

# Raport

January 8, 2021

## Contents

<b>1 Custom Vision vs Own Model for object detection</b>	<b>1</b>
1.1 Cel :	1
1.2 Zbiory danych:	2
<b>2 Azure Machine Learning Configuration</b>	<b>2</b>
2.1 Ładowanie datasetów Mask R-CNN	4
2.2 Wykresy mAP dla Mask R-CNN	6
<b>3 Porównanie Mask R-CNN z Custom Vision</b>	<b>12</b>
3.1 Dataset Fruits	12
3.1.1 Custom Vision	14
3.1.2 Mask R-CNN	14
3.1.3 Wnioski z przykładu	15
3.2 Dataset Monkey Cat Dog	16
3.2.1 Custom Vision	17
3.2.2 Mask R-CNN	17
3.2.3 Wnioski z przykładu	19
3.3 Dataset Play Cards	19
3.3.1 Mask R-CNN	19
3.3.2 Wnioski z przykładu	20
3.3.3 Custom Vision	22
3.3.4 Wnioski z przykładu	24
<b>4 Ogólne wnioski &amp; porównanie</b>	<b>24</b>
4.1 Mask R-CNN do wykrywania obiektów	24
4.2 Azure Custom Vision	24
4.3 Porównanie	24
4.4 Podsumowanie	26

## 1 Custom Vision vs Own Model for object detection

### 1.1 Cel :

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

- Azure Custom Vision
- Mask R-CNN

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

## 1.2 Zbiory danych:

W celu porównania 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 modeli na wyżej omówionych zbiorach danych.

## 2 Azure Machine Learning Configuration

Poniżej zostały dostarczone wytrenowane modeli na wyżej omówionych zbiorach danych.

Podłączanie bibliotek i pakietów:

```
[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 podłączenie do Workspace Azure. Wyświetlenie wersji:

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

You are currently using version 1.17.0 of the Azure ML SDK

Konfiguracja Azure oraz environment

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

Znajdowanie zasobu oraz podłączenie po przez kluczy oraz subskrypcje:

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

```
[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)
```

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

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

```
[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()`)

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

Poniżej jest importowanie bibliotek oraz podłączenie ścieżki do plików, które wskazują na moduły:

```
[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

## 2.1 Ładowanie datasetów Mask R-CNN

### 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

### 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

### DataSet PlayCards

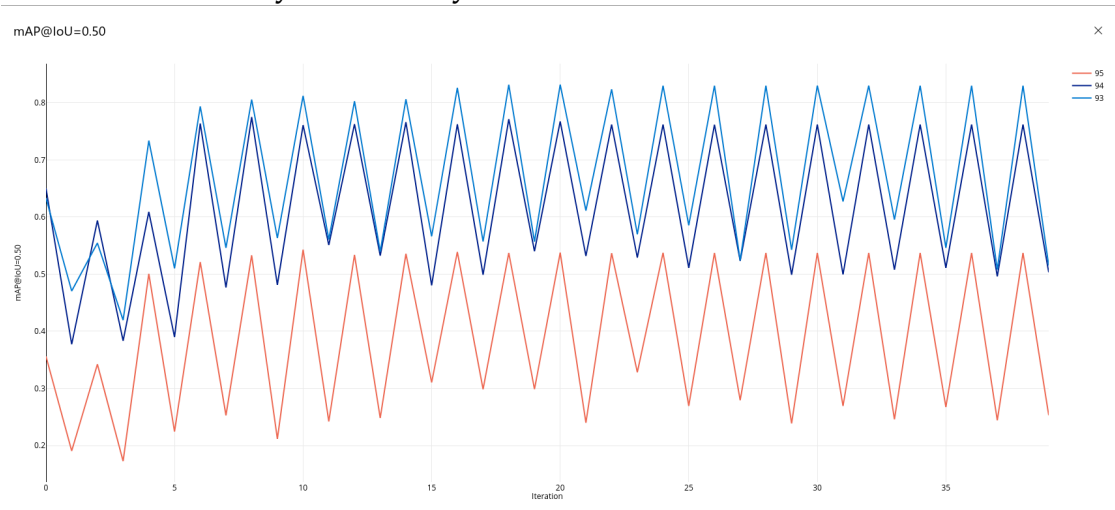
#### Arguments

```
--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

Poniżej pokazuje metryki wszystkich (trenowanych oraz testujących danych razem) na jednym wykresie, żeby zobaczyć rozbieżności:

### 93 - Fruit 94 - MonkeyCats 95 - PlayCards



```
[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))
```

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

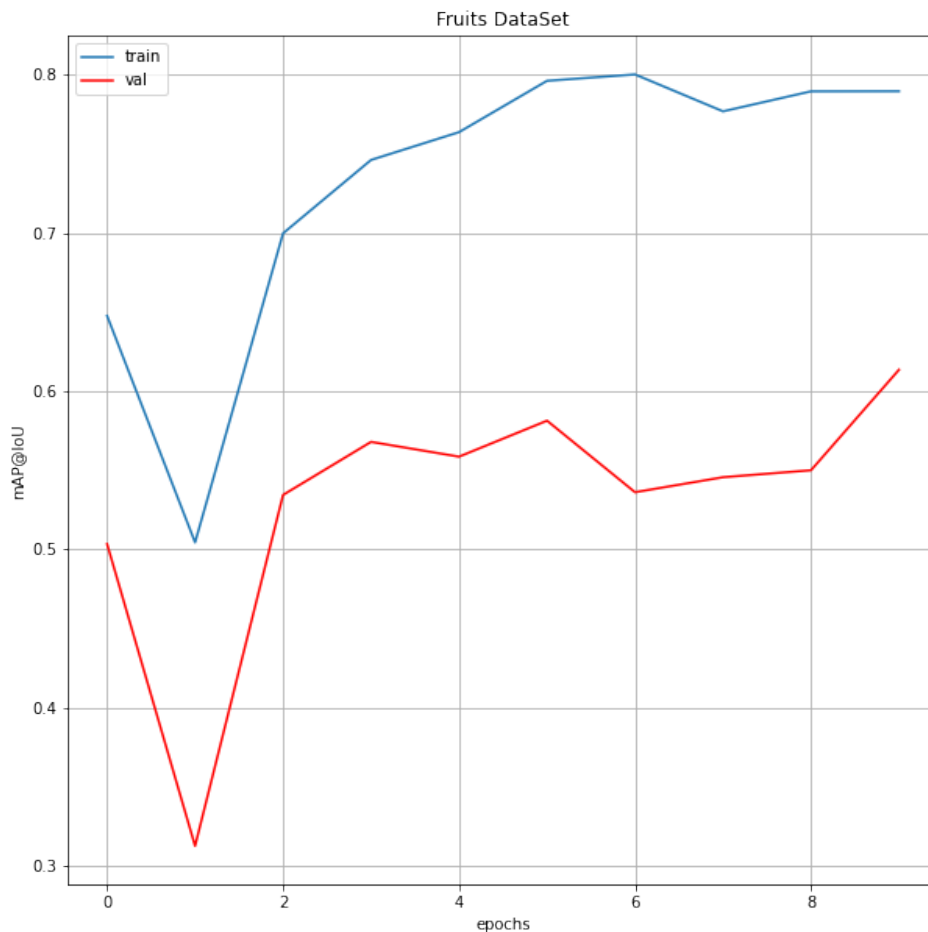
```
[56]: #UWAGA! UWAGA! UWAGA!
```

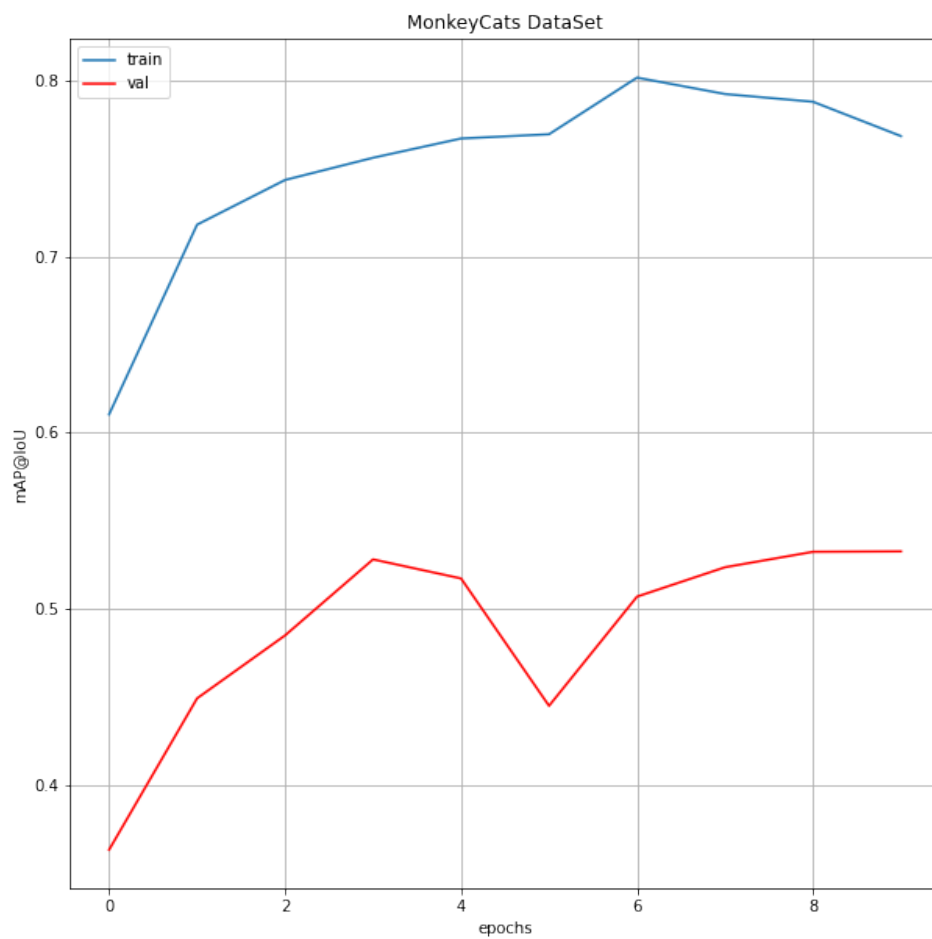
## 2.2 Wykresy mAP dla Mask R-CNN

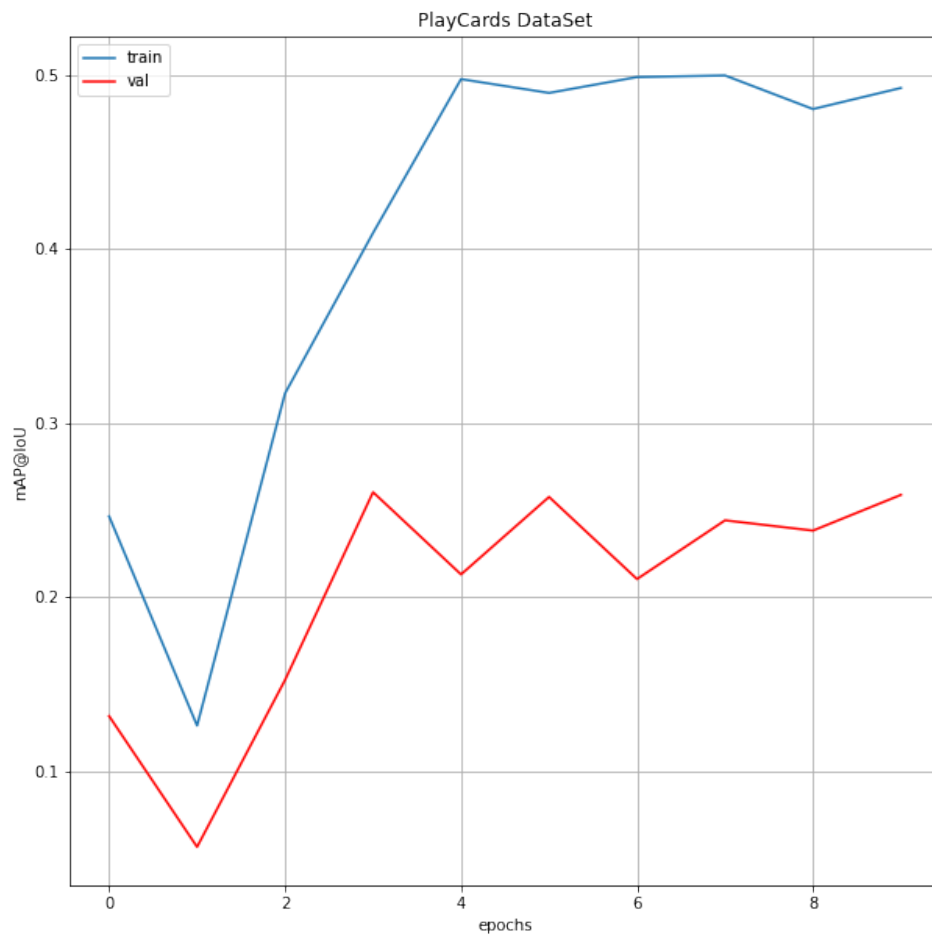
W tym rozdziale znajdują się wykresy reprezentujące trening oraz walidację modelu dla 10 oraz 20 epok trenowania modelu Mask R-CNN. Z wykresów dla 20 epok widać że po 10 epokach następuje przeuczenie modelu i nie ma sens kontynuować process uczenia. Najprawdopodobniej jest to związane ze zbyt małym zbiorem danych treningowych.

- **mAP** - (średnia dokładność) ogólna wydajność detektora obiektów we wszystkich znacznikach

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

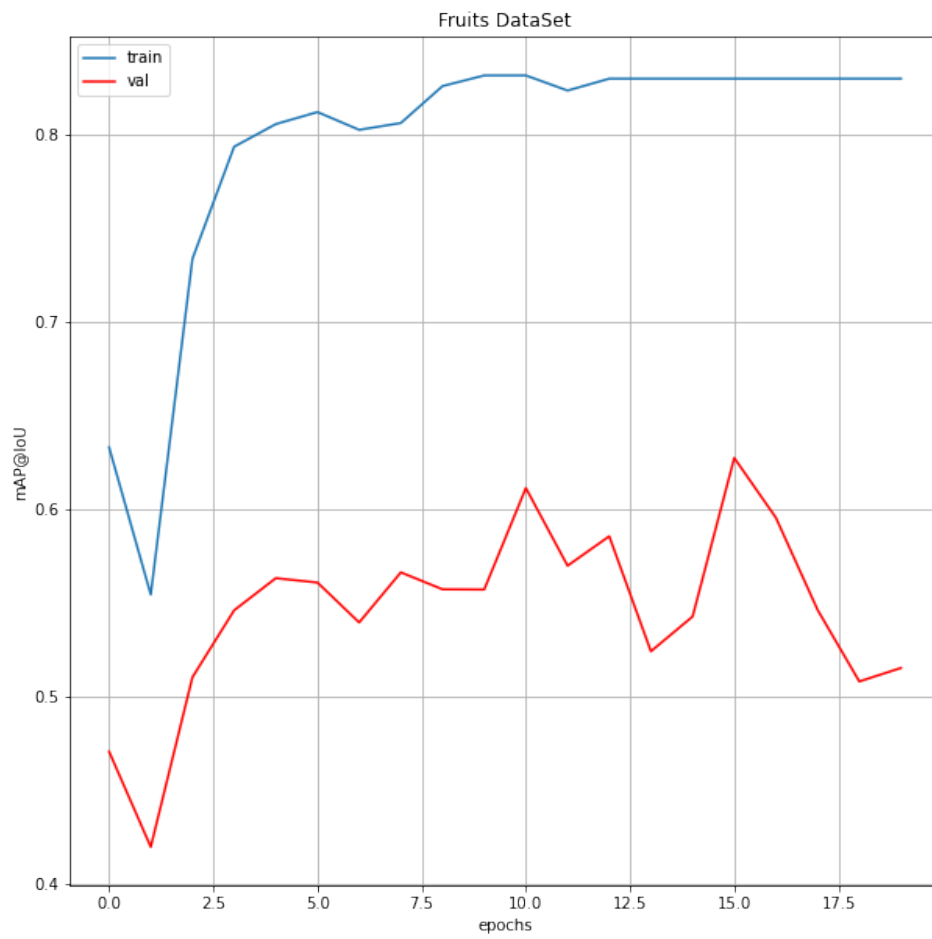


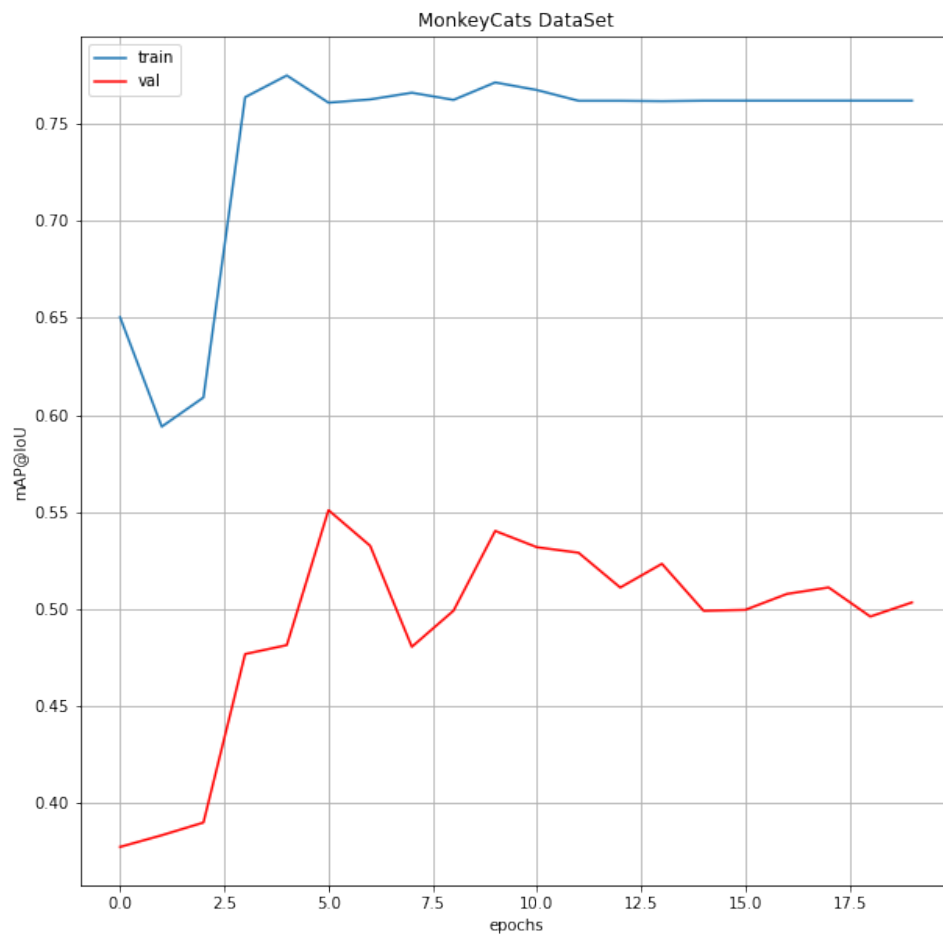


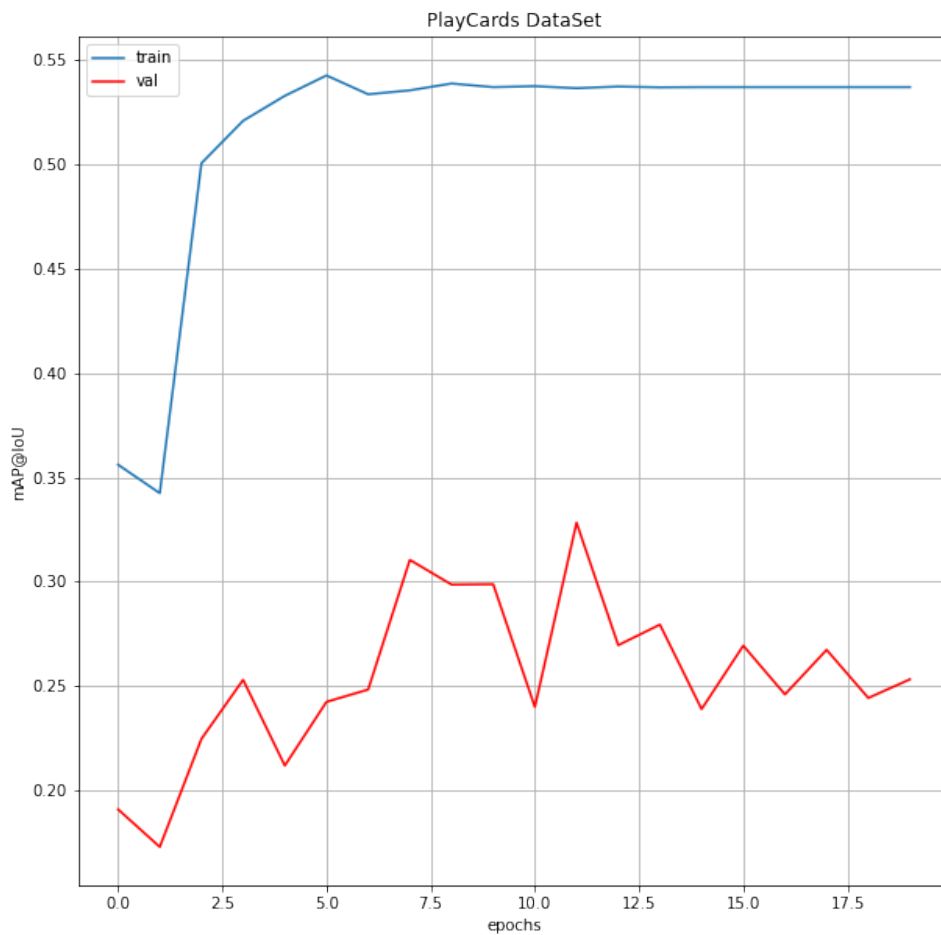


```
[16]: plotmAP("metrixs.txt",['Fruits DataSet', 'MonkeyCats DataSet', 'PlayCards_
    ↳DataSet'])
```









```
[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),
→train=False)
    dataset = BuildDataset(data_path, dataset, get_transform(train=False),
→train=False)

    #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]['labels'][i].item()], fill=(52, 55, 235,128))
    display(img)

```

### 3 Porównanie Mask R-CNN z Custom Vision

#### 3.1 Dataset Fruits

Po wytrenowaniu Custom Vision otrzymaliśmy takie wyniki dla datasetu Fruits:

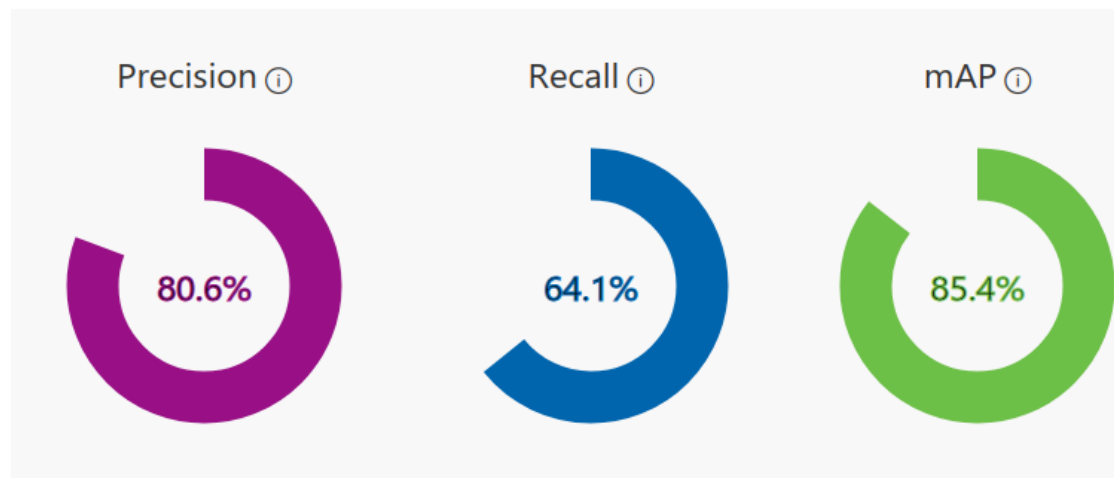
- **Precision** - jak prawdopodobne jest, że jest to prawda?
- **Recall** - spośród tagów, które należy prawidłowo przewidzieć, jaki procent poprawnie znalazł twój model?

- **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: **fdbab1ff-90fc-41ab-a150-f33e7fa85259**

Published as: **detectModel**



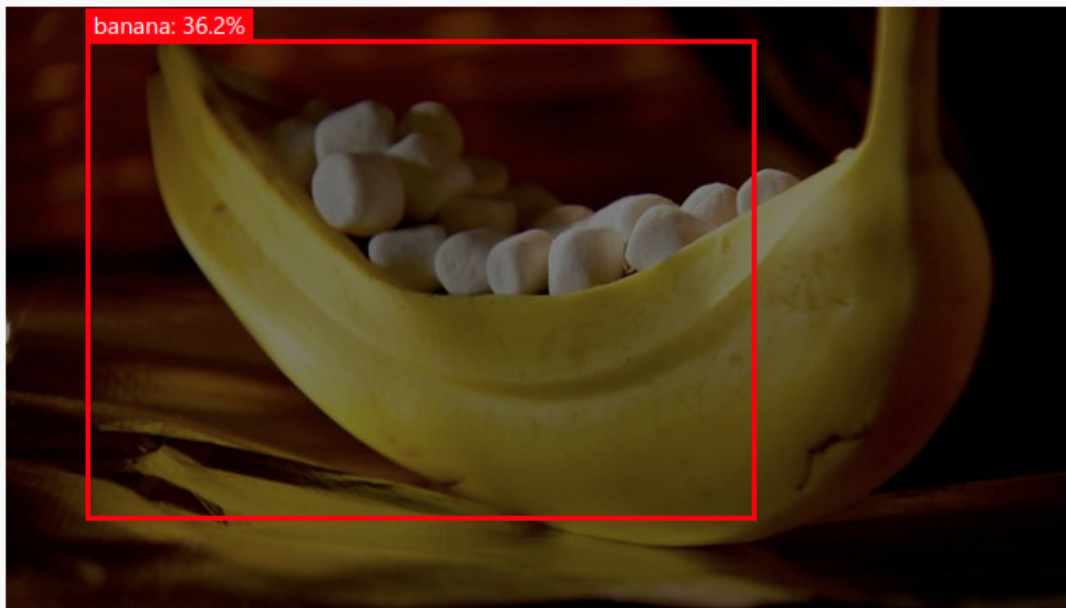
### Performance Per Tag

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

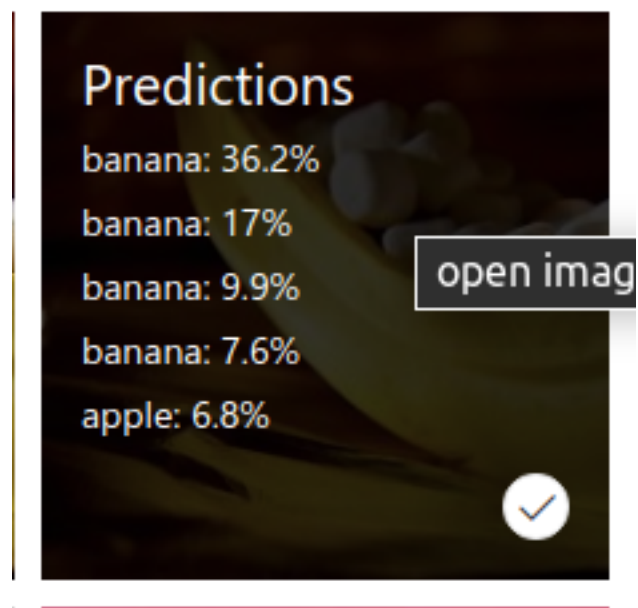
Przykład wykrywania obiektu na zdjęciu. Jest pokazane, że to jest banan.

### 3.1.1 Custom Vision

Zdjęcie dla Custom Vision:



Prediction:



### 3.1.2 Mask R-CNN

```
[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')
```



### 3.1.3 Wnioski z przykładu

W przykładzie zdjęcia, na którym wyeksponowanym obiektem był banan, CustomVision zwrócił nam prawdopodobieństwo 36.2%, co jest wartością nieporównywalnie mniejszą do tej otrzymanej z Mask R-CNN gdzie pewność klasyfikacji wyniosła aż 88.69%. W tym wypadku Mask R-CNN wykazał się lepiej.

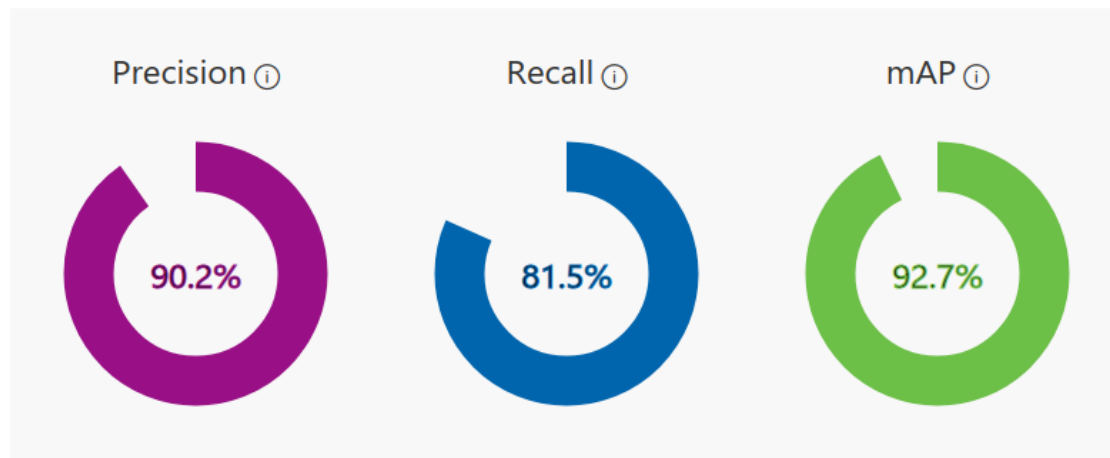
### 3.2 Dataset Monkey Cat Dog

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

Finished training on **12/21/2020, 1:22:06 PM** using **General** domain

Iteration id: **0a93b42e-3f8d-4cd1-b4b3-504c9efbdf7a**

Published as: **detectModel**



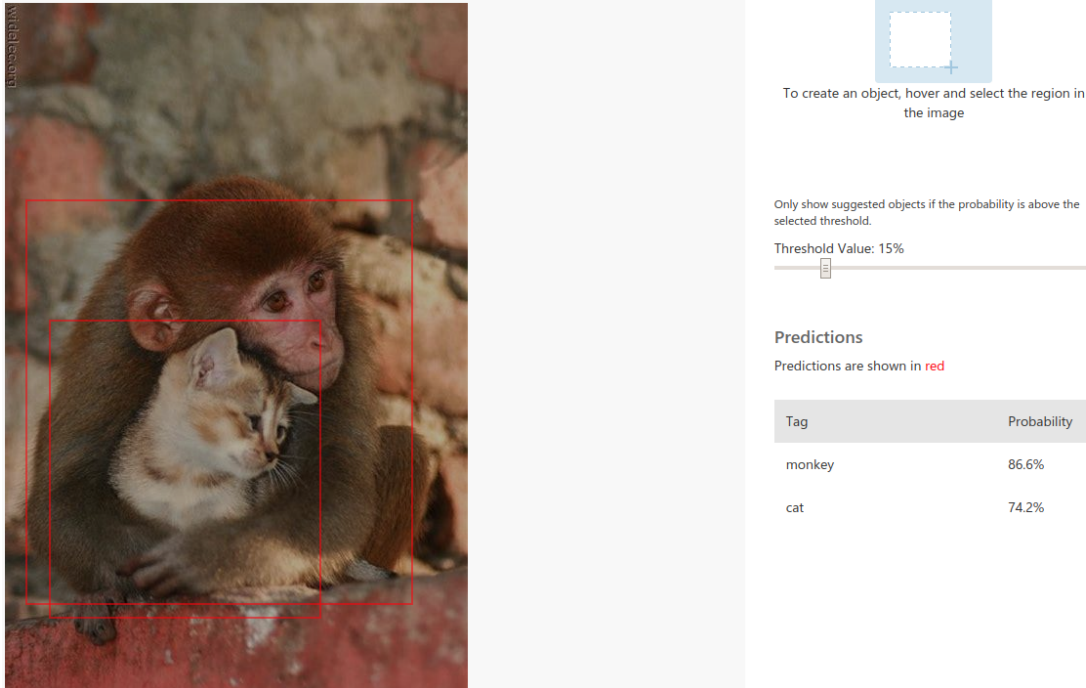
#### Performance Per Tag

Tag	Precision <sup>^</sup>	Recall	A.P.	Image count
monkey	92.5%	81.9%	93.8%	172 <div></div>
cat	89.9%	79.2%	91.4%	213 <div></div>
dog	87.8%	83.7%	92.9%	186 <div></div>



### 3.2.1 Custom Vision

Zdjęcie dla Custom Vision:



Tag	Probability
monkey	86.6%
cat	74.2%

Prediction:



### 3.2.2 Mask R-CNN

```
[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')
```



### 3.2.3 Wnioski z przykładu

W przykładzie zdjęcia, na którym wyeksponowanym obiektami ze zbioru MonkeyCats CustomVision zwrócił nam prawdopodobieństwo wystąpienia obiektu należącego do klasy kot równe 74.2% oraz małpy 86.6%. Natomiast architektura Mask R-CNN nie wykrała na obrazie kota, co było zaskakujące, jednak małpa została wykryta została z prawdopodobieństwem wystąpienia 90%.

W tym wypadku Custom Vision okazał się lepszy niż sieć w architekturze Mask R-CNN. Jednak z wyników cząstkowych działania algorytmu wiem, że Mask R-CNN zwrócił prawdopodobieństwo wystąpienia kota na obrazie o wartości 48% co było nie wystarczające do przedzielenia tego obiektu do klasy kot.

## 3.3 Dataset Play Cards

Dla zbioru danych Play Cards nie udało wytrenować modelu za pomocą platformy Azur Custom Vision ze względu na nie wystarczającą ilość instancji poszczególnych klas w zbiorze uczącym. Jednak dla sieci Mask R-CNN ubogie zasoby zbioru treningowego nie stanowiły problemu, skutecznie udało się wytrenować model oraz go przetestować.

### 3.3.1 Mask R-CNN

```
[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')
```



### 3.3.2 Wnioski z przykładu

Powyżej znajduje się obraz ze zbioru testowego na którym łatwo można zauważyć, że jednak mały zbiór danych miał negatywny wpływ na wyniki. Pomimo poprawnej klasyfikacji obu obiektów do klasy ace wymiar przypisanych ramek utożsamiany z pozycjonowaniem obiektów okazał się niepoprawnym. Niemniej, pomimo nie obejmowania całości reprezentantów na zdjęciu, wykryty obszar jest wystarczający dla autorów.



```

[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

[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,
      )

[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)

[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=False)
      # indices = torch.randperm(len(dataset)).tolist()
      # dataset = torch.utils.data.Subset(dataset, indices[-50:])

[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)

```

```
[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')}
```

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

#labels_dict = {1: 'ace', 2: 'king', 3: 'queen', 4: 'jack', 5: 'ten', 6: 'nine'}

#labels_dict = {1: "cat", 2: "dog", 3: "monkey"}
```

### 3.3.3 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%

```
[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:
```

```

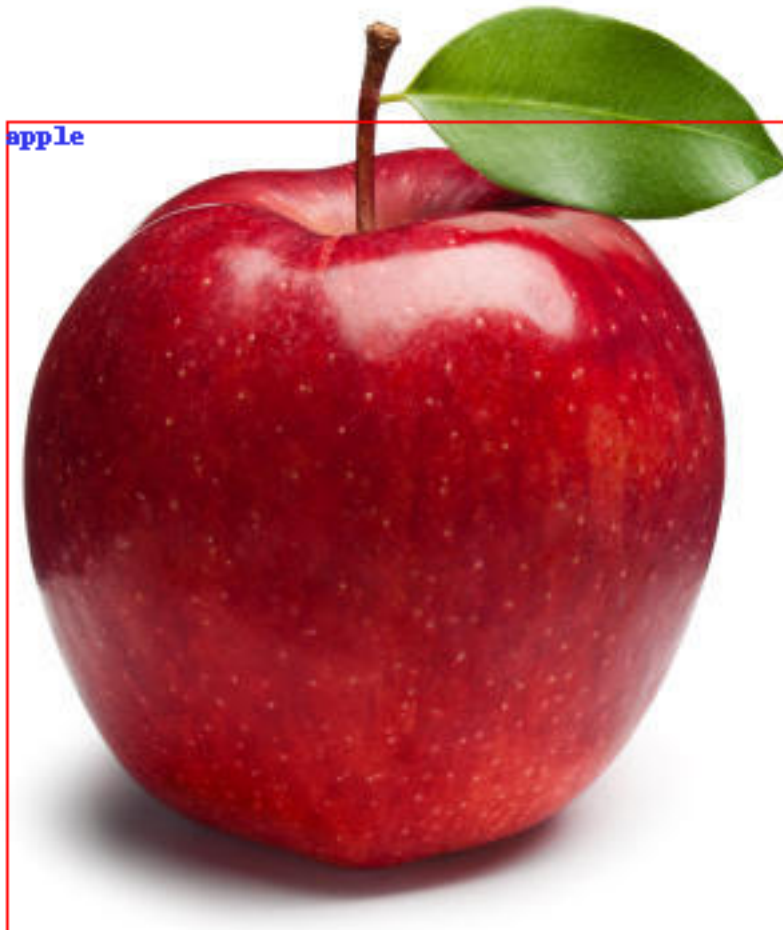
        draw.rectangle(
            ((preds[i][0], preds[i][1]), (preds[i][2], preds[i][3])),
            →outline="red"
        )
        draw.text((preds[i][0], preds[i][1]),
            →labels_dict[prediction[0]['labels'][i].item()], fill=(52, 55, 235, 128))
display(img)

```

```

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

```



### 3.3.4 Wnioski z przykładu

Dla zbioru danych Fruit obydwa modele poprawnie zakwalifikowały obiekt do klasy apple na zdjęciu testowym. Jednak zarówno teraz jak i wcześniej we wszystkich testach przeprowadzonych Custom Vision zwracał niższe prawdopodobieństwa niż Mask R-CNN. Pomimo wysokiej jakości obrazu testowego i niczym nie przesłoniętego obiektu Custom Vision zwraca prawdopodobieństwo klasy apple 62.4% a przypadku modelu Mask R-CNN prawdopodobieństwo wynosi prawie 90%.

## 4 Ogólne wnioski & porównanie

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

Stworzyliśmy wytrenowane modele dla trzech różnych zbiorów danych wykorzystując Mask R-CNN (Own model) oraz Azure Custom Vision.

### 4.1 Maska R-CNN do wykrywania obiektów

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

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

Model Mask 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

### 4.2 Azure Custom Vision

- Custom Vision obsługuje następujące domeny do wykrywania obiektów: General, Logo, Products i Compact.
- Szybka implementacja, dość 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 wytrenować model z wykorzystaniem znaczącej ilości plików treningowych napisanie skryptu wczytującego jest konieczne.

### 4.3 Porównanie

Custom vision w ogóle nie był trenowany na datasecie Play Cards, bo był on niewystarczająco duży.



### Średni procent wykrywania obiektu na zdjęciu

Dataset	MaskaR-CNN	Azure Custom Vision
Fruit	80%	85.4%
MonkeyCats	75%	92.7%
PlayCards	54%	-

### Czas trwania

Dataset	MaskaR-CNN	Azure Custom Vision
Fruit	1h 54m 18.30s	9m 52.18s
MonkeyCats	3h 47m 4.478s	12m 28.89s
PlayCards	2h 23m 4.841s	-

### Plusy Mask R-CNN

- Da się wytrenować model dla różnych zbiorów danych, tylko kwestia czasu.
- Możliwość przeprowadzenia procesu treningowego nie zależy tak silnie od ilości danych treningowych jak ma to miejsce dla Custom Vision.
- Model jest dostępny publicznie.
- Możliwość ustawienia progu decyzyjności.

### Minusy Mask R-CNN

- Złożona architektura sieci neuronowej.
- Długi czas treningu sieci, co przekłada się na koszty.
- Wymaga posiadania wystarczającej mocy obliczeniowej.

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

### Plusy Azure Custom Vision:

- Nie jest konieczne posiadanie dużej wiedzy z dziedziny widzenia komputerowego.
- Algorytmy będą ciągle optymalizowane i udoskonalane.
- Wysoka dokładność
- Krótki czas trenowania modelu.

### Minusy Azure Custom Vision:

- Wymagana subskrypcja Microsoft Azure.
- Niższa personalizacja w porównaniu do Mas R-CNN.
- Ograniczenia związane z algorytmami dostarczonymi przez Microsoft.
- Nie ma możliwości zmiany parametrów dostępnych modeli.
- Wielkość zbioru uczącego jest krytyczna.

#### 4.4 Podsumowanie

W przypadku gdy mamy jasno zdefiniowany złożony problem który będzie wymagał wysokiego stopnia dostrojenia sieci do wymagań, a także posiadamy wystarczającą specjalistyczną wiedzę dziedzinową z zakresu widzenia komputerowego, lepszym wyborem jest skorzystanie z własnej implementacji sieci w architekturze Mask R-CNN.

Natomiast gdy nie posiadamy zaawansowanej wiedzy technicznej, a interesujący nas problem jest stosunkowo prostym zadaniem z dziedziny widzenia komputerowego łatwiej będzie skorzystać z usługi Custom Vision na portalu Microsoft Azure.

Wybór zatem nie jest oczywisty i zależy od poszczególnego przypadku.