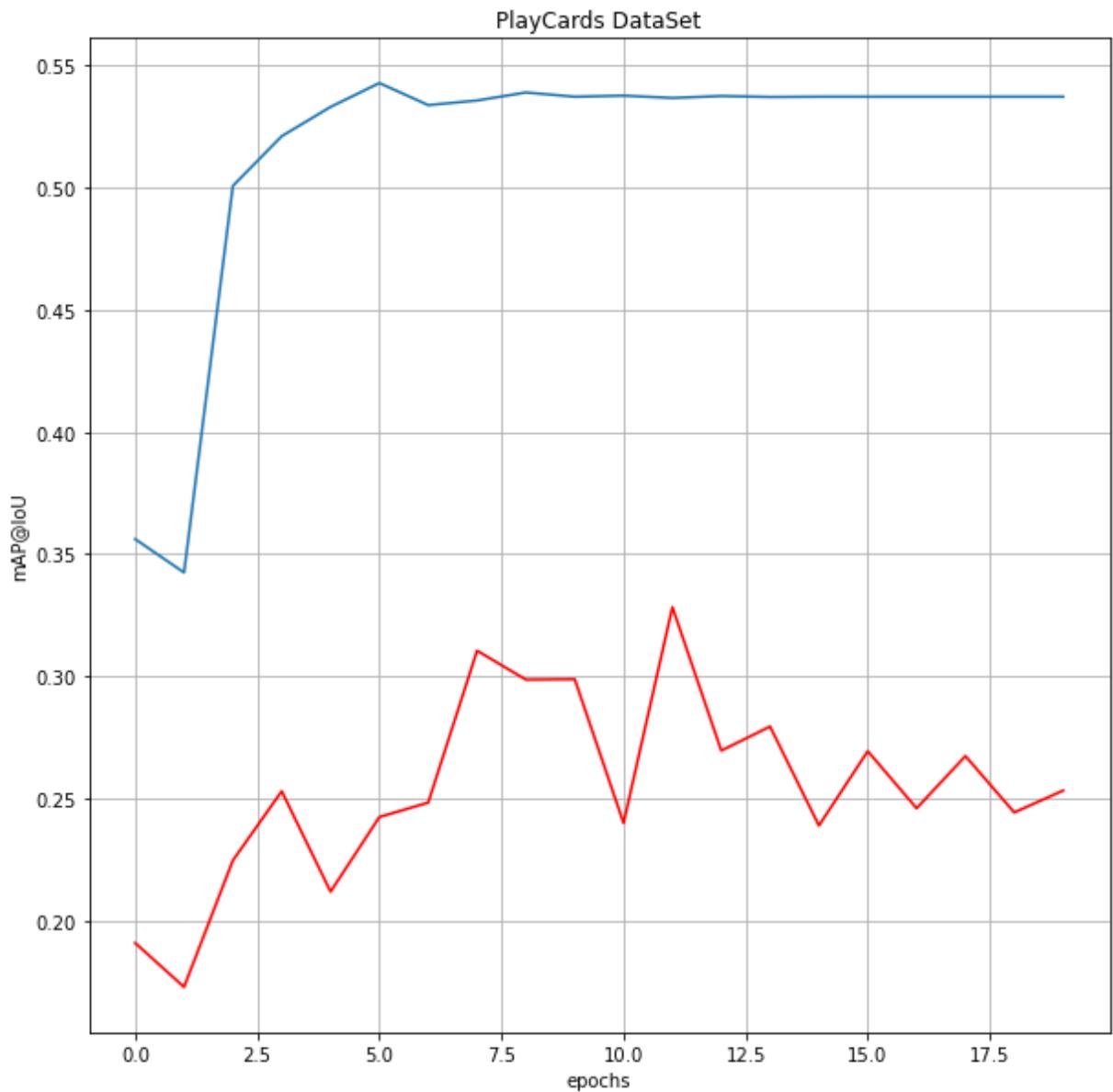
```
In [16]: plotmAP("metrixs.txt",['Fruits DataSet', 'MonkeyCats DataSet', 'PlayCards Da
```

## Fruits DataSet



## MonkeyCats DataSet





```
In [41]: def makePrediction(num_classes,dataset,model_path,index,labels_dict):
    anchor_sizes = "16,32,64,128,256,512"
    anchor_aspect_ratios = "0.25,0.5,1.0,2.0"
    rpn_nms_threshold = 0.5
    box_nms_threshold = 0.3
    box_score_threshold = 0.1
    num_box_detections = 100

    #prepare model to make prediction
    model = get_model(
        num_classes,
        anchor_sizes,
        anchor_aspect_ratios,
        rpn_nms_threshold,
        box_nms_threshold,
        box_score_threshold,
        num_box_detections,
    )
    #model_path = "model_latest.pth"
    model.load_state_dict(torch.load(model_path))
    device = torch.device("cuda") if torch.cuda.is_available() else torch.device("cpu")
    model.to(device)

    # Use a random subset of the data to visualize predictions on the images.
    data_path = "./scripts"
    #dataset = BuildDataset(data_path, "Fruit" ,get_transform(train=False), t
```

```
dataset = BuildDataset(data_path, dataset, get_transform(train=False), transform)
#prediction
img, _ = dataset[index]
model.eval()
with torch.no_grad():
    prediction = model([img.to(device)])
img = Image.fromarray(img.mul(255).permute(1, 2, 0).byte().numpy())
preds = prediction[0]["boxes"].cpu().numpy()
print(prediction[0]["scores"])
print(prediction[0]['labels'])
draw = ImageDraw.Draw(img)
for i in range(len(preds)):
    if prediction[0]["scores"][i].item() > 0.5:
        draw.rectangle(
            ((preds[i][0], preds[i][1]), (preds[i][2], preds[i][3])), outline="red")
        draw.text((preds[i][0], preds[i][1]), labels_dict[prediction[0][0][i]])
display(img)
```

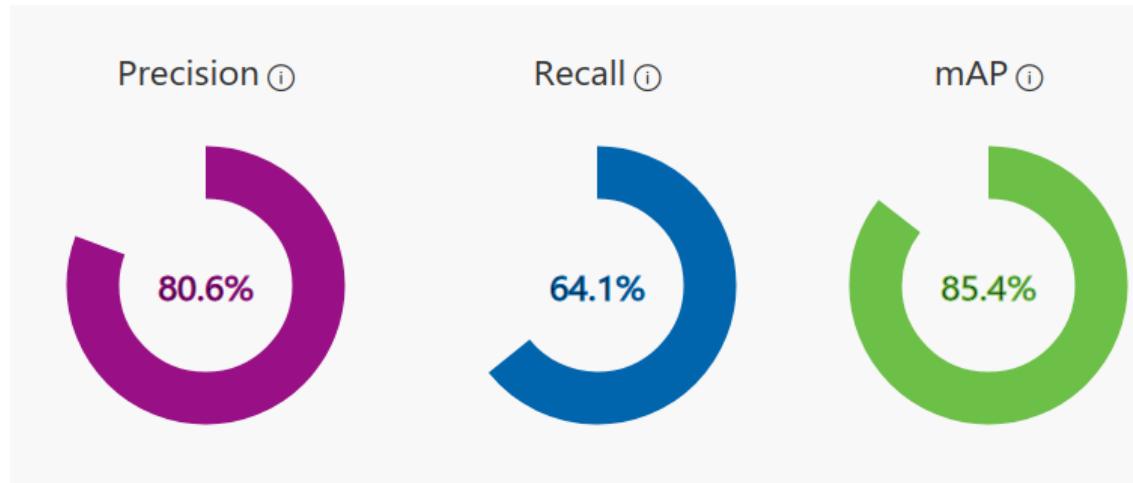
# Porównanie Mask R-CNN z Custom Vision

## Dataset Fruits

Po wytrenowaniu Custom Vision otrzymaliśmy takie wyniki dla datasetu Fruits:

- **Precision** - jak prawdopodobne jest, że jest to prawda?
- **Recall** - spośród tagów, które należy prawidłowo przewidzieć, jaki procent poprawnie znalazł twój model?
- **mAP** - (średnia dokładność) ogólna wydajność detektora obiektów we wszystkich znacznikach

Finished training on **1/6/2021, 4:06:59 PM** using **General** domain  
Iteration id: **fdbbab1ff-90fc-41ab-a150-f33e7fa85259**  
Published as: **detectModel**



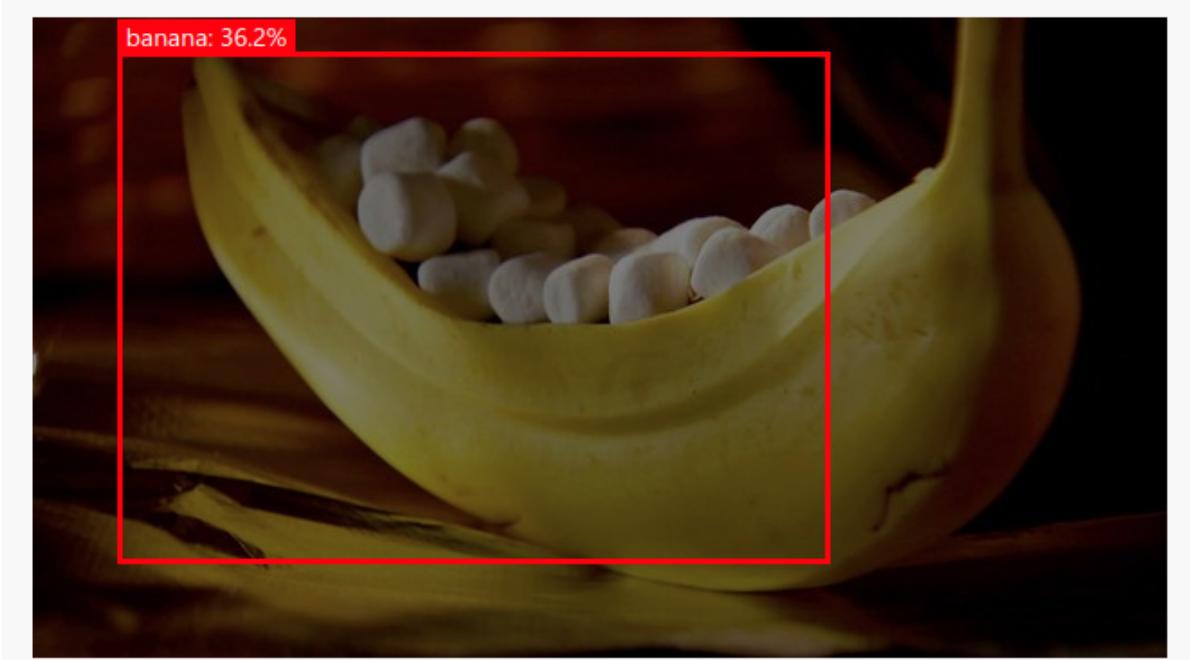
### Performance Per Tag

Tag	Precision	Recall	A.P.	Image count
apple	92.0%	88.5%	97.2%	80
orange	81.8%	85.7%	95.0%	75
banana	60.0%	29.0%	64.0%	83

Przykład wykrywania obiektu na zdjęciu. Jest pokazane, że to jest banan.

### Custom Vision

Zdjęcie dla Custom Vision:





## Mask R-CNN

In [42]: #Można pobawić się z predykcją

```
labels_dict = {1: 'apple', 2: 'orange', 3: 'banana'}  
makePrediction(4,"Fruit","Fruitmodel_latest.pth",25,labels_dict)
```

```
tensor([0.8869, 0.4075, 0.2982], device='cuda:0')  
tensor([3, 1, 3], device='cuda:0')
```



## Wnioski z przykładu

Zdjęcie z bananem wyszło nam, że dla Custom Vision jest 36.2% prawdopodobieństwa, natomiast dla Mask R-CNN wychodzi 88.69% prawdopodobieństwa.

**W tym wypadku Mask R-CNN pokazał siebie z lepszej strony**

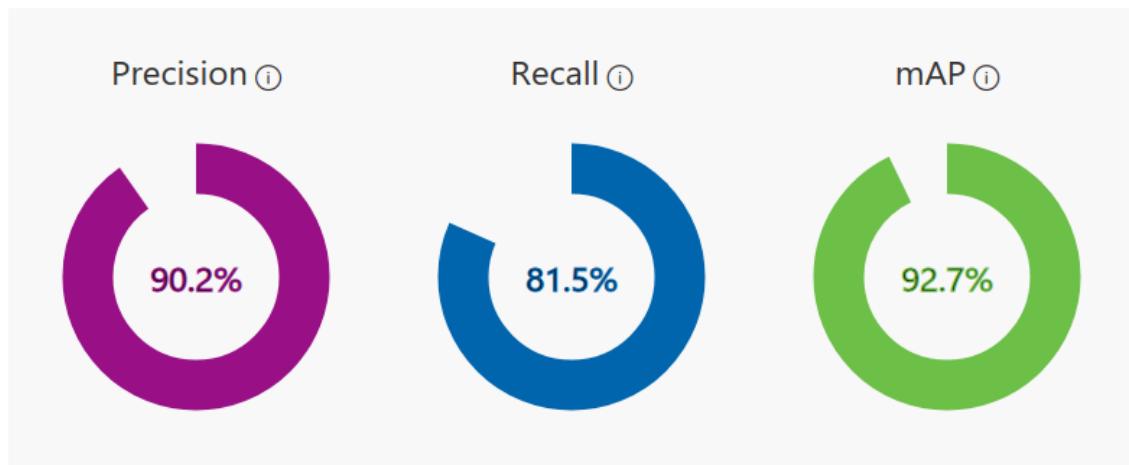
## Dataset Monkey Cat Dog

Po wytrenowaniu Custom Vision otrzymaliśmy takie wyniki dla datasetu Monkey Cat Dog:

- **Precision** - jak prawdopodobne jest, że jest to prawda?

- **Recall** - spośród tagów, które należy prawidłowo przewidzieć, jaki procent poprawnie znalazł twój model?
- **mAP** - (średnia dokładność) ogólna wydajność detektora obiektów we wszystkich znacznikach

Finished training on 12/21/2020, 1:22:06 PM using **General** domain  
 Iteration id: 0a93b42e-3f8d-4cd1-b4b3-504c9efbd7a  
 Published as: **detectModel**

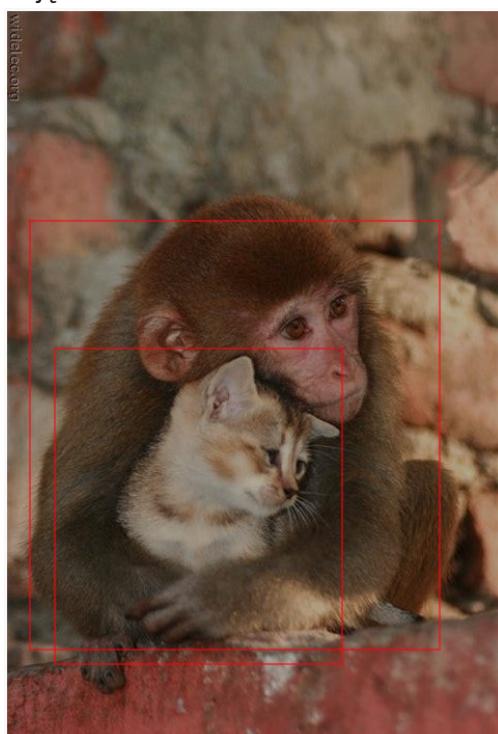


### Performance Per Tag

Tag	Precision	▲	Recall	A.P.	Image count
monkey	92.5%		81.9%	93.8%	172
cat	89.9%		79.2%	91.4%	213
dog	87.8%		83.7%	92.9%	186

## Custom Vision

Zdjęcie dla Custom Vision:



+
To create an object, hover and select the region in the image

Only show suggested objects if the probability is above the selected threshold.

Threshold Value: 15%

**Predictions**

Predictions are shown in red

Tag	Probability
monkey	86.6%
cat	74.2%



Prediction:

## Mask R-CNN

```
In [45]: labels_dict = {1: "cat", 2: "dog", 3: "monkey"}  
makePrediction(4,"monkeyCats","monkeyCatsmodel_latest1.pth",11,labels_dict)  
  
tensor([0.9081, 0.4157], device='cuda:0')  
tensor([3, 1], device='cuda:0')
```



## Wnioski z przykładu

Zdjęcie z małpą oraz kotem wyszło nam, że dla Custom Vision Kot(**74.2%**) a małpa(**86.6%**) prawdopodobieństwa, natomiast dla Mask R-CNN nie wykrał w ogóle kota, a tylko małpę(**90%**)

**W tym wypadku Custom Vision pokazał siebie z lepszej strony** Czyli dla wykrywanie jednego obiektu na zdjęciu radzi sobie lepiej Mask R-CNN, natomiast Custom Vision wykrawa lepiej zdjęcia z dwoma lub więcej obiektyami.

## Dataset Play Cards

Po wytrenowaniu Custom Vision otrzymaliśmy takie wyniki dla datasetu Play Cards:

- **Precision** - jak prawdopodobne jest, że jest to prawda?

- **Recall** - spośród tagów, które należy prawidłowo przewidzieć, jaki procent poprawnie znalazły się w twoim modelu?
  - **mAP** - średnia dokładność) ogólna wydajność detektora obiektów we wszystkich znacznikach

## Mask R-CNN

# Wnioski z przykładu

Na zbiorze danych Play Cards wykryło dwa asy (czyli pozytywnie). Z powodu zmiany predykcji z 4 na 7.

```
In [54]: labels_dict = {1: 'ace', 2: 'king', 3: 'queen', 4: 'jack', 5: 'ten', 6: 'nine'
makePrediction(7,"PlayCards","PlayCardsmodel_latest1.pth",55,labels_dict)

tensor([0.8226, 0.5251, 0.1733, 0.1463], device='cuda:0')
tensor([1, 1, 5, 6], device='cuda:0')
```



```
In [14]: num_classes = 4  
anchor_sizes = "16,32,64,128,256,512"  
anchor_aspect_ratios = "0.25,0.5,1.0,2.0"  
rpn_nms_threshold = 0.5  
box_nms_threshold = 0.3  
box_score_threshold = 0.1  
num_box_detections = 100
```

```
In [15]: # Load Mask RCNN model  
model = get_model(
```

```

        num_classes,
        anchor_sizes,
        anchor_aspect_ratios,
        rpn_nms_threshold,
        box_nms_threshold,
        box_score_threshold,
        num_box_detections,
    )
)
```

```
In [16]: model_path = "model_latest.pth"
model.load_state_dict(torch.load(model_path))
device = torch.device("cuda") if torch.cuda.is_available() else torch.device("cpu")
model.to(device)
```

```
In [17]: # Use a random subset of the data to visualize predictions on the images.
data_path = "./scripts"
dataset = BuildDataset(data_path, "Fruit", get_transform(train=False), train=True)
# indices = torch.randperm(len(dataset)).tolist()
# dataset = torch.utils.data.Subset(dataset, indices[-50:])
```

```
In [18]: # for i in range(5):
#     img, _ = dataset[i]
#     model.eval()
#     with torch.no_grad():
#         prediction = model([img.to(device)])
#         img = Image.fromarray(img.mul(255).permute(1, 2, 0).byte().numpy())
#         preds = prediction[0]["boxes"].cpu().numpy()
#         print(prediction[0]["scores"])
#         draw = ImageDraw.Draw(img)
#         for i in range(len(preds)):
#             draw.rectangle(
#                 ((preds[i][0], preds[i][1]), (preds[i][2], preds[i][3])), outline="red")
#         display(img)
```

```
In [19]: img, _ = dataset[1]
model.eval()
with torch.no_grad():
    prediction = model([img.to(device)])
img = Image.fromarray(img.mul(255).permute(1, 2, 0).byte().numpy())
preds = prediction[0]["boxes"].cpu().numpy()
print(prediction[0])

{'boxes': tensor([[ 9.2525,  24.6268, 350.0000, 340.1003]], device='cuda:0'), 'labels': tensor([1], device='cuda:0'), 'scores': tensor([0.6061], device='cuda:0')}
```

```
In [20]: labels_dict = {1: 'apple', 2: 'orange', 3: 'banana'}

#labels_dict = {1: 'ace', 2: 'king', 3: 'queen', 4: 'jack', 5: 'ten', 6: 'nir'}
#labels_dict = {1: "cat", 2: "dog", 3: "monkey"}
```

## Custom Vision

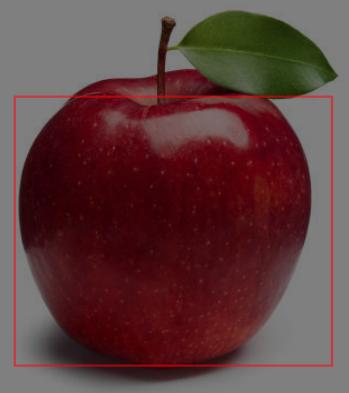
Zdjęcie dla Custom Vision:

Using model trained in  
Iteration  
Iteration 1

Predicted Object Threshold  
Only show suggested objects if the probability is above the selected threshold.  
Threshold Value: 15%

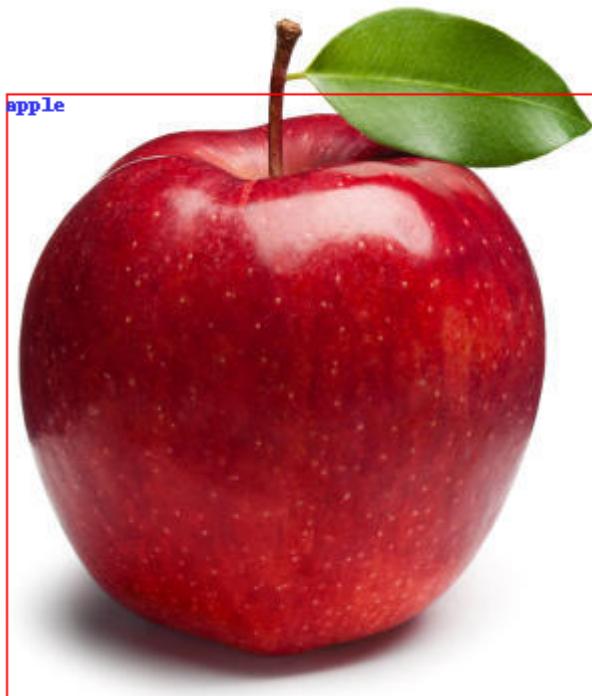
Predictions  
Predictions are shown in red

Tag	Probability
apple	62.4%



```
In [21]: img, _ = dataset[12]
model.eval()
with torch.no_grad():
    prediction = model([img.to(device)])
img = Image.fromarray(img.mul(255).permute(1, 2, 0).byte().numpy())
preds = prediction[0]["boxes"].cpu().numpy()
print(prediction[0]["scores"])
print(prediction[0]['labels'])
draw = ImageDraw.Draw(img)
for i in range(len(preds)):
    if prediction[0]["scores"][i].item() > 0.5:
        draw.rectangle(
            ((preds[i][0], preds[i][1]), (preds[i][2], preds[i][3])), outline="red")
        draw.text((preds[i][0], preds[i][1]), labels_dict[prediction[0]['labels'][i]])
display(img)

tensor([0.8912], device='cuda:0')
tensor([1], device='cuda:0')
```



## Wnioski z przykładu

Zdjęcie z jabłkiem wyszło nam, że dla Custom Vision (**62,4%**) prawdopodobieństwa, natomiast dla Mask R-CNN (**89%**)

**W tym wypadku Mask R-CNN pokazał siebie z lepszej strony** Czyli dla wykrywanie jednego obiektu na zdjęciu radzi sobie lepiej Mask R-CNN.

## Custom Vision

Zdjęcie dla Custom Vision:

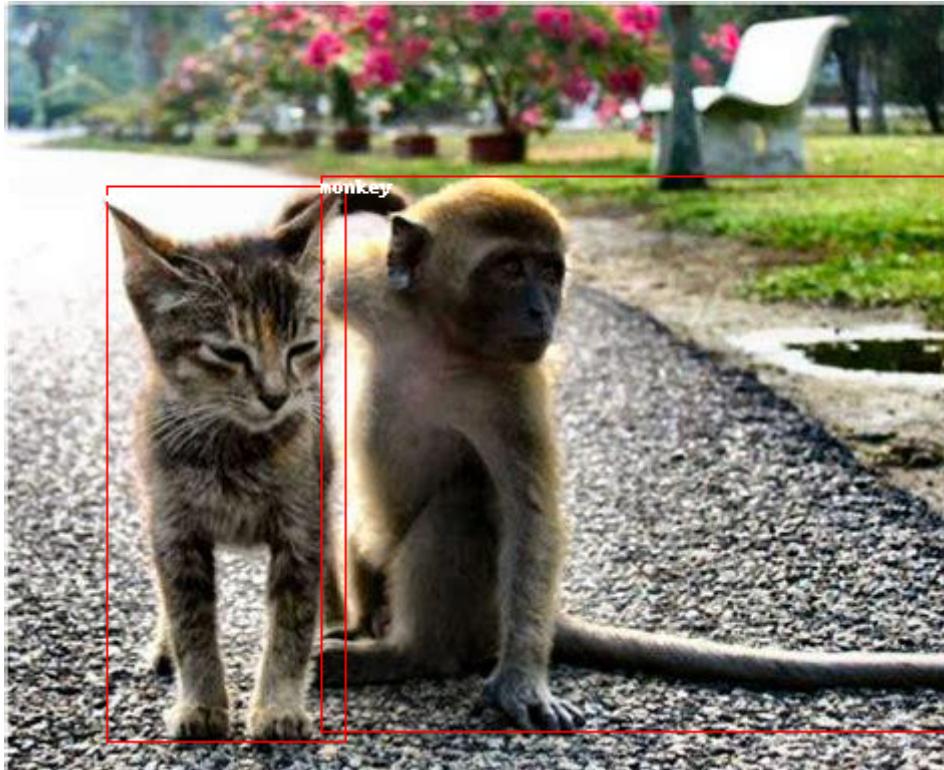
Using model trained in  
Iteration  
Iteration 1

Predicted Object Threshold  
Only show suggested objects if the probability is above the selected threshold.  
Threshold Value: 15%

Predictions  
Predictions are shown in red

Tag	Probability
cat	88.6%
monkey	25.6%

```
In [25]: import xml.etree.ElementTree as ET
#tree = ET.parse('scripts/Data/PlayCards/AnnotationsTest/cam_image45.xml')
tree = ET.parse('scripts/Data/monkeyCats/AnnotationsTest/cats_and_monkeys_029.xml')
t_root = tree.getroot()
#img = Image.open('scripts/Data/PlayCards/JPEGImagesTest/cam_image45.jpg')
img = Image.open('scripts/Data/monkeyCats/JPEGImagesTest/cats_and_monkeys_029.jpg')
draw = ImageDraw.Draw(img)
for obj in t_root.findall("object"):
    bnd_box = obj.find("bndbox")
    xmin = float(bnd_box.find("xmin").text)
    xmax = float(bnd_box.find("xmax").text)
    ymin = float(bnd_box.find("ymin").text)
    ymax = float(bnd_box.find("ymax").text)
    label_name = str(obj.find("name").text)
    draw.rectangle(((xmin, ymin), (xmax, ymax)), outline="red")
    draw.text((xmin, ymin), label_name, fill=(255,255,255,128))
display(img)
```



## Wnioski z przykładu

Zdjęcie z małpą oraz kotem wyszło nam, że dla Custom Vision Kot(**88.6%**) a małpa(**25.6%**) prawdopodobieństwa, natomiast dla Mask R-CNN dla kota jest prawie tak samo, natomiast małpa wykryła się lepiej od Custom Vision.

**W tym wypadku Mask R-CNN pokazał siebie z lepszej strony**

## Ogólne wnioski & porównanie

Wykrywanie obiektów to zadanie w wizji komputerowej, które obejmuje identyfikację obecności, lokalizacji i typu jednego lub więcej obiektów na danym zdjęciu.

Stworzyliśmy wytrenerowane modele dla różnych typów danych (3 zbiory) dla Mask R-CNN (Own model) oraz Azure Custom Vision.

## Maska R-CNN do wykrywania obiektów

Maska R-CNN - rozszerzenie szybszego R-CNN, które dodaje model wyjściowy do przewidywania maski dla każdego wykrytego obiektu.

- Maska R-CNN ma dodatkową gałąź do przewidywania masek segmentacji w każdym regionie zainteresowania (RoI) w sposób piksel-piksel

Model maski R-CNN jest podzielony na dwie części:

- Sieć propozycji regionów (RPN) do proponowania ramek ograniczających obiekty kandydatów.
- Binarny klasyfikator maski do generowania maski dla każdej klasy

## Azure Custom Vision

- Custom Vision obsługuje następujące domeny do wykrywania obiektów: General, Logo, Products i Compact.
- Szybka implementacja, dojść prosta w porównaniu do Mask R-CNN.

Azure Custom Vision to szybki i łatwy sposób tworzenia i wdrażania modeli klasyfikacji i wykrywania obiektów. Portal internetowy umożliwia eksperymentowanie ze zbiorem danych bez użycia kodu, ale jeśli chcemy zaimplementować. Przesłaliśmy dane do Azure Custom Vision, następnie wytrenerowaliśmy model.

## Porównanie

### Średni procent wykrywania (precyzja detektora obiektów w znalezieniu)

Dataset	MaskaR-CNN	Azure Custom Vision
Fruit	55% - 80%	64% - 97%
MonkeyCats	49% - 75%	91% - 94%
PlayCards	24% - 54%	

### Czas trwania

Dataset	MaskaR-CNN	Azure Custom Vision
Fruit	1h 54m 18.30s	
MonkeyCats	3h 47m 4.478s	
PlayCards	2h 23m 4.841s	

Mask-RCNN to kolejna ewolucja modeli wykrywania obiektów, które umożliwiają wykrywanie z większą precyzacją. To tylko mały przykład tego, co możemy osiągnąć dzięki temu wspaniałemu modelowi.

### Plusy Azure Custom Vision:

- Nie potrzebujemy dużo czasu, aby opracować algorytmy dla obrazu
- Algorytmy będą ciągle optymalizowane i ewolucjonowane przez analityka danych Microsoftu
- Wysoka dokładność

### Minusy Azure Custom Vision:

- Płatne
- Nie wiemy, czy jest odpowiedni dla wszystkich sytuacji dla Klienta.
- Ograniczenie się do algorytmów dostarczanych przez Microsoft i możemy zmobilizować bardzo niewiele parametrów do dopasowania modelu

Usługa platformy Azure jest tak użytecznym sposobem uczenia modelu uczenia maszynowego, więc w tej usłudze nie ma się czego nie lubić. Zdecydowanie polecamy innym korzystanie z usługi Azure Custom Vision. Model możemy trenować po prostu klikając i nie wymagamy znajomości algorytmów uczenia maszynowego.

Więc naszym zdaniem Azure Custom Vision jest lepszy!

## Polecamy Microsoft Azure Custom Vision !!!



In [ ]: