HO CHI MINH UNIVERSITY OF SCIENCE

FACULTY OF INFORMATION TECHNOLOGY

Final Semester Project

YOLO PROJECT

Student in charge:

1. Huỳnh Cao Nhật Hiếu – 19127399
2. Ngô Đăng Khoa – 19127444

Contents

1. Requirement 1: Install YOLO

- Step 1: We install Yolo from darknet on Google colab

Install Yolo

Clone Darknet

```
# clone darknet repo
!git clone https://github.com/AlexeyAB/darknet
```

```
Cloning into 'darknet'...
remote: Enumerating objects: 15386, done.
remote: Total 15386 (delta 0), reused 0 (delta 0), pack-reused 15386
Receiving objects: 100% (15386/15386), 14.01 MiB | 12.50 MiB/s, done.
Resolving deltas: 100% (10346/10346), done.
```

Compile

```
# change makefile to have GPU and OPENCV enabled
%cd darknet
!sed -i 's/OPENCV=0/OPENCV=1/' Makefile
!sed -i 's/GPU=0/GPU=1/' Makefile
!sed -i 's/CUDNN=0/CUDNN=1/' Makefile
!sed -i 's/CUDNN_HALF=0/CUDNN_HALF=1/' Makefile
!sed -i 's/LIBSO=0/LIBSO=1/' Makefile
```

```
/content/darknet
```

```
# verify CUDA
!/usr/local/cuda/bin/nvcc --version
```

```
nvcc: NVIDIA (R) Cuda compiler driver
Copyright (c) 2005-2020 NVIDIA Corporation
Built on Mon_Oct_12_20:09:46_PDT_2020
Cuda compilation tools, release 11.1, V11.1.105
Build cuda_11.1.TC455_06.29190527_0
```

```
# make darknet (builds darknet so that you can then use the darknet executable file to run or train object detectors)
!make
```

```
In file included from /usr/local/cuda/include/cuda_runtime.h:96:0,
                 from include/darknet.h:41,
                 from ./src/gaussian_yolo_layer.h:5,
                 from ./src/gaussian_yolo_layer.c:7:
/usr/local/cuda/include/cuda_runtime_api.h:4811:39: note: expected 'void **' but argument is of type 'float **'
 extern __host__ cudaError_t CUDARTAPI cudaHostAlloc(void **pHost, size_t size, unsigned int flags);
                                       ^~~~~~~~~~~~~
gcc -Iinclude/ -I3rdparty/stb/include -DOPENCV `pkg-config --cflags opencv4 2> /dev/null || pkg-config --cflags opencv` -DGPU -I/usr/local/cuda/in
gcc -Iinclude/ -I3rdparty/stb/include -DOPENCV `pkg-config --cflags opencv4 2> /dev/null || pkg-config --cflags opencv` -DGPU -I/usr/local/cuda/in
gcc -Iinclude/ -I3rdparty/stb/include -DOPENCV `pkg-config --cflags opencv4 2> /dev/null || pkg-config --cflags opencv` -DGPU -I/usr/local/cuda/in
gcc -Iinclude/ -I3rdparty/stb/include -DOPENCV `pkg-config --cflags opencv4 2> /dev/null || pkg-config --cflags opencv` -DGPU -I/usr/local/cuda/in
gcc -Iinclude/ -I3rdparty/stb/include -DOPENCV `pkg-config --cflags opencv4 2> /dev/null || pkg-config --cflags opencv` -DGPU -I/usr/local/cuda/in
nvcc -gencode arch=compute_35,code=sm_35 -gencode arch=compute_50,code=[sm_50,compute_50] -gencode arch=compute_52,code=[sm_52,compute_52] -gencod
nvcc warning : The 'compute_35', 'compute_37', 'compute_50', 'sm_35', 'sm_37' and 'sm_50' architectures are deprecated, and may be removed in a fu
nvcc -gencode arch=compute_35,code=sm_35 -gencode arch=compute_50,code=[sm_50,compute_50] -gencode arch=compute_52,code=[sm_52,compute_52] -gencod
nvcc warning : The 'compute_35', 'compute_37', 'compute_50', 'sm_35', 'sm_37' and 'sm_50' architectures are deprecated, and may be removed in a fu
./src/activation_kernels.cu(263): warning: variable "MISH_THRESHOLD" was declared but never referenced
```

- Step 2: Import library and implement predict function

```

# import dependencies
from IPython.display import display, Javascript, Image
from google.colab.output import eval_js
from google.colab.patches import cv2_imshow
from base64 import b64decode, b64encode
import cv2
import numpy as np
import PIL
import io
import html
import time
import matplotlib.pyplot as plt
%matplotlib inline
from darknet import *
```

Then we implemented **darknet_helper** to run detection on image, this function we refered from
https://github.com/MINED30/Face_Mask_Detection_YOLO?fbclid=IwAR1uVLNqP3l9_6Wi3El_F9wAoQeHDM-ENJ8Iw_99_nUSxv-zq23pKu-i4Kk

```python
# darknet helper function to run detection on image
def darknet_helper(img, width, height):
  darknet_image = make_image(width, height, 3)
  img_rgb = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
  img_resized = cv2.resize(img_rgb, (width, height),
                              interpolation=cv2.INTER_LINEAR)

  # get image ratios to convert bounding boxes to proper size
  img_height, img_width, _ = img.shape
  width_ratio = img_width/width
  height_ratio = img_height/height

  # run model on darknet style image to get detections
  copy_image_from_bytes(darknet_image, img_resized.tobytes())
  detections = detect_image(network, class_names, darknet_image)
  free_image(darknet_image)
  return detections, width_ratio, height_ratio
```

Next, we implemented **yolo_predict** draw a bounding box and classify object. This function's input is a path of image, then we use OpenCV to read that image to a 3D array then draw a bounding box on it and save back to its file path.

```python
def yolo_predict(input_file):
  img = cv2.imread(input_file)
  detections, width_ratio, height_ratio = darknet_helper(img, width, height)
  for label, confidence, bbox in detections:
      left, top, right, bottom = bbox2points(bbox)
      left, top, right, bottom = int(left * width_ratio), int(top * height_ratio), int(right * width_ratio), int(bottom * height_ratio)
      cv2.rectangle(img, (left, top), (right, bottom), class_colors[label], 2)
      cv2.putText(img, "{} [{:.2f}]".format(label, float(confidence)),
                        (left, top - 5), cv2.FONT_HERSHEY_SIMPLEX, 0.5,
                        class_colors[label], 2)
  cv2.imwrite(input_file, img)
```

- Step 3: Implement web application to demo Yolo
  To make a web application on Colab we used **flask-ngrok** to make a local server.

```
!pip install flask-ngrok
```

```
Collecting flask-ngrok
  Downloading flask_ngrok-0.0.25-py3-none-any.whl (3.1 kB)
Requirement already satisfied: requests in /usr/local/lib/python3.7/dist-packages (from flask-ngrok) (2.23.0)
Requirement already satisfied: Flask>=0.8 in /usr/local/lib/python3.7/dist-packages (from flask-ngrok) (1.1.4)
Requirement already satisfied: click<8.0,>=5.1 in /usr/local/lib/python3.7/dist-packages (from Flask>=0.8->flask-ngrok) (7.1.2)
Requirement already satisfied: Jinja2<3.0,>=2.10.1 in /usr/local/lib/python3.7/dist-packages (from Flask>=0.8->flask-ngrok) (2.11.3)
Requirement already satisfied: itsdangerous<2.0,>=0.24 in /usr/local/lib/python3.7/dist-packages (from Flask>=0.8->flask-ngrok) (1.1.0)
Requirement already satisfied: Werkzeug<2.0,>=0.15 in /usr/local/lib/python3.7/dist-packages (from Flask>=0.8->flask-ngrok) (1.0.1)
Requirement already satisfied: MarkupSafe>=0.23 in /usr/local/lib/python3.7/dist-packages (from Jinja2<3.0,>=2.10.1->Flask>=0.8->flask-ngrok) (2.0.1
Requirement already satisfied: chardet<4,>=3.0.2 in /usr/local/lib/python3.7/dist-packages (from requests->flask-ngrok) (3.0.4)
Requirement already satisfied: idna<3,>=2.5 in /usr/local/lib/python3.7/dist-packages (from requests->flask-ngrok) (2.10)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.7/dist-packages (from requests->flask-ngrok) (2021.10.8)
Requirement already satisfied: urllib3!=1.25.0,!=1.25.1,<1.26,>=1.21.1 in /usr/local/lib/python3.7/dist-packages (from requests->flask-ngrok) (1.24.
Installing collected packages: flask-ngrok
Successfully installed flask-ngrok-0.0.25
```

```
!pip install pyngrok
!ngrok authtoken 23594Kn4vbYxJ0zcGK5G0J3Gqux_XuEKpCypMYoW1nA8hfG9
```

```
Collecting pyngrok
  Downloading pyngrok-5.1.0.tar.gz (745 kB)
     |████████████████████████████████| 745 kB 5.4 MB/s
Requirement already satisfied: PyYAML in /usr/local/lib/python3.7/dist-packages (from pyngrok) (3.13)
Building wheels for collected packages: pyngrok
  Building wheel for pyngrok (setup.py) ... done
  Created wheel for pyngrok: filename=pyngrok-5.1.0-py3-none-any.whl size=19006 sha256=447c374cbe4ebb8740e71b537c697e393fd297031d02efaaed54e250600ff
  Stored in directory: /root/.cache/pip/wheels/bf/e6/af/ccf6598ecefecd44104069371795cb9b3afbcd16987f6ccfb3
Successfully built pyngrok
Installing collected packages: pyngrok
Successfully installed pyngrok-5.1.0
Authtoken saved to configuration file: /root/.ngrok2/ngrok.yml
```

To run this server, we have to load the model first.

```python
network, class_names, class_colors = load_network("/content/drive/MyDrive/Yolo/yolo_testing.cfg",
                                                  "/content/drive/MyDrive/Yolo/obj.data", "/content/drive/MyDrive/Yolo/yolov3_training_last.weights")
width = network_width(network)
height = network_height(network)
```

Backend Flask server

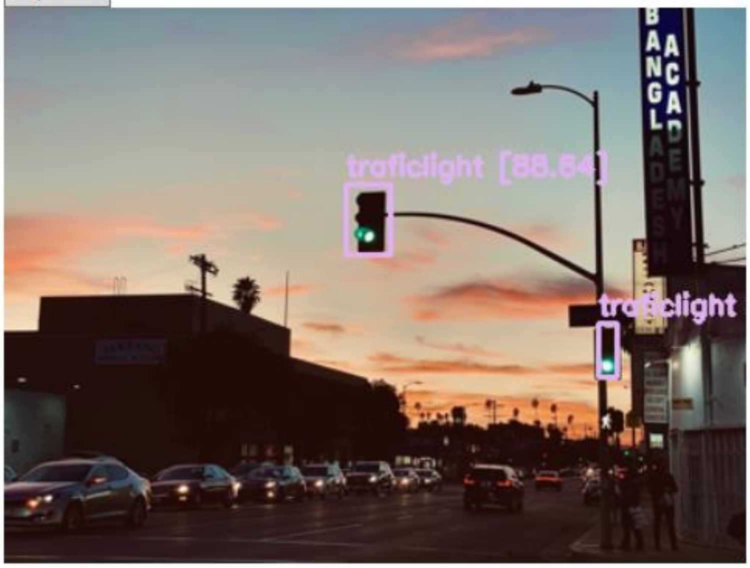
```python
import os
from flask import Flask, request, render_template
from flask_ngrok import run_with_ngrok
app = Flask(__name__,template_folder='/content/drive/MyDrive/Yolo')
run_with_ngrok(app)
app.config["IMAGE_UPLOADS"] = "/content/drive/MyDrive/Yolo"


@app.route("/")
def home():
    return render_template('index.html')
# Route to upload image

@app.route('/upload-image', methods=['GET', 'POST'])
def upload_image():
    if request.method == "POST":
        if request.files:
            image = request.files["image"]
            save_path = os.path.join(app.config["IMAGE_UPLOADS"], image.filename)
            image.save(save_path)
            yolo_predict(save_path)
            return render_template("index.html", uploaded_image=image.filename)
    return render_template("index.html")

@app.route('/uploads/<filename>')
def send_uploaded_file(filename=''):
    from flask import send_from_directory
    return send_from_directory(app.config["IMAGE_UPLOADS"], filename)
app.run()
```

Frontend html file:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Upload Face with ID</title>
</head>
<body>

    <div class="container">
      <div class="row">
        <div class="col">
          <hr>

    <form action="/upload-image" method="POST" enctype="multipart/form-data">

    <div class="form-group">
              <label>Select image</label>
              <div class="custom-file">
                <input type="file" class="custom-file-input" name="image"

    id="image">
                <label class="custom-file-label" for="image">Select image...</label>
              </div>
            </div>
    <button type="submit" class="btn btn-primary">Upload</button>
    </form>

        </div>
      </div>
    </div>

    <img src="{{ url_for('send_uploaded_file', filename=uploaded_image) }}" />

</body>
</html>
```

Idea: After upload an image to server, the image will be saved to a folder on my Google Drive then call function **yolo_predict** to draw bounding box and save it again. After that the backend server will return predicted image back to front end to show the result.

Select image
Choose File No file chosen Select image...
Upload



Select image
Choose File No file chosen Select image...
Upload

2. Requirement 2: Apply YOLO to build application

2.1 EXPLAIN STEP BY STEP:

- Step 1: We prepare the dataset. My group choose the dataset “Road Sign Detection” on the Kaggle.
  The link of dataset: https://www.kaggle.com/andrewmvd/road-sign-detection
  In this dataset, there are 4 classes: Trafic Light, Stop, Speedlimit and Crosswalk. Therefore, we need to add a class named Paking.
- Step 2: We annotate the dataset. We use LabelImg to annotate the dataset.
- Step 3: We prepare a folder yolo in the Google drive to contain the data of the model that we will train.
  Note: From step 3, we refered from [YOLOv3 Custom Object Detection with Transfer Learning | by Nitin Tiwari | Medium]()

```
[2] from google.colab import drive
    drive.mount('/content/gdrive')
    !ln -s /content/gdrive/My\ Drive/ /mydrive
    !ls /mydrive

Mounted at /content/gdrive
'Review UI.gdoc'               'Untitled Diagram (3).drawio'           yolo
'Untitled Diagram (1).drawio'  'Untitled Diagram.drawio'
'Untitled Diagram (2).drawio'  'Weekly Report Template Group 10.docx'
```

```
%cd /mydrive/yolo

/content/gdrive/My Drive/yolo
```

```
[4] !ls

classes.txt  yolo_testing.cfg                   yolov3_training_4000.weights
darknet      YOLOv3_Custom_Object_Detection.ipynb  yolov3_training_best.weights
images.zip   yolov3_training_1000.weights       yolov3_training_last.weights
obj.data     yolov3_training_2000.weights
obj.names    yolov3_training_3000.weights
```

- Step 3: We clone, configure, and compile the Darknet

1) Clone, configure & compile Darknet

```
[ ] # Clone
    !git clone https://github.com/AlexeyAB/darknet

Cloning into 'darknet'...
remote: Enumerating objects: 15386, done.
remote: Total 15386 (delta 0), reused 0 (delta 0), pack-reused 15386
Receiving objects: 100% (15386/15386), 14.01 MiB | 5.81 MiB/s, done.
Resolving deltas: 100% (10345/10345), done.
Checking out files: 100% (2050/2050), done.
```

```
# Configure
%cd darknet
!sed -i 's/OPENCV=0/OPENCV=1/' Makefile
!sed -i 's/GPU=0/GPU=1/' Makefile
!sed -i 's/CUDNN=0/CUDNN=1/' Makefile

/content/gdrive/My Drive/yolo/darknet
```

```
# Compile
!make
```

- Step 4: We fine-tune the model by configure yolov3.cfg file:

2) Configure yolov3.cfg file

```
# Make a copy of yolov3.cfg
!cp cfg/yolov3.cfg cfg/yolov3_training.cfg
```

```
cp: cannot stat 'cfg/yolov3.cfg': No such file or directory
```

```
# Change lines in yolov3.cfg file
!sed -i 's/batch=1/batch=64/' cfg/yolov3_training.cfg
!sed -i 's/subdivisions=1/subdivisions=16/' cfg/yolov3_training.cfg
!sed -i 's/max_batches = 500200/max_batches =  8000/' cfg/yolov3_training.cfg
!sed -i '610 s@classes=80@classes=4@' cfg/yolov3_training.cfg
!sed -i '696 s@classes=80@classes=4@' cfg/yolov3_training.cfg
!sed -i '783 s@classes=80@classes=4@' cfg/yolov3_training.cfg
!sed -i '603 s@filters=255@filters=27@' cfg/yolov3_training.cfg
!sed -i '689 s@filters=255@filters=27@' cfg/yolov3_training.cfg
!sed -i '776 s@filters=255@filters=27@' cfg/yolov3_training.cfg
```

```
%cd '/content/gdrive/My Drive/yolo/darknet'
```

```
/content/gdrive/My Drive/yolo/darknet
```

  - Firstly, we create a copy of yolov3.cfg file and names it as yolov3_training.cfg so that all the changes and configurations that we will make for our custom model would be reflected in the copy and not the original file. It’s a good practice to keep a backup of the original .cfg file if in case anything goes wrong.
  - Secondly, we edit the yolov3_training.cfg:

As darknet recommend, if the custom object detection model includes ‘n’ no. of classes, then max_batches = 2000 * n and filters = (n + 5) * 3

My model has 5 classes so max_batches =10000, classes=5, filters=30.

- Step 5: we Create.names and .data files.

3) Create .names and .data files

```
!echo -e 'traficlight\nstop\nspeedlimit\ncrosswalk\nparking' > data/obj.names
!echo -e 'classes= 5\ntrain  = data/train.txt\nvalid  = data/test.txt\nnames = data/obj.names\nbackup = /mydrive/yolo' > data/obj.data
```

This cell creates obj.names and obj.data files inside the darknet/data/obj directory. These files include metadata such as class names and no. of classes required for training.

- Step 6: We save the yolov3_training.cfg and obj.name files in the folder of the Drive.

4) Save yolov3_training.cfg and obj.names files in Google drive

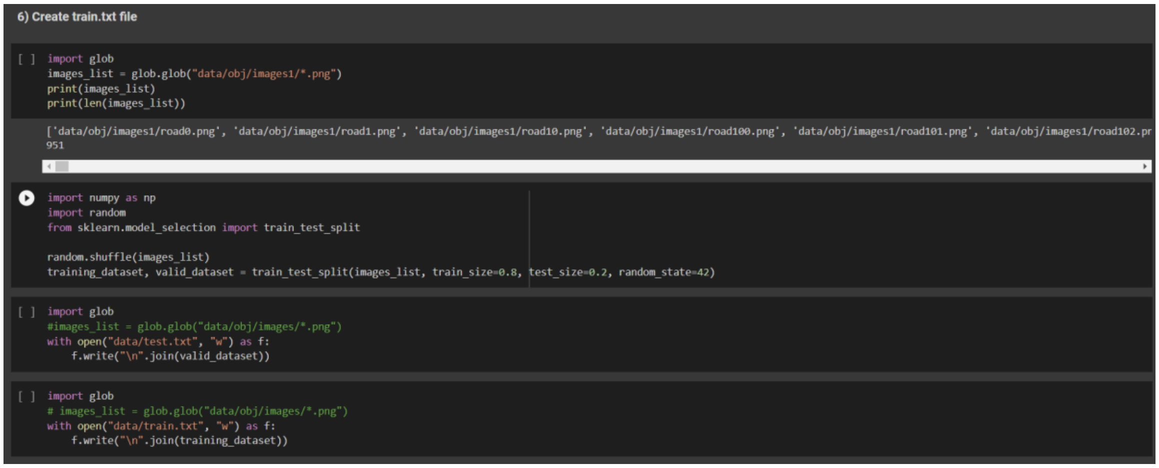
```
!cp cfg/yolov3_training.cfg /mydrive/yolo/yolo_testing.cfg
!cp data/obj.names /mydrive/yolo/classes.txt
```

- Step 7: We unzip the dataset and move it into data/obj of the Darknet

5) Create a folder and unzip image dataset

```
!mkdir data/obj
!unzip /mydrive/yolo/images.zip -d data/obj
```

- Step 8: We create train.txt and test.txt that 2 files to contain the images to train and valid. In this project, we split dataset into 8 training and 2 validation.



6) Create train.txt file

```
import glob
images_list = glob.glob("data/obj/images1/*.png")
print(images_list)
print(len(images_list))
```

```
['data/obj/images1/road0.png', 'data/obj/images1/road1.png', 'data/obj/images1/road10.png', 'data/obj/images1/road100.png', 'data/obj/images1/road101.png', 'data/obj/images1/road102.pr
951
```

```
import numpy as np
import random
from sklearn.model_selection import train_test_split

random.shuffle(images_list)
training_dataset, valid_dataset = train_test_split(images_list, train_size=0.8, test_size=0.2, random_state=42)
```

```
import glob
#images_list = glob.glob("data/obj/images/*.png")
with open("data/test.txt", "w") as f:
    f.write("\n".join(valid_dataset))
```

```
import glob
# images_list = glob.glob("data/obj/images/*.png")
with open("data/train.txt", "w") as f:
    f.write("\n".join(training_dataset))
```

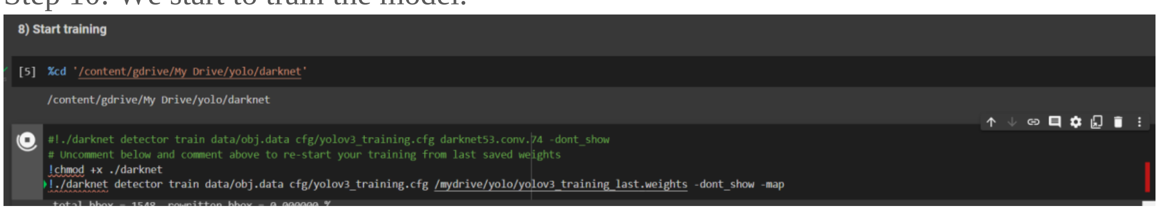
- Step 9: We are applying Transfer Learning by adding our own layers to an already trained model. So, we download the pre-trained weights called darknet53.conv.74. Thus, our custom model will be trained using these pre-trained weights instead of randomly initialized weights which in turn will save a lot of time and computations while training our own model.

7) Download pre-trained weights for the convolutional layers file

```
!wget https://pjreddie.com/media/files/darknet53.conv.74
```

```
--2022-01-03 07:12:46--  https://pjreddie.com/media/files/darknet53.conv.74
Resolving pjreddie.com (pjreddie.com)... 128.208.4.108
Connecting to pjreddie.com (pjreddie.com)|128.208.4.108|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 162482580 (155M) [application/octet-stream]
Saving to: 'darknet53.conv.74'

darknet53.conv.74   100%[===================>] 154.96M  25.0MB/s    in 6.2s

2022-01-03 07:12:53 (25.1 MB/s) - 'darknet53.conv.74' saved [162482580/162482580]
```

- Step 10: We start to train the model.



8) Start training

```
%cd '/content/gdrive/My Drive/yolo/darknet'
```

```
/content/gdrive/My Drive/yolo/darknet
```

```
#!./darknet detector train data/obj.data cfg/yolov3_training.cfg darknet53.conv.74 -dont_show
# Uncomment below and comment above to re-start your training from last saved weights
!chmod +x ./darknet
!./darknet detector train data/obj.data cfg/yolov3_training.cfg /mydrive/yolo/yolov3_training_last.weights -dont_show -map
```

2.2 EVALUATE THE MODEL

- Average Precision

Class_id	*Name*	*TP*	*FP*	*ap*
0	traficlight	16	10	55.02%
1	stop	10	7	81.07%
2	speedlimit	105	4	98.43%

3	crosswalk	34	5	93.67%
4	parking	8	0	70.91%

- F1-score & Average IoU

conf_thresh	precision	recall	F1-score	TP	FP	FN	average IoU
0.25	0.89	0.85	0.87	220	26	40	72.34%

- Mean Average Precision

mean average precision (mAP@0.50) = 79.68%

```
calculation mAP (mean average precision)...
Detection layer: 82 - type = 28
Detection layer: 94 - type = 28
Detection layer: 106 - type = 28
192
detections_count = 945, unique_truth_count = 260
class_id = 0, name = traficlight, ap = 55.02%        (TP = 16, FP = 10)
class_id = 1, name = stop, ap = 81.07%        (TP = 10, FP = 7)
class_id = 2, name = speedlimit, ap = 98.43%        (TP = 152, FP = 4)
class_id = 3, name = crosswalk, ap = 93.67%        (TP = 34, FP = 5)
class_id = 4, name = parking, ap = 70.19%        (TP = 8, FP = 0)

for conf_thresh = 0.25, precision = 0.89, recall = 0.85, F1-score = 0.87
for conf_thresh = 0.25, TP = 220, FP = 26, FN = 40, average IoU = 72.34 %

IoU threshold = 50 %, used Area-Under-Curve for each unique Recall
mean average precision (mAP@0.50) = 0.796775, or 79.68 %
Total Detection Time: 14 Seconds

Set -points flag:
`-points 101` for MS COCO
`-points 11` for PascalVOC 2007 (uncomment `difficult` in voc.data)
`-points 0` (AUC) for ImageNet, PascalVOC 2010-2012, your custom dataset

mean_average_precision (mAP@0.50) = 0.796775
Saving weights to /mydrive/yolo/yolov3_training_last.weights
Resizing, random_coef = 1.40

384 x 384
```

3. References:

[1] https://github.com/MINED30/Face_Mask_Detection_YOLO

[2] YOLOv3 Custom Object Detection with Transfer Learning | by Nitin Tiwari | Medium