Paris-Est Créteil Val-de-Marne University
Faculty of Science and Technology
M2 SDTDS 2020-2021

PROJECT REPORT

Deep Learning

Training CNN model
with darknet framework

TP & Report prepared by :

- **MERAD Fouad Fouzi**
- **BENSAMMAR Mohamed Aimene**
- **BADAD Sanae**
- **TALEB Yanis**

Introduction:

Artificial intelligence (AI) is a process of imitating human intelligence that relies on the creation and application of algorithms executed in a dynamic computing environment. Its purpose is to enable computers to think and act like human beings.

To achieve this, three components are needed:

- Computer systems
- Data with management systems
- Advanced AI algorithms (code)

To get as close as possible to human behavior, artificial intelligence needs a high amount of data and high processing capacity.

The goal of the project is to create a code in Google Colab doped with artificial intelligence (AI) that can detect faces with or without a mask, bikes, scooters...

In this project we will we will answer the following questions:

- What is a Darknet?
- What is Yolo and what the advantages of YOLO?
- What are the different steps of project realization?
- Model training with yolov4
- Test the model
- Submit the result to Kaggle

I- What is a Darknet

Darknet is an open-source neural network framework written in C and CUDA, It is fast, easy to install, and supports CPU and GPU computation. Darknet is mainly for Object Detection, and have different architecture, features than other deep learning frameworks. It is faster than many other NN architectures and approaches

Liens utile Here you can some find project using darknet framework: Darknet: Open-Source Neural Networks in C (pjreddie.com)

II- What is Yolo

Object detection is one of the classical problems in computer vision where you work to recognize *what* and *where* — specifically what objects are inside a given image and also where they are in the image. The problem of object detection is more complex than classification, which also can recognize objects but doesn't indicate where the object is located in the image. In addition, classification doesn't work on images containing more than one object.

YOLO is popular because it achieves high accuracy while also being able to run in real-time. The algorithm "only looks once" at the image in the sense that it requires only one forward propagation pass through the neural network to make predictions. After non-max suppression (which makes sure the object detection algorithm only detects each object once), it then outputs recognized objects together with the bounding boxes.

The advantages of YOLO:

With YOLO, a single CNN simultaneously predicts multiple bounding boxes and class probabilities for those boxes. YOLO trains on full images and directly optimizes detection performance. This model has a number of benefits over other object detection methods:

YOLO is extremely fast

YOLO sees the entire image during training and test time so it implicitly encodes contextual information about classes as well as their appearance.

YOLO learns generalizable representations of objects so that when trained on natural images and tested on artwork, the algorithm outperforms other top detection methods.

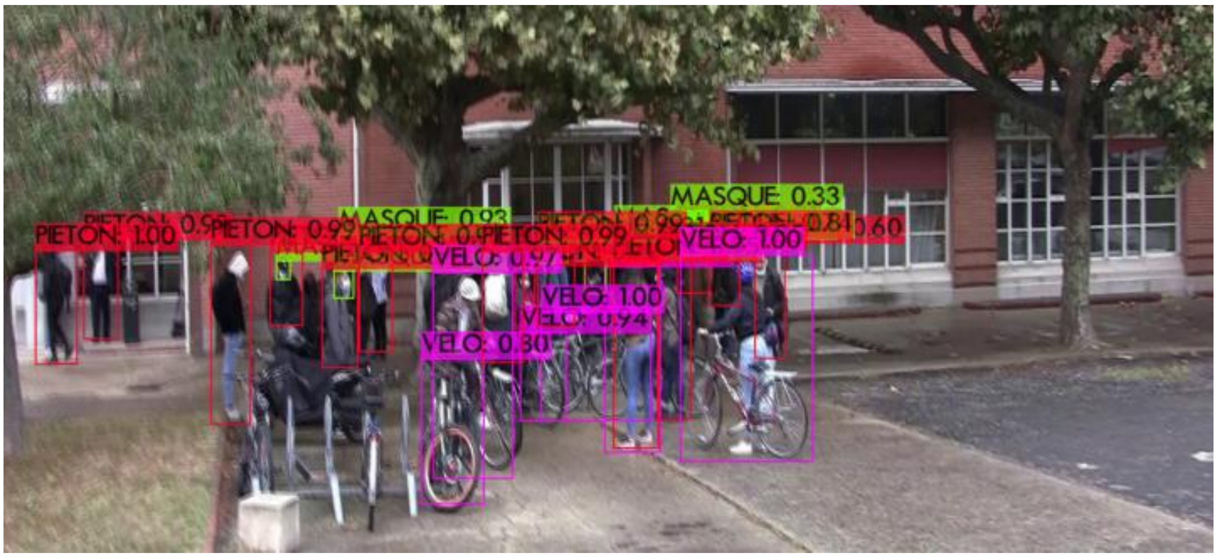
Note:

Darknet architecture & YOLO is a specialized framework, and they are on top of their game in speed and accuracy. YOLO can run on CPU but you get 500 times more speed on GPU as it leverages CUDA and cuDNN.

III- Different steps of project realization

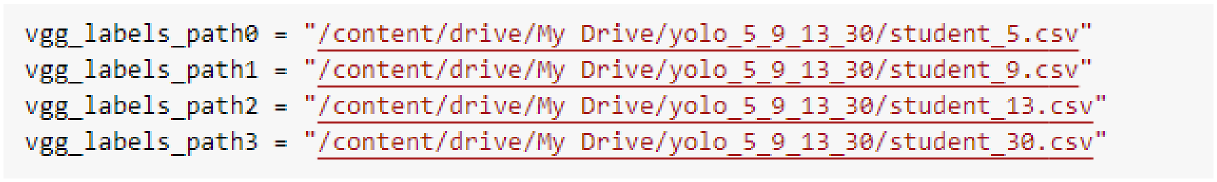
Data Preparation:

After making a video that addresses people with or without a mask, bicycles and scooters, we have moved to the fragmentation of videos into images



<u>**Example of a labeled image**</u>

For our project, we chose to use the labels of 4 students (5, 9, 13, 30):

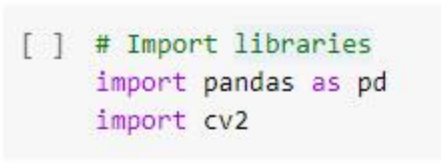


```
vgg_labels_path0 = "/content/drive/My Drive/yolo_5_9_13_30/student_5.csv"
vgg_labels_path1 = "/content/drive/My Drive/yolo_5_9_13_30/student_9.csv"
vgg_labels_path2 = "/content/drive/My Drive/yolo_5_9_13_30/student_13.csv"
vgg_labels_path3 = "/content/drive/My Drive/yolo_5_9_13_30/student_30.csv"
```

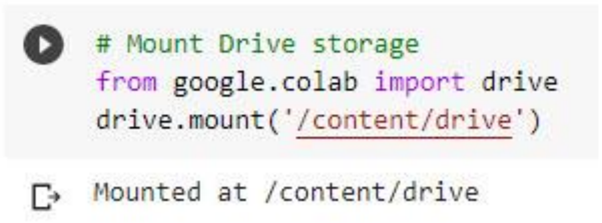
Labels of student (5, 9, 13, 30)

After the fragmentation of the videos into images using OpenCV, we labeled the images obtained, using VGG Annotator, classified into 6 classes (Bike, Scooter, pedestrian, mask, pas_de_masque, mal_mis). Then we got the labeling results in (.csv) format. We worked on 776 images. Then we cleaned up the data obtained, ignoring the bad labeling, and keeping only the well-labeled images. Then we save the coordinates of the labeling rectangles of the images in files (.txt). Finally, we split the data taking 75% of the images for training and the remaining 15% for validation.

Presenting our code (commented) :

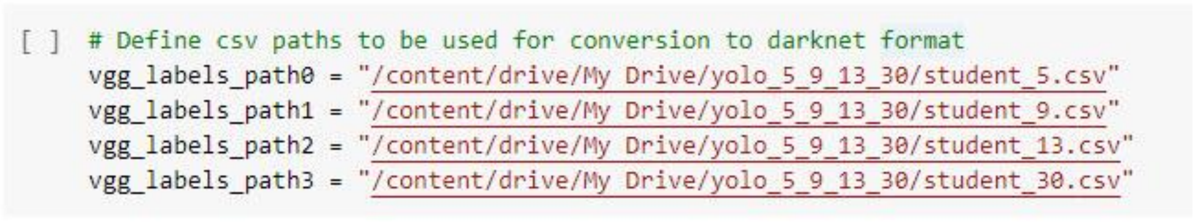


```
# Import libraries
import pandas as pd
import cv2
```

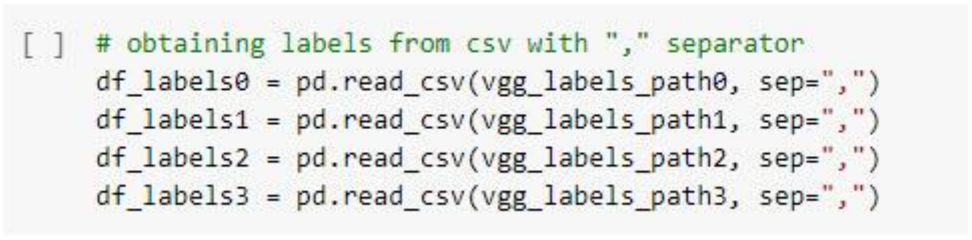


```
# Mount Drive storage
from google.colab import drive
drive.mount('/content/drive')
```

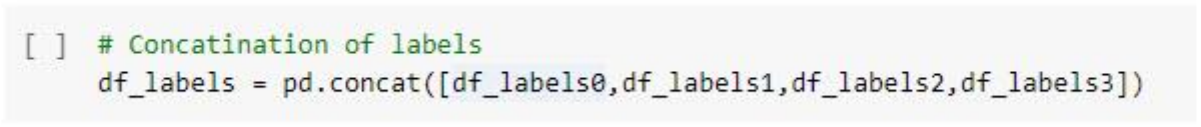
```
Mounted at /content/drive
```



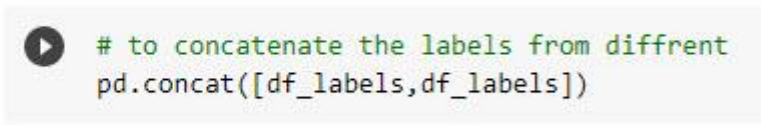
```
# Define csv paths to be used for conversion to darknet format
vgg_labels_path0 = "/content/drive/My Drive/yolo_5_9_13_30/student_5.csv"
vgg_labels_path1 = "/content/drive/My Drive/yolo_5_9_13_30/student_9.csv"
vgg_labels_path2 = "/content/drive/My Drive/yolo_5_9_13_30/student_13.csv"
vgg_labels_path3 = "/content/drive/My Drive/yolo_5_9_13_30/student_30.csv"
```



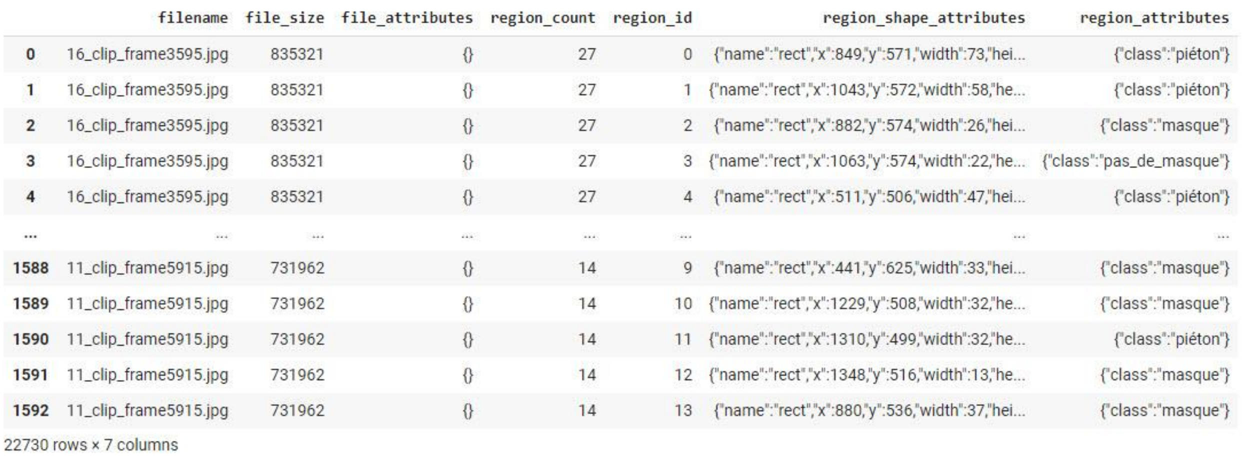
```
# obtaining labels from csv with "," separator
df_labels0 = pd.read_csv(vgg_labels_path0, sep=",")
df_labels1 = pd.read_csv(vgg_labels_path1, sep=",")
df_labels2 = pd.read_csv(vgg_labels_path2, sep=",")
df_labels3 = pd.read_csv(vgg_labels_path3, sep=",")
```



```
# Concatination of labels
df_labels = pd.concat([df_labels0,df_labels1,df_labels2,df_labels3])
```

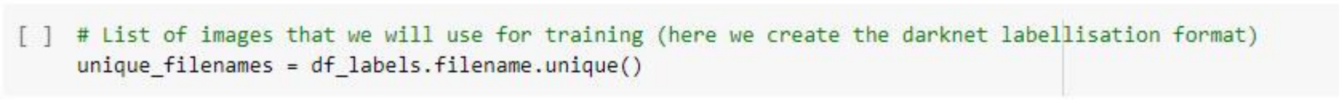


```
# to concatenate the labels from diffrent
pd.concat([df_labels,df_labels])
```

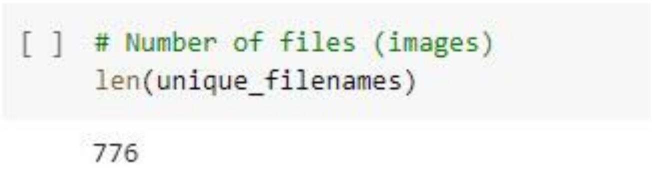


	filename	file_size	file_attributes	region_count	region_id	region_shape_attributes	region_attributes
0	16_clip_frame3595.jpg	835321	{}	27	0	{"name":"rect","x":849,"y":571,"width":73,"hei...	{"class":"piéton"}
1	16_clip_frame3595.jpg	835321	{}	27	1	{"name":"rect","x":1043,"y":572,"width":58,"he...	{"class":"piéton"}
2	16_clip_frame3595.jpg	835321	{}	27	2	{"name":"rect","x":882,"y":574,"width":26,"hei...	{"class":"masque"}
3	16_clip_frame3595.jpg	835321	{}	27	3	{"name":"rect","x":1063,"y":574,"width":22,"he...	{"class":"pas_de_masque"}
4	16_clip_frame3595.jpg	835321	{}	27	4	{"name":"rect","x":511,"y":506,"width":47,"hei...	{"class":"piéton"}
...	...	...	...	...	...	...	...
1588	11_clip_frame5915.jpg	731962	{}	14	9	{"name":"rect","x":441,"y":625,"width":33,"hei...	{"class":"masque"}
1589	11_clip_frame5915.jpg	731962	{}	14	10	{"name":"rect","x":1229,"y":508,"width":32,"he...	{"class":"masque"}
1590	11_clip_frame5915.jpg	731962	{}	14	11	{"name":"rect","x":1310,"y":499,"width":32,"he...	{"class":"piéton"}
1591	11_clip_frame5915.jpg	731962	{}	14	12	{"name":"rect","x":1348,"y":516,"width":13,"he...	{"class":"masque"}
1592	11_clip_frame5915.jpg	731962	{}	14	13	{"name":"rect","x":880,"y":536,"width":37,"hei...	{"class":"masque"}

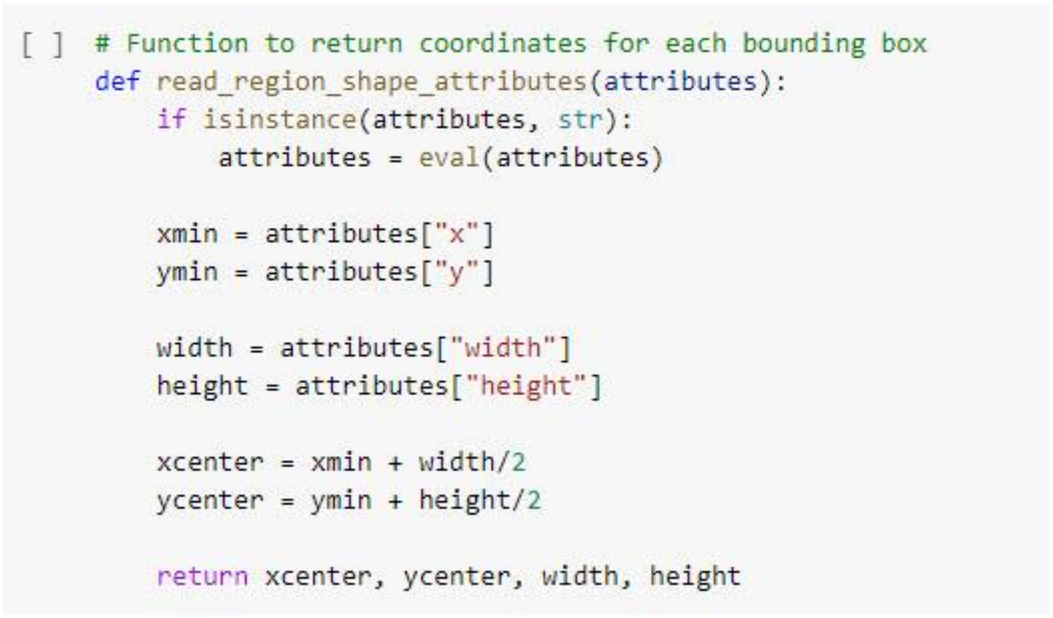
22730 rows × 7 columns



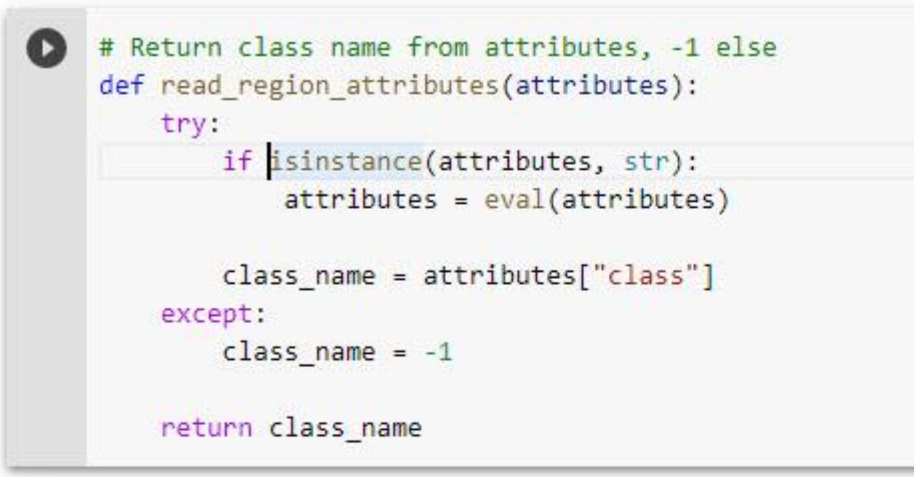
```
[ ] # List of images that we will use for training (here we create the darknet labellisation format)
    unique_filenames = df_labels.filename.unique()
```



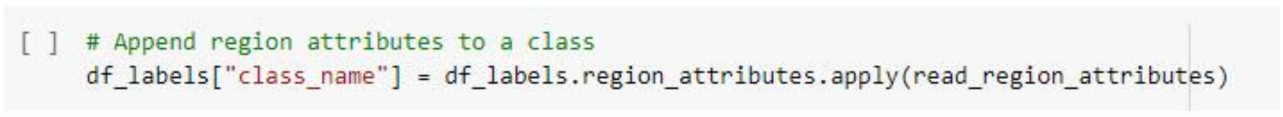
```
[ ] # Number of files (images)
    len(unique_filenames)

    776
```



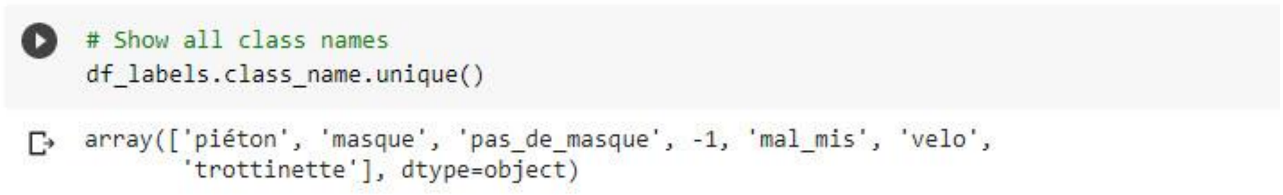
```
[ ] # Function to return coordinates for each bounding box
    def read_region_shape_attributes(attributes):
        if isinstance(attributes, str):
            attributes = eval(attributes)

        xmin = attributes["x"]
        ymin = attributes["y"]

        width = attributes["width"]
        height = attributes["height"]

        xcenter = xmin + width/2
        ycenter = ymin + height/2

        return xcenter, ycenter, width, height
```



```python
# Return class name from attributes, -1 else
def read_region_attributes(attributes):
    try:
        if isinstance(attributes, str):
            attributes = eval(attributes)

        class_name = attributes["class"]
    except:
        class_name = -1

    return class_name
```

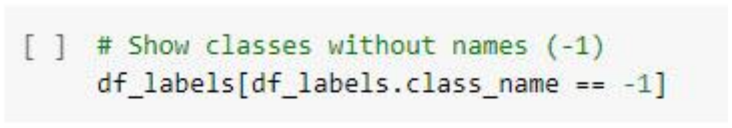


```python
# Append region attributes to a class
df_labels["class_name"] = df_labels.region_attributes.apply(read_region_attributes)
```

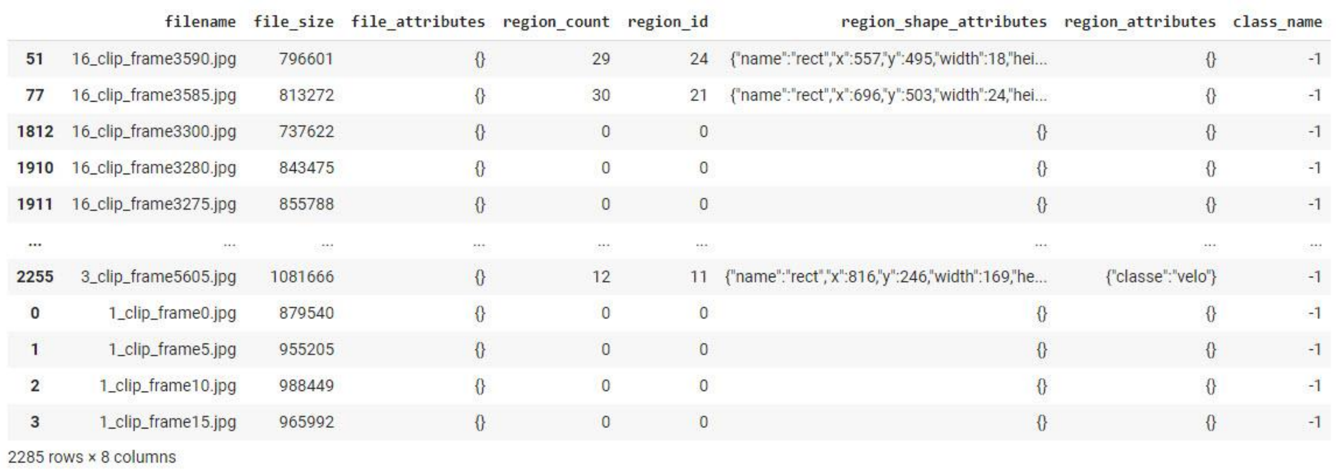


```python
# Show all class names
df_labels.class_name.unique()
```

```
array(['piéton', 'masque', 'pas_de_masque', -1, 'mal_mis', 'velo',
       'trottinette'], dtype=object)
```

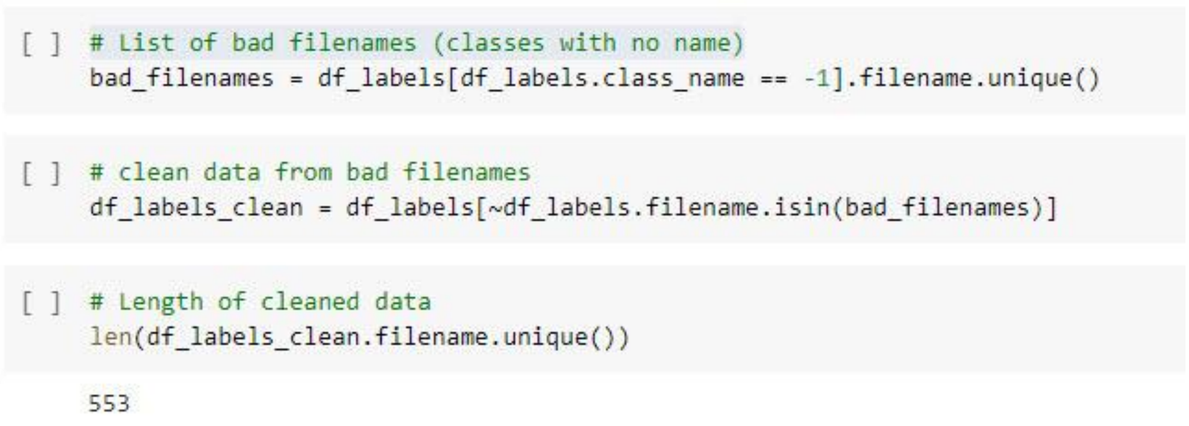


```python
# Show classes without names (-1)
df_labels[df_labels.class_name == -1]
```



	filename	file_size	file_attributes	region_count	region_id	region_shape_attributes	region_attributes	class_name
51	16_clip_frame3590.jpg	796601	{}	29	24	{"name":"rect","x":557,"y":495,"width":18,"hei...	{}	-1
77	16_clip_frame3585.jpg	813272	{}	30	21	{"name":"rect","x":696,"y":503,"width":24,"hei...	{}	-1
1812	16_clip_frame3300.jpg	737622	{}	0	0	{}	{}	-1
1910	16_clip_frame3280.jpg	843475	{}	0	0	{}	{}	-1
1911	16_clip_frame3275.jpg	855788	{}	0	0	{}	{}	-1
...	...	...	...	...	...	...	...	...
2255	3_clip_frame5605.jpg	1081666	{}	12	11	{"name":"rect","x":816,"y":246,"width":169,"he...	{"classe":"velo"}	-1
0	1_clip_frame0.jpg	879540	{}	0	0	{}	{}	-1
1	1_clip_frame5.jpg	955205	{}	0	0	{}	{}	-1
2	1_clip_frame10.jpg	988449	{}	0	0	{}	{}	-1
3	1_clip_frame15.jpg	965992	{}	0	0	{}	{}	-1

2285 rows × 8 columns

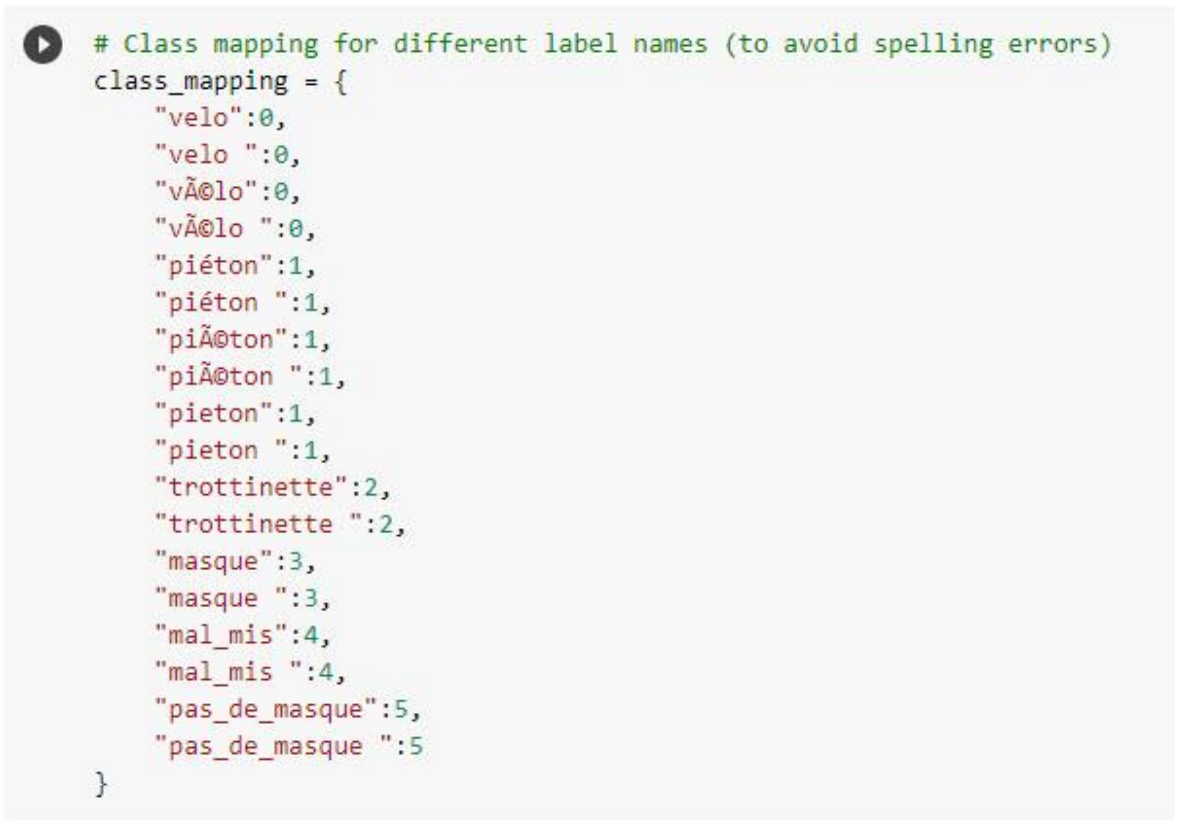


```python
# List of bad filenames (classes with no name)
bad_filenames = df_labels[df_labels.class_name == -1].filename.unique()
```

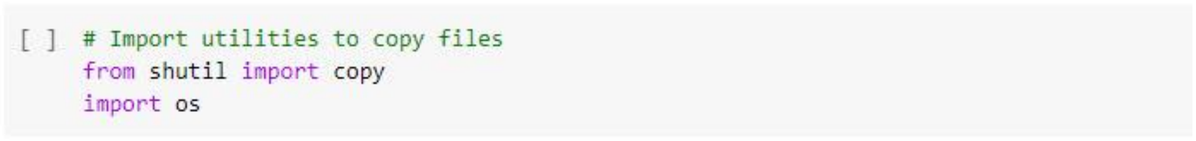
```python
# clean data from bad filenames
df_labels_clean = df_labels[~df_labels.filename.isin(bad_filenames)]
```

```python
# Length of cleaned data
len(df_labels_clean.filename.unique())
```

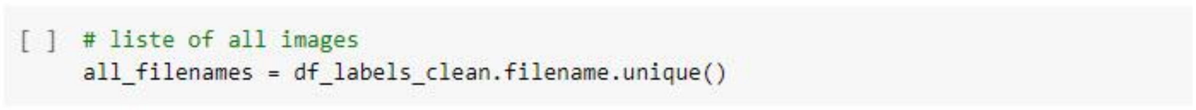
```
553
```



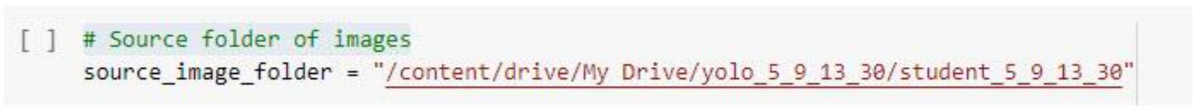
```python
# Class mapping for different label names (to avoid spelling errors)
class_mapping = {
    "velo":0,
    "velo ":0,
    "vÃ©lo":0,
    "vÃ©lo ":0,
    "piéton":1,
    "piéton ":1,
    "piÃ©ton":1,
    "piÃ©ton ":1,
    "pieton":1,
    "pieton ":1,
    "trottinette":2,
    "trottinette ":2,
    "masque":3,
    "masque ":3,
    "mal_mis":4,
    "mal_mis ":4,
    "pas_de_masque":5,
    "pas_de_masque ":5
}
```



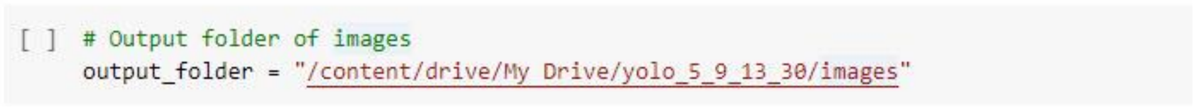
```python
# Import utilities to copy files
from shutil import copy
import os
```



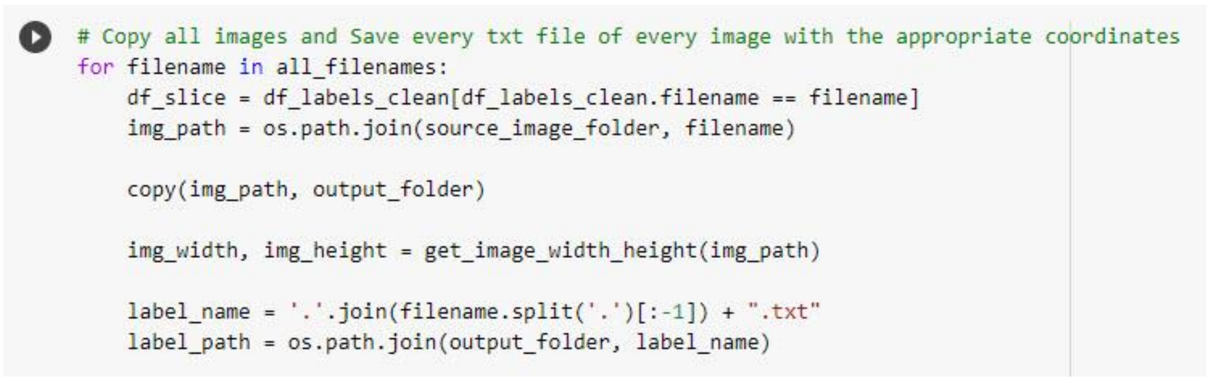
```python
# liste of all images
all_filenames = df_labels_clean.filename.unique()
```



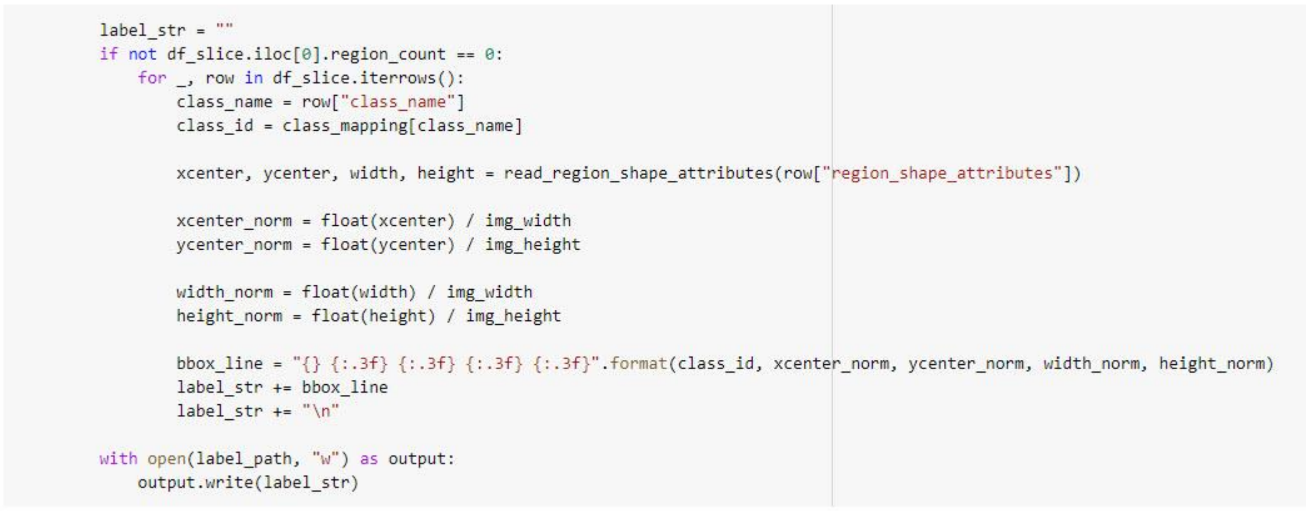
```python
# Source folder of images
source_image_folder = "/content/drive/My Drive/yolo_5_9_13_30/student_5_9_13_30"
```



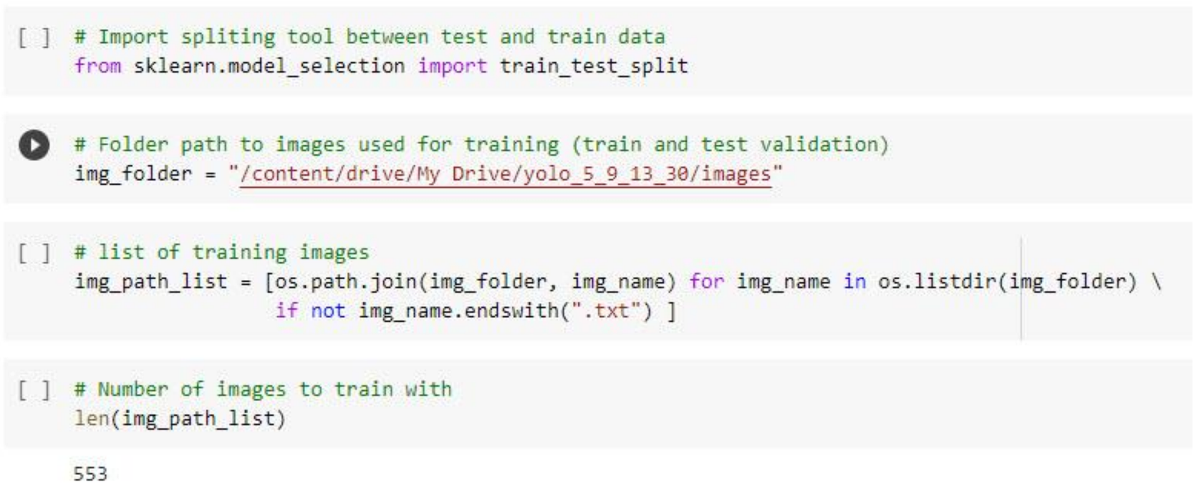
```python
# Output folder of images
output_folder = "/content/drive/My Drive/yolo_5_9_13_30/images"
```



```python
# Copy all images and Save every txt file of every image with the appropriate coordinates
for filename in all_filenames:
    df_slice = df_labels_clean[df_labels_clean.filename == filename]
    img_path = os.path.join(source_image_folder, filename)

    copy(img_path, output_folder)

    img_width, img_height = get_image_width_height(img_path)

    label_name = '.'.join(filename.split('.')[:-1]) + ".txt"
    label_path = os.path.join(output_folder, label_name)
```



```python
    label_str = ""
    if not df_slice.iloc[0].region_count == 0:
        for _, row in df_slice.iterrows():
            class_name = row["class_name"]
            class_id = class_mapping[class_name]

            xcenter, ycenter, width, height = read_region_shape_attributes(row["region_shape_attributes"])

            xcenter_norm = float(xcenter) / img_width
            ycenter_norm = float(ycenter) / img_height

            width_norm = float(width) / img_width
            height_norm = float(height) / img_height

            bbox_line = "{} {:.3f} {:.3f} {:.3f} {:.3f}".format(class_id, xcenter_norm, ycenter_norm, width_norm, height_norm)
            label_str += bbox_line
            label_str += "\n"

    with open(label_path, "w") as output:
        output.write(label_str)
```



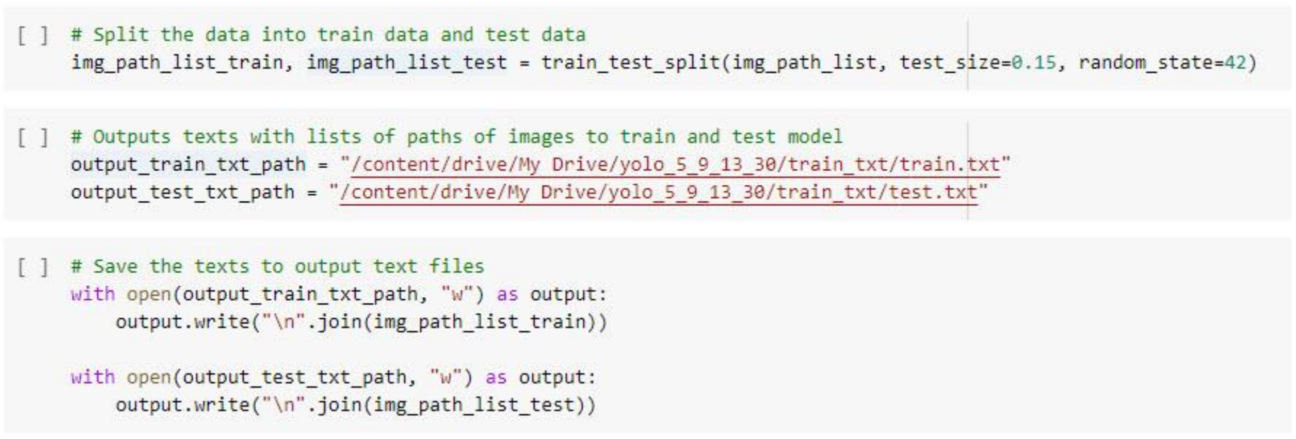
```
# Import spliting tool between test and train data
from sklearn.model_selection import train_test_split
```

```
# Folder path to images used for training (train and test validation)
img_folder = "/content/drive/My Drive/yolo_5_9_13_30/images"
```

```
# list of training images
img_path_list = [os.path.join(img_folder, img_name) for img_name in os.listdir(img_folder) \
                 if not img_name.endswith(".txt") ]
```

```
# Number of images to train with
len(img_path_list)
```

```
553
```

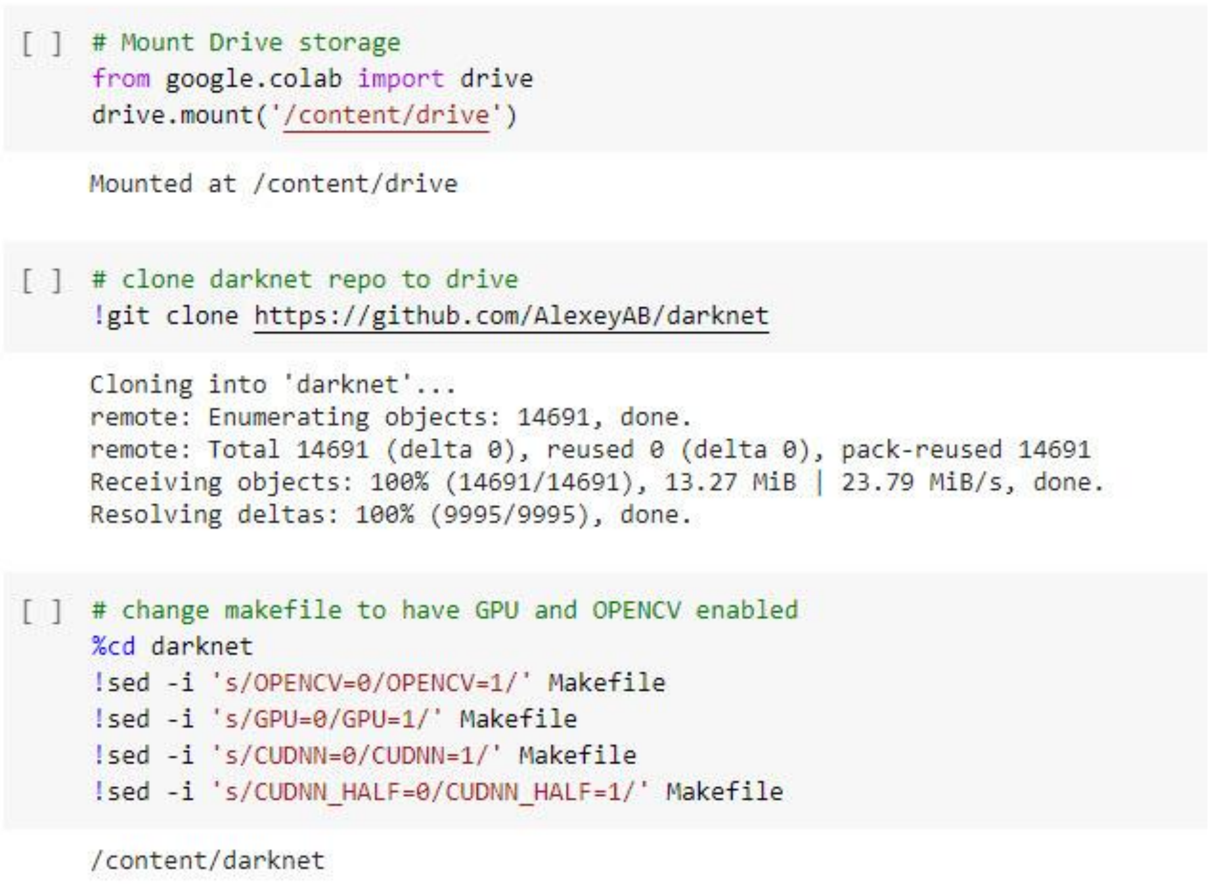


```
# Split the data into train data and test data
img_path_list_train, img_path_list_test = train_test_split(img_path_list, test_size=0.15, random_state=42)
```

```
# Outputs texts with lists of paths of images to train and test model
output_train_txt_path = "/content/drive/My Drive/yolo_5_9_13_30/train_txt/train.txt"
output_test_txt_path = "/content/drive/My Drive/yolo_5_9_13_30/train_txt/test.txt"
```

```
# Save the texts to output text files
with open(output_train_txt_path, "w") as output:
    output.write("\n".join(img_path_list_train))

with open(output_test_txt_path, "w") as output:
    output.write("\n".join(img_path_list_test))
```

Now after conversion to darknet format labelling, we are going to start training and then predicting images, and finally saving results to csv file:



```
# Mount Drive storage
from google.colab import drive
drive.mount('/content/drive')
```

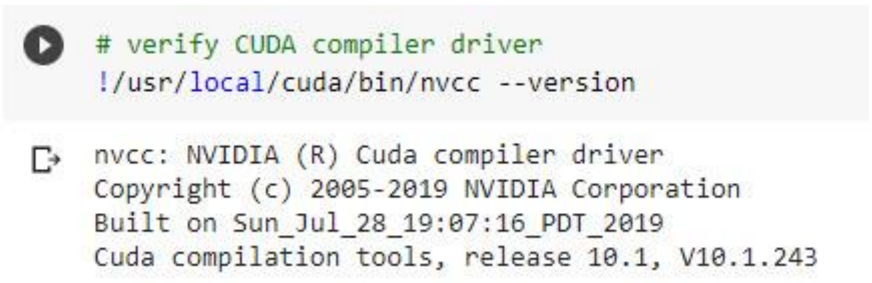
```
Mounted at /content/drive
```

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

```
Cloning into 'darknet'...
remote: Enumerating objects: 14691, done.
remote: Total 14691 (delta 0), reused 0 (delta 0), pack-reused 14691
Receiving objects: 100% (14691/14691), 13.27 MiB | 23.79 MiB/s, done.
Resolving deltas: 100% (9995/9995), done.
```

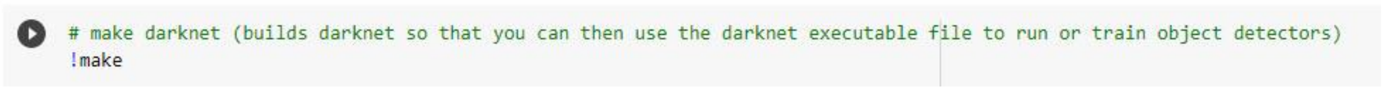
```
# change makefile to have GPU and OPENCV enabled
%cd darknet
!sed -i 's/OPENCV=0/OPENCV=1/' Makefile
!sed -i 's/GPU=0/GPU=1/' Makefile
!sed -i 's/CUDNN=0/CUDNN=1/' Makefile
!sed -i 's/CUDNN_HALF=0/CUDNN_HALF=1/' Makefile
```

```
/content/darknet
```

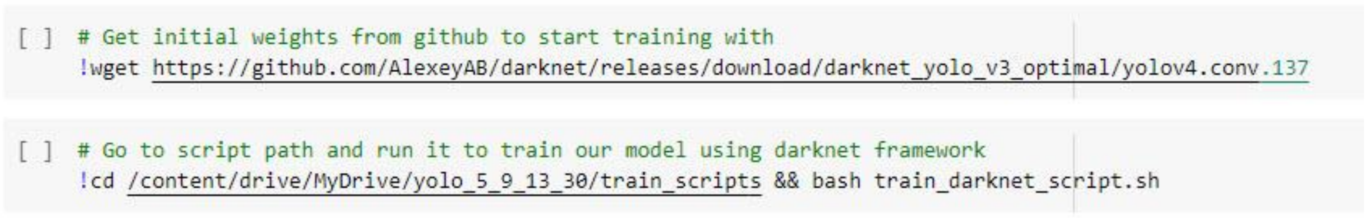


```
# verify CUDA compiler driver
!/usr/local/cuda/bin/nvcc --version
```

```
nvcc: NVIDIA (R) Cuda compiler driver
Copyright (c) 2005-2019 NVIDIA Corporation
Built on Sun_Jul_28_19:07:16_PDT_2019
Cuda compilation tools, release 10.1, V10.1.243
```

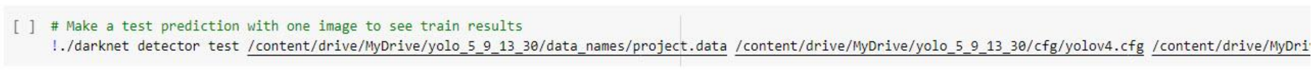


```
# make darknet (builds darknet so that you can then use the darknet executable file to run or train object detectors)
!make
```

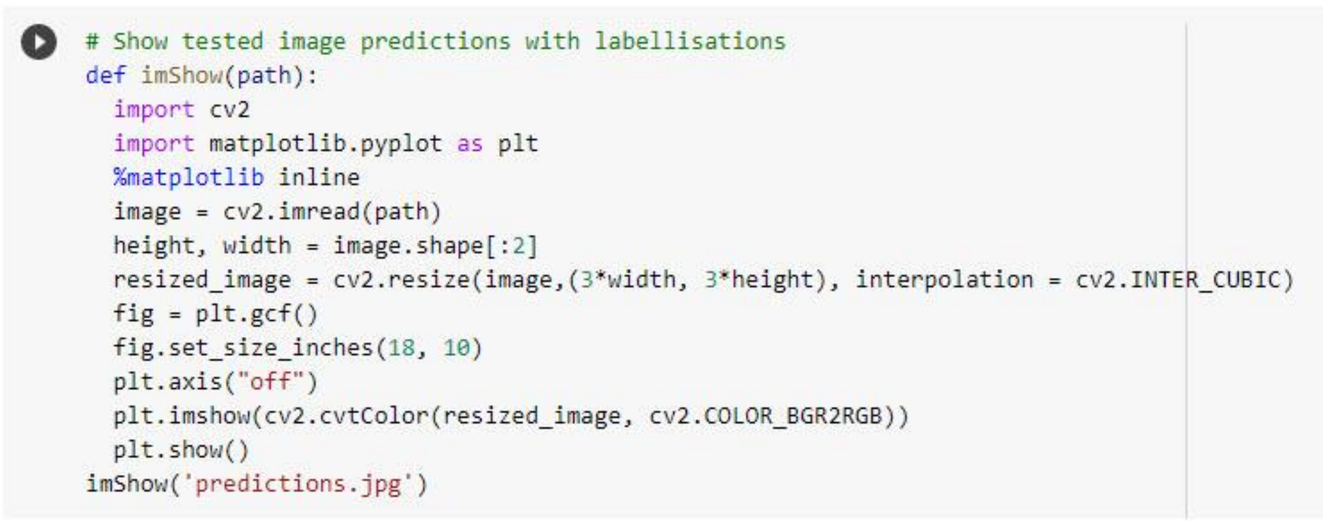


```
# Get initial weights from github to start training with
!wget https://github.com/AlexeyAB/darknet/releases/download/darknet_yolo_v3_optimal/yolov4.conv.137
```

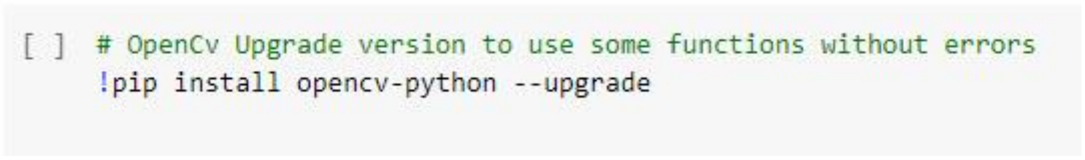
```
# Go to script path and run it to train our model using darknet framework
!cd /content/drive/MyDrive/yolo_5_9_13_30/train_scripts && bash train_darknet_script.sh
```



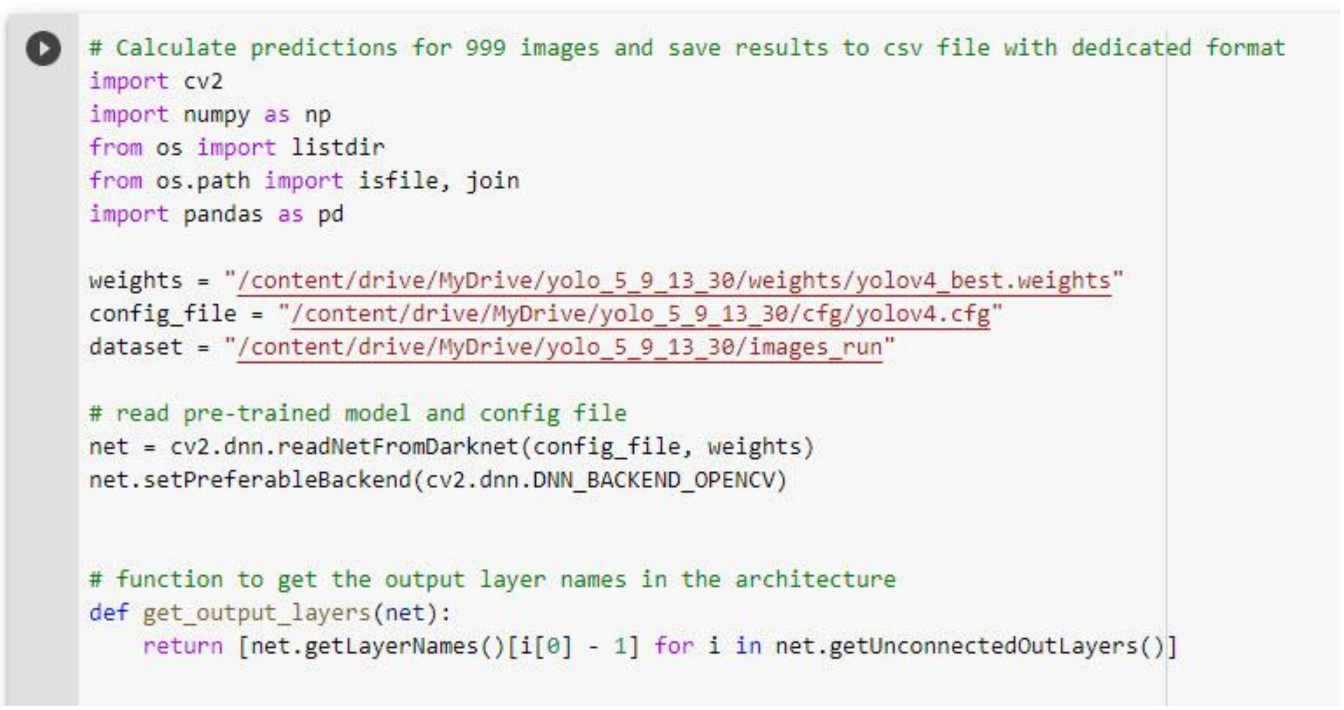
```
# Make a test prediction with one image to see train results
!./darknet detector test /content/drive/MyDrive/yolo_5_9_13_30/data_names/project.data /content/drive/MyDrive/yolo_5_9_13_30/cfg/yolov4.cfg /content/drive/MyDri
```



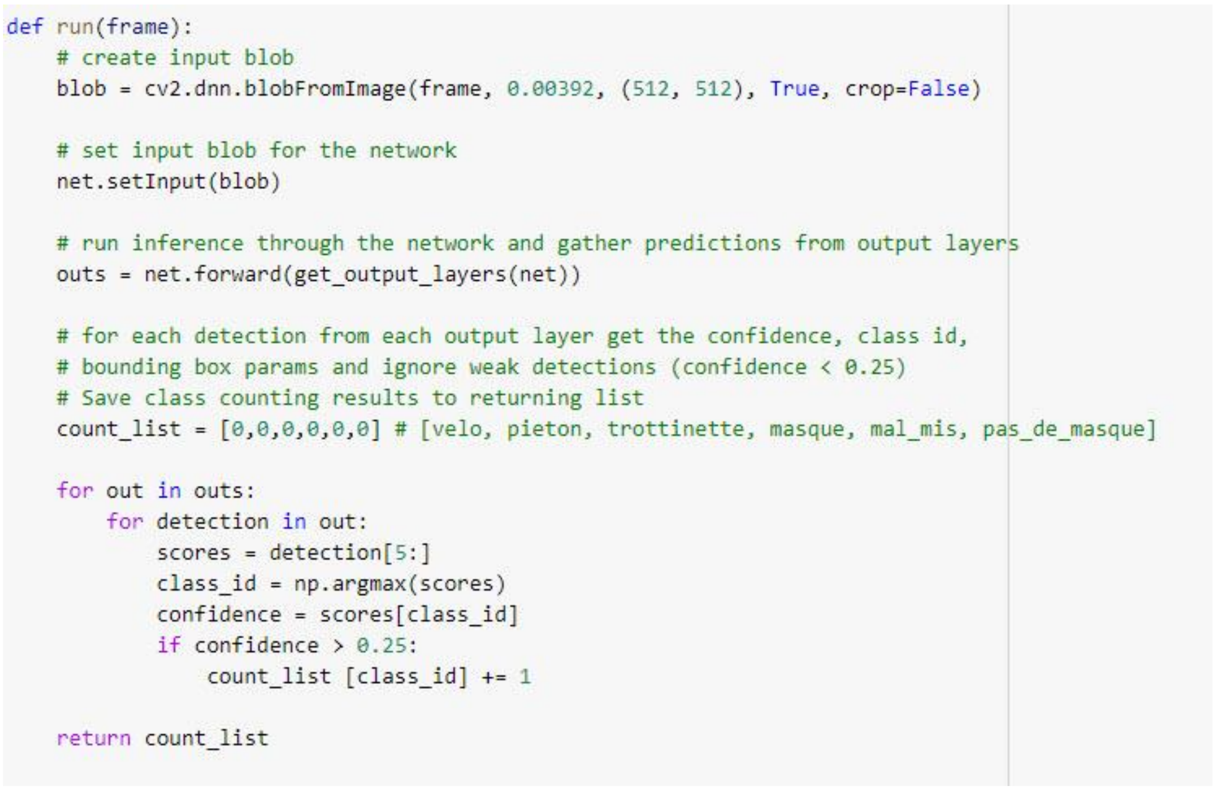
```python
# Show tested image predictions with labellisations
def imShow(path):
  import cv2
  import matplotlib.pyplot as plt
  %matplotlib inline
  image = cv2.imread(path)
  height, width = image.shape[:2]
  resized_image = cv2.resize(image,(3*width, 3*height), interpolation = cv2.INTER_CUBIC)
  fig = plt.gcf()
  fig.set_size_inches(18, 10)
  plt.axis("off")
  plt.imshow(cv2.cvtColor(resized_image, cv2.COLOR_BGR2RGB))
  plt.show()
imShow('predictions.jpg')
```



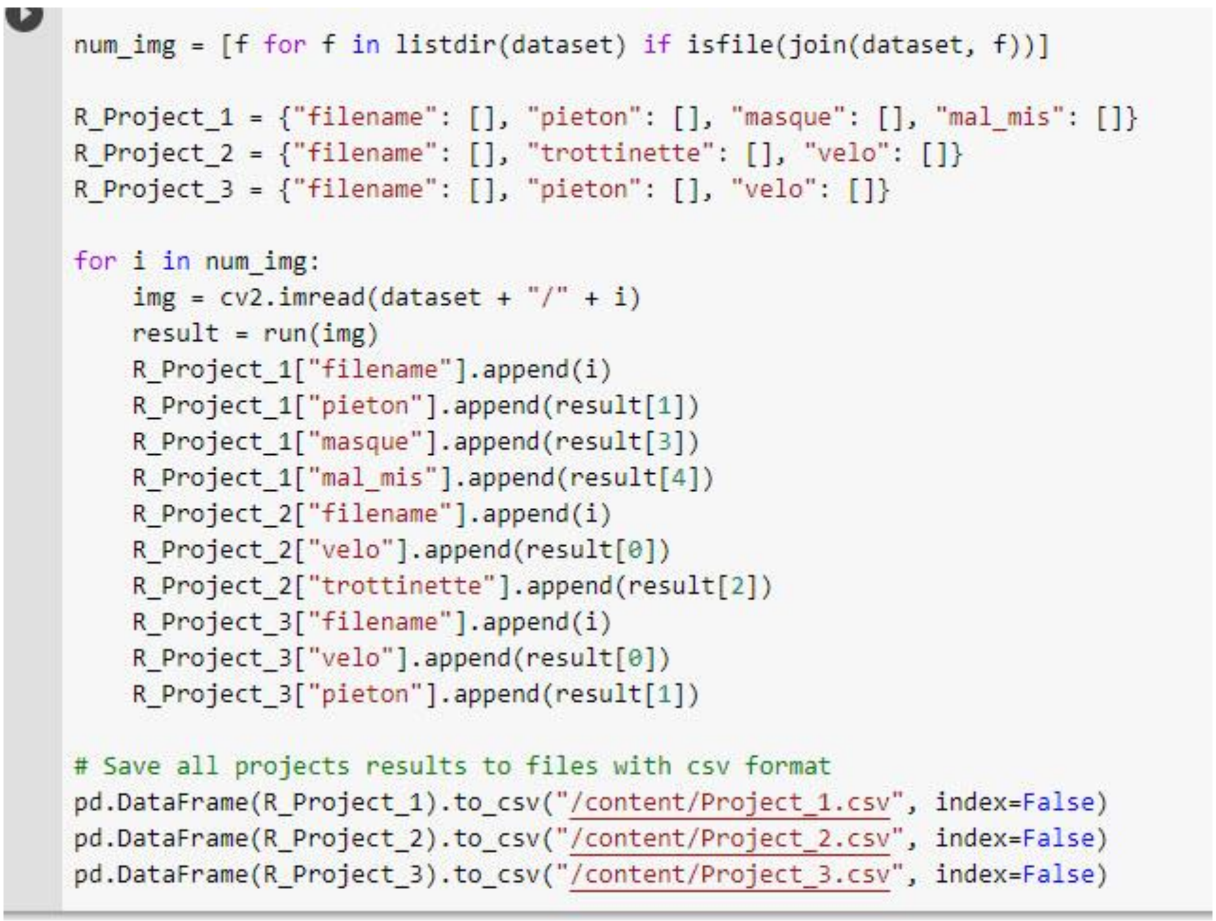
```
[ ] # OpenCv Upgrade version to use some functions without errors
    !pip install opencv-python --upgrade
```



```python
# Calculate predictions for 999 images and save results to csv file with dedicated format
import cv2
import numpy as np
from os import listdir
from os.path import isfile, join
import pandas as pd

weights = "/content/drive/MyDrive/yolo_5_9_13_30/weights/yolov4_best.weights"
config_file = "/content/drive/MyDrive/yolo_5_9_13_30/cfg/yolov4.cfg"
dataset = "/content/drive/MyDrive/yolo_5_9_13_30/images_run"

# read pre-trained model and config file
net = cv2.dnn.readNetFromDarknet(config_file, weights)
net.setPreferableBackend(cv2.dnn.DNN_BACKEND_OPENCV)


# function to get the output layer names in the architecture
def get_output_layers(net):
    return [net.getLayerNames()[i[0] - 1] for i in net.getUnconnectedOutLayers()]
```



```python
def run(frame):
    # create input blob
    blob = cv2.dnn.blobFromImage(frame, 0.00392, (512, 512), True, crop=False)

    # set input blob for the network
    net.setInput(blob)

    # run inference through the network and gather predictions from output layers
    outs = net.forward(get_output_layers(net))

    # for each detection from each output layer get the confidence, class id,
    # bounding box params and ignore weak detections (confidence < 0.25)
    # Save class counting results to returning list
    count_list = [0,0,0,0,0,0] # [velo, pieton, trottinette, masque, mal_mis, pas_de_masque]

    for out in outs:
        for detection in out:
            scores = detection[5:]
            class_id = np.argmax(scores)
            confidence = scores[class_id]
            if confidence > 0.25:
                count_list [class_id] += 1

    return count_list
```



```python
num_img = [f for f in listdir(dataset) if isfile(join(dataset, f))]

R_Project_1 = {"filename": [], "pieton": [], "masque": [], "mal_mis": []}
R_Project_2 = {"filename": [], "trottinette": [], "velo": []}
R_Project_3 = {"filename": [], "pieton": [], "velo": []}

for i in num_img:
    img = cv2.imread(dataset + "/" + i)
    result = run(img)
    R_Project_1["filename"].append(i)
    R_Project_1["pieton"].append(result[1])
    R_Project_1["masque"].append(result[3])
    R_Project_1["mal_mis"].append(result[4])
    R_Project_2["filename"].append(i)
    R_Project_2["velo"].append(result[0])
    R_Project_2["trottinette"].append(result[2])
    R_Project_3["filename"].append(i)
    R_Project_3["velo"].append(result[0])
    R_Project_3["pieton"].append(result[1])

# Save all projects results to files with csv format
pd.DataFrame(R_Project_1).to_csv("/content/Project_1.csv", index=False)
pd.DataFrame(R_Project_2).to_csv("/content/Project_2.csv", index=False)
pd.DataFrame(R_Project_3).to_csv("/content/Project_3.csv", index=False)
```