# Object Detection Using YOLOV4 & Darknet

**[1]Nikita Gaurihar**
**[2]Anshuman Karan**

## 1. **Abstract**:

With each passing day, machine learning applications are improving in accuracy. Availability of huge datasets combined with strategies like pattern recognition, object detection, Recommendation Engine are showing exponentially better results compared to previous years. As world is getting used to data-oriented applications, traditional methods in areas like surveillance becoming obsolete and costly. Traditionally, we use CCTV for surveillance, and we need a human proctor to monitor the abnormal activities. This project aims to produce a robust real-time application that helps in detection and identification of all instances of various objects in the image, video, or live streaming through webcam. The algorithms related to this study such as Darknet, YoloV4, Data Scrapping, Data Annotation, etc. will be implemented using python programming.
**Keywords**: **Object Detection, YOLO V4, Darknet, Data Annotation, etc.**

## 2. **Introduction**:

Object detection is a computer vision and image processing technology that deals with detecting instances of semantic objects of a certain class like a person, vehicle, trees, animals, etc. in images and videos. A few of the renowned applications of object detection in the field of computer vision are face detection, pedestrian detection, image retrieval, video surveillance, etc.

### 2.1 Object Detection Methods:

The dramatic advancements in the field of Computer Vision can be attributed to a variety of object detecting methods. Object detection's primary idea is that it locates objects or classes from a given image or set of images. 'Object localization' is the concept of detecting the location of an object in each image, whereas 'Object classification' is the concept of determining what that object is. The following are some of the object detection methods:
a. Fast R-CNN
b. YOLO (You Only Live Once) (You only look once)
c. HOG - Histogram of Oriented Gradients
d. Convolutional Neural Network, etc.

In this project, we'll look at the YOLO V4 object identification technology, which makes use of the open-source neural network framework known as Darknet.

### 2.2 Important Terminologies:

#### A. Darknet:
Darknet is a neural network framework that is open source. It is a framework for real-time object identification that is both fast and accurate. It is faster than other frameworks since it is written in the C and CUDA languages.

## B. Yolo:

- You Only Look Once, Version 3 (YOLOv3) is a real-time object detection system that detects specific objects in movies, live feeds, and photos. YOLO detects an item by learning features from a deep convolutional neural network. Keras or OpenCV deep learning libraries are used to create YOLOv3.
- The features learnt by CNNs are fed into a classifier, which predicts detection. The prediction of YOLOv3 is based on an 11-convolutional layer, hence the term "you only look once." The prediction map is the same size as the feature map. YOLO has the benefit of being faster than other networks while maintaining accuracy.
- An image is initially divided into a grid via the YOLOv3 algorithm. Each grid cell forecasts the placement of a certain number of boundary boxes around items that score well in the predefined classes. There is a number assigned to each boundary box.
- The YOLOv3 algorithm first separates an image into a grid. Each grid cell predicts some number of bounding boxes around objects that score highly with the predefined classes. Each bounding box has a respective confidence score of how accurate it assumes that prediction should be and detects only one object per bounding box. The bounding boxes are generated by clustering the dimensions of the ground truth boxes from the original dataset to find the most common shapes and sizes.

## C. Pretrained Model:

A pre-trained model is one that has already been taught to address a similar problem. Instead of starting from scratch to solve a similar problem, we start with a model that has been trained on a similar problem. This saves time and allows the model to operate more efficiently while staying within the time constraints.

## D. Data Annotation:

If we have labeled data in machine learning, it indicates that our data has been marked up, or annotated, to show the target, or what we want our machine learning model to predict. For example- A data labeler can, make use of frame-by-frame video labeling tools in computer vision for autonomous vehicles to show the location of street signs, pedestrians, or other vehicles.

## 3. Dataset Used:

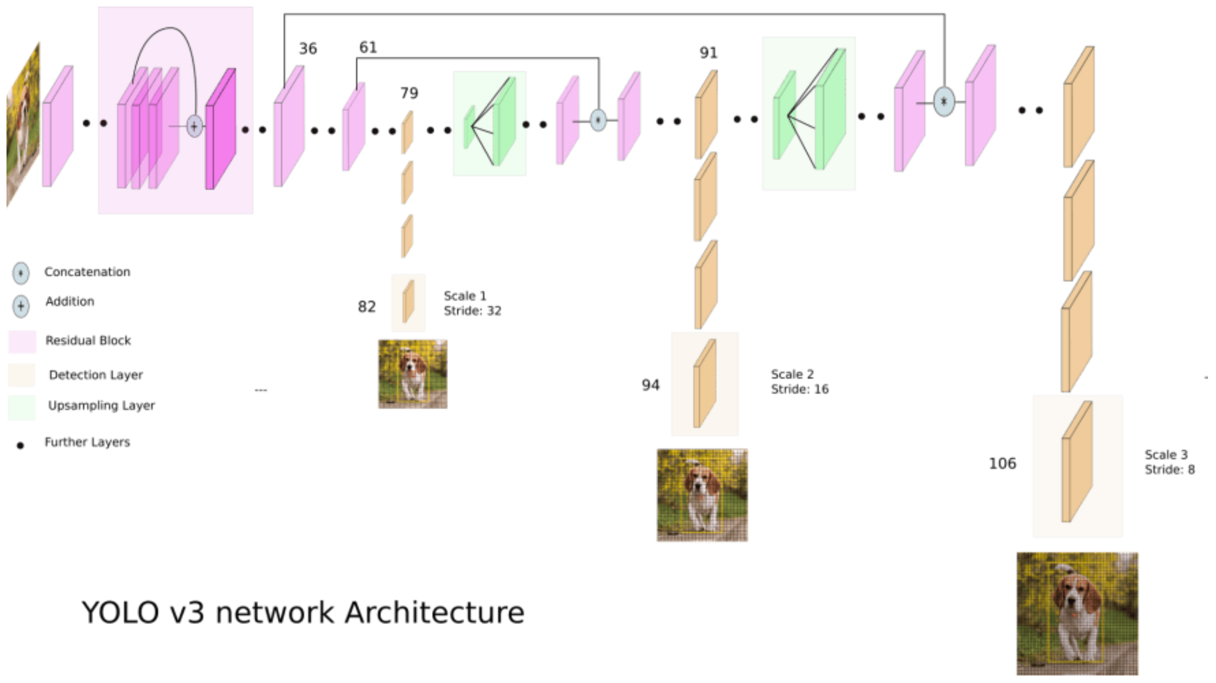
The famous TensorFlow's COCO dataset with 91 classes. The link to the dataset is as follows – https://www.tensorflow.org/datasets/catalog/coco

## 4. Understanding YOLO Architecture:

**YOLO** Stands for – You only look once is a Deep Learning Architecture majorly used for fast processing of applications related to Image Classification, Object Detection and Semantic Segmentation. YOLO v3 uses a variant of Darknet, which originally has 53-layer network trained on ImageNet. YOLO v3 uses binary cross-entropy for calculating the classification loss for each label while object confidence and class predictions are predicted through logistic regression. Hyperparameters used are class_threshold, Non-Max Suppression Threshold, input_height and input_shape.

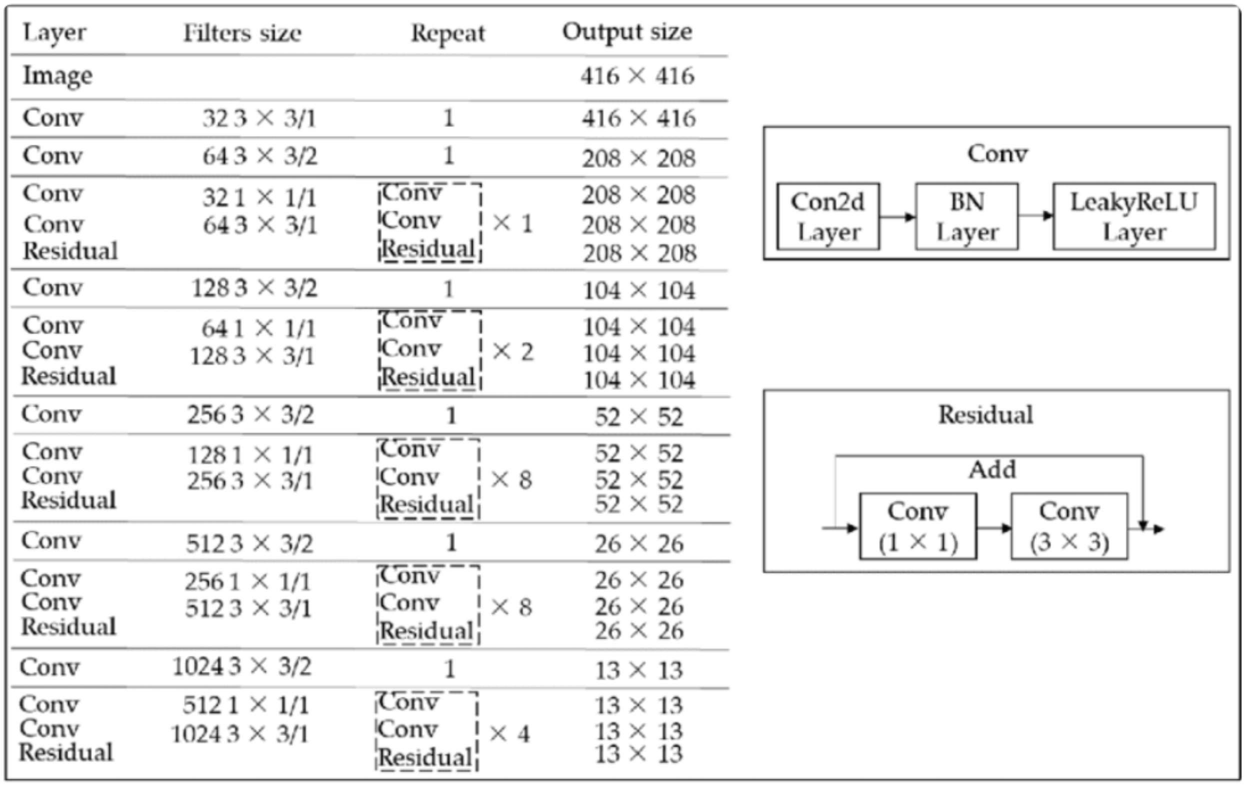
**Step by step description of architecture is mentioned as follows [3]:**

1. Batches of images with shape in the format (m, 416, 416, 3) are provided as an input.
2. Once the input is received by the YOLOV3 model, it passes the input information to Convolutional Neural network as shown in the figure.
3. The last two dimensions of output are flattened to receive output of (19,19,425) where 19x19 grid returns 425 number.
   – Further calculation involves:
   - 425 = 5 * 85,
     where 5 = number of anchor boxes per grid.
   - 85 = 5 + 80,
     where 5 is (pc, bx, by, bh, bw) and
   - 80 = the number of unique classes for detection
4. The output is a list of bounding boxes along with the recognized classes. Each bounding box is represented by 6 numbers (pc, bx, by, bh, bw, c).
5. Finally, we do the IoU (Intersection over Union) and Non-Max Suppression to avoid selecting overlapping boxes.



YOLO v3 network Architecture

## CNN architecture of Darknet-53

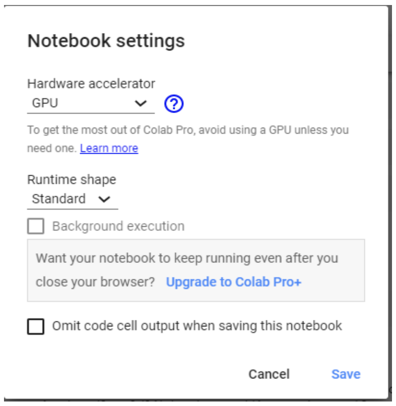


| Layer | Filters size | Repeat | Output size |
|---|---|---|---|
| Image | | | 416 × 416 |
| Conv | 32 3 × 3/1 | 1 | 416 × 416 |
| Conv | 64 3 × 3/2 | 1 | 208 × 208 |
| Conv | 32 1 × 1/1 | Conv | 208 × 208 |
| Conv | 64 3 × 3/1 | Conv × 1 | 208 × 208 |
| Residual | | Residual | 208 × 208 |
| Conv | 128 3 × 3/2 | 1 | 104 × 104 |
| Conv | 64 1 × 1/1 | Conv | 104 × 104 |
| Conv | 128 3 × 3/1 | Conv × 2 | 104 × 104 |
| Residual | | Residual | 104 × 104 |
| Conv | 256 3 × 3/2 | 1 | 52 × 52 |
| Conv | 128 1 × 1/1 | Conv | 52 × 52 |
| Conv | 256 3 × 3/1 | Conv × 8 | 52 × 52 |
| Residual | | Residual | 52 × 52 |
| Conv | 512 3 × 3/2 | 1 | 26 × 26 |
| Conv | 256 1 × 1/1 | Conv | 26 × 26 |
| Conv | 512 3 × 3/1 | Conv × 8 | 26 × 26 |
| Residual | | Residual | 26 × 26 |
| Conv | 1024 3 × 3/2 | 1 | 13 × 13 |
| Conv | 512 1 × 1/1 | Conv | 13 × 13 |
| Conv | 1024 3 × 3/1 | Conv × 4 | 13 × 13 |
| Residual | | Residual | 13 × 13 |

**5. Python Implementation:**
I have used documentation from YOLO Official Website [1] –
https://pjreddie.com/darknet/yolo/

**Step 1: Setting up the GPU within notebook**

GPU is used when higher computational power is required. For Deep Learning applications, where simultaneous operations are performed, computational complexity increases. With the help of GPU, the distribution of training process significantly scales up the machine learning operations. Following settings need to be done before we start training our model.



Notebook settings

Hardware accelerator
GPU

To get the most out of Colab Pro, avoid using a GPU unless you need one. Learn more

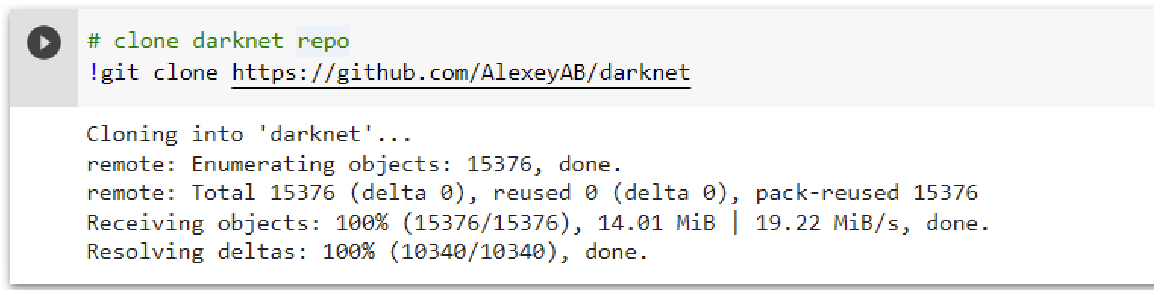
Runtime shape
Standard

☐ Background execution

Want your notebook to keep running even after you close your browser? Upgrade to Colab Pro+

☐ Omit code cell output when saving this notebook

**Step 2: Setting up the Darknet-**
To implement YoloV3, we need to first implement Darknet framework. This can be done by cloning the darknet repository. We can clone either pjreddie's Git or AlexyAB's Git. Here I am using, AlexyAB's GitHub repository.



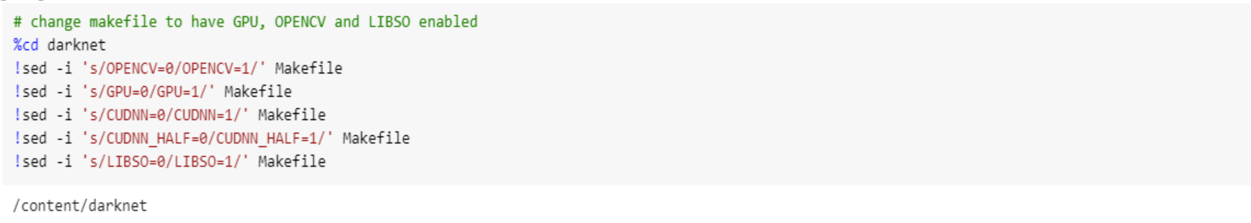
```
# clone darknet repo
!git clone https://github.com/AlexeyAB/darknet
```

```
Cloning into 'darknet'...
remote: Enumerating objects: 15376, done.
remote: Total 15376 (delta 0), reused 0 (delta 0), pack-reused 15376
Receiving objects: 100% (15376/15376), 14.01 MiB | 19.22 MiB/s, done.
Resolving deltas: 100% (10340/10340), done.
```

Since Darknet is fast, readily available via GitHub and supports CPU & GPU computation, it is widely used in image and object classification with number of classes as high as 1000.

**Step 3: Made changes in the makefile to enable OPENCV and GPU**

1. The deep neural network modules in OpenCV allow you to-
   - load a pre-trained network from disk
   - perform predictions on an input image,
   - show the results by creating our own customized computer vision pipeline.
2. changing makefile to have GPU and OPENCV enabled

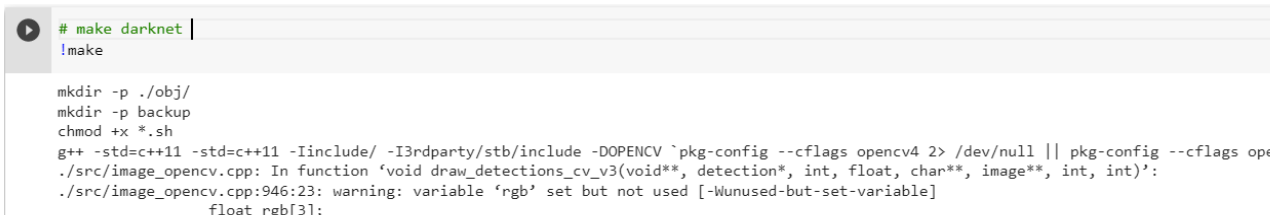


```
# change makefile to have GPU, OPENCV and LIBSO enabled
%cd darknet
!sed -i 's/OPENCV=0/OPENCV=1/' Makefile
!sed -i 's/GPU=0/GPU=1/' Makefile
!sed -i 's/CUDNN=0/CUDNN=1/' Makefile
!sed -i 's/CUDNN_HALF=0/CUDNN_HALF=1/' Makefile
!sed -i 's/LIBSO=0/LIBSO=1/' Makefile
```

```
/content/darknet
```

**Step 4: To organize our code compilations, using 'make' command.**

Below command is used to compile and build darknet to use the darknet executable file to train obj ect detector model



```
# make darknet
!make
```

```
mkdir -p ./obj/
mkdir -p backup
chmod +x *.sh
g++ -std=c++11 -std=c++11 -Iinclude/ -I3rdparty/stb/include -DOPENCV `pkg-config --cflags opencv4 2> /dev/null || pkg-config --cflags ope
./src/image_opencv.cpp: In function 'void draw_detections_cv_v3(void**, detection*, int, float, char**, image**, int, int)':
./src/image_opencv.cpp:946:23: warning: variable 'rgb' set but not used [-Wunused-but-set-variable]
                 float rgb[3];
```

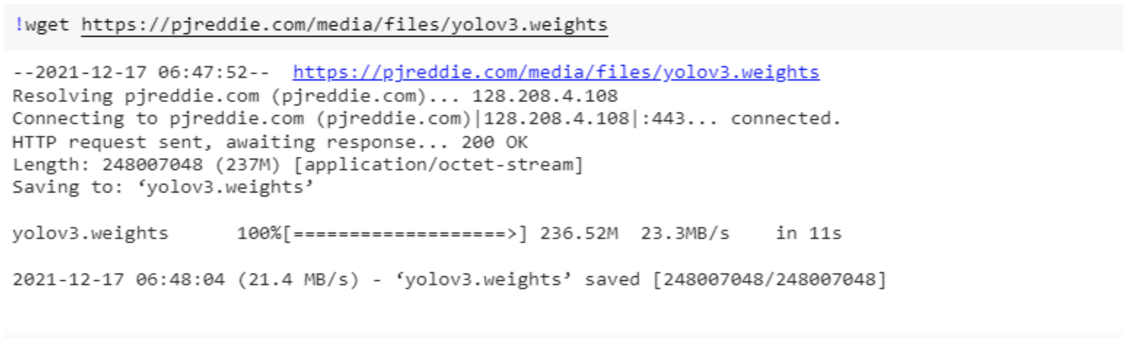
**Interpretation**:

- In the darknet directory, now we will get a config file for Yolo in 'cfg' Subdirectory.

- The neural network model architecture is stored in the yolov3.cfg file, and the pre-trained weights of the neural network are stored in yolov3.weights.
- The list of 80 object class that the model will be detecting is mentioned in a file called 'coco. names'.
- The model has been trained only on these 80 object classes.

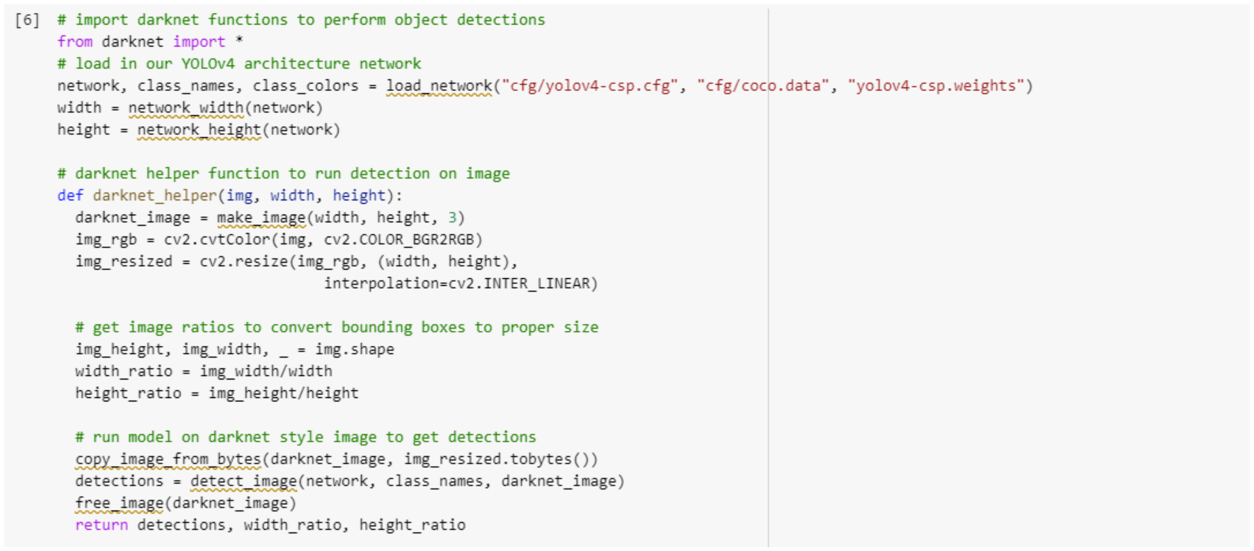
**Step 5: Loading the scales YoloV3 weights file that is pretrained to detect 80 classes**

- YOLOv3 has been trained already on the coco dataset which has 80 classes that it can predic t. Using these pretrained weights so that we can run YOLOv4 on these pretrained classes, get detections.
- This step downloads the weights for the convolutional layers of the YOLOv4 network.



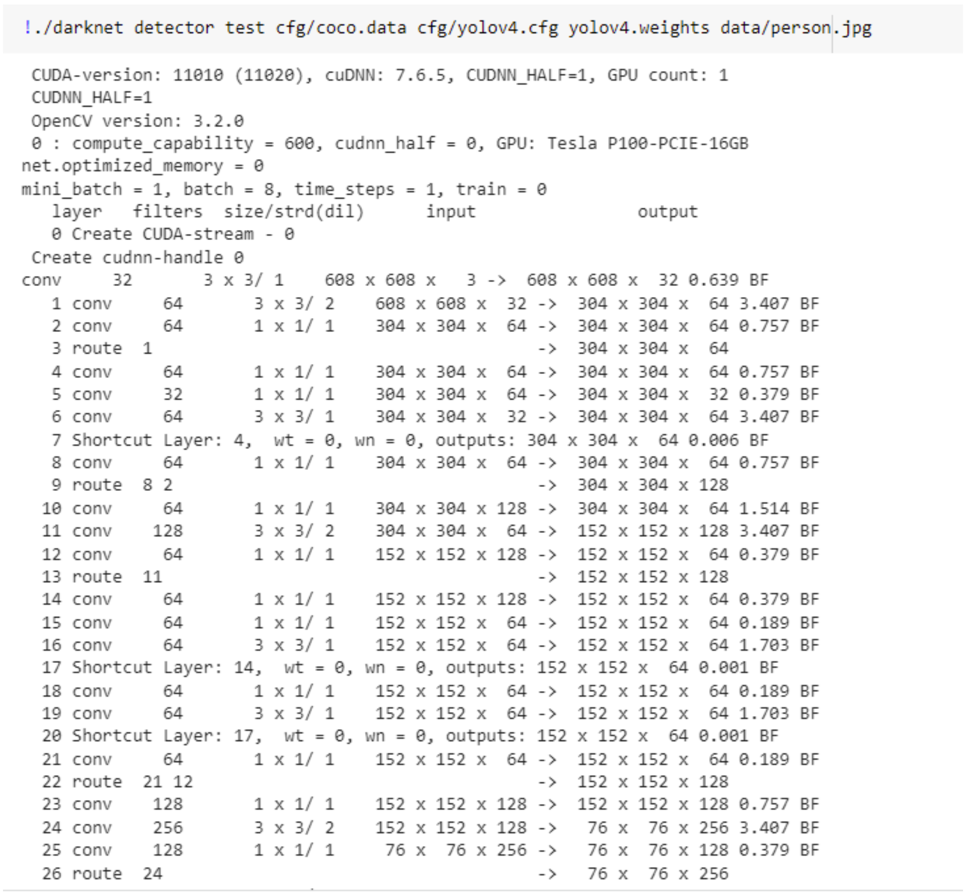
```
!wget https://pjreddie.com/media/files/yolov3.weights

--2021-12-17 06:47:52--  https://pjreddie.com/media/files/yolov3.weights
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: 248007048 (237M) [application/octet-stream]
Saving to: 'yolov3.weights'

yolov3.weights      100%[===================>] 236.52M  23.3MB/s    in 11s

2021-12-17 06:48:04 (21.4 MB/s) - 'yolov3.weights' saved [248007048/248007048]
```

**Step 6: To implement YoloV3 with Darknet framework in Python, the following pre-built function will be used –**



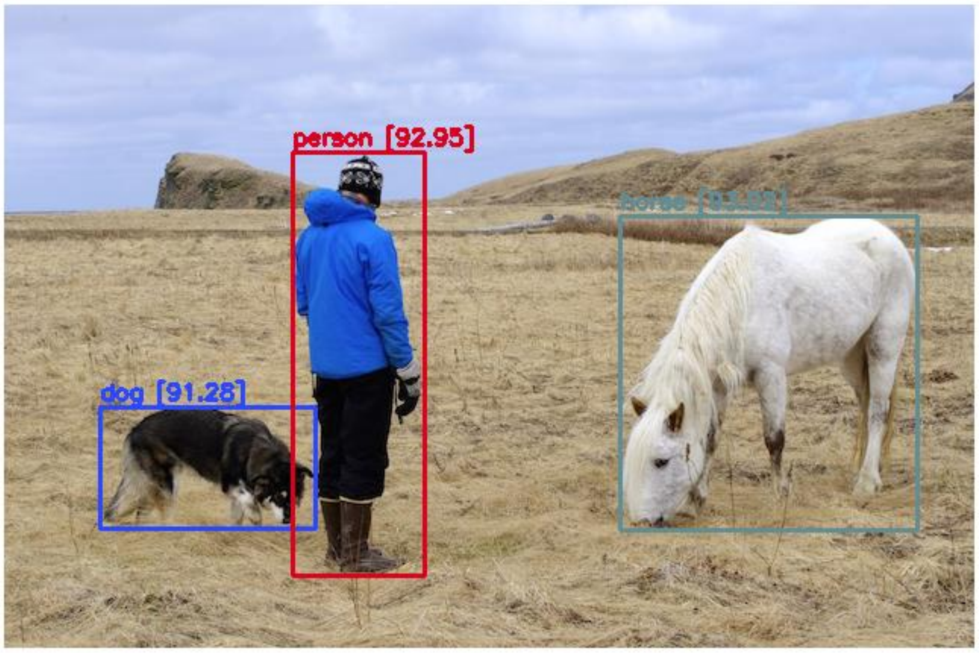
```
[6] # import darknet functions to perform object detections
    from darknet import *
    # load in our YOLOv4 architecture network
    network, class_names, class_colors = load_network("cfg/yolov4-csp.cfg", "cfg/coco.data", "yolov4-csp.weights")
    width = network_width(network)
    height = network_height(network)

    # 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
```

## Step 7: Detection using a pre-trained model:

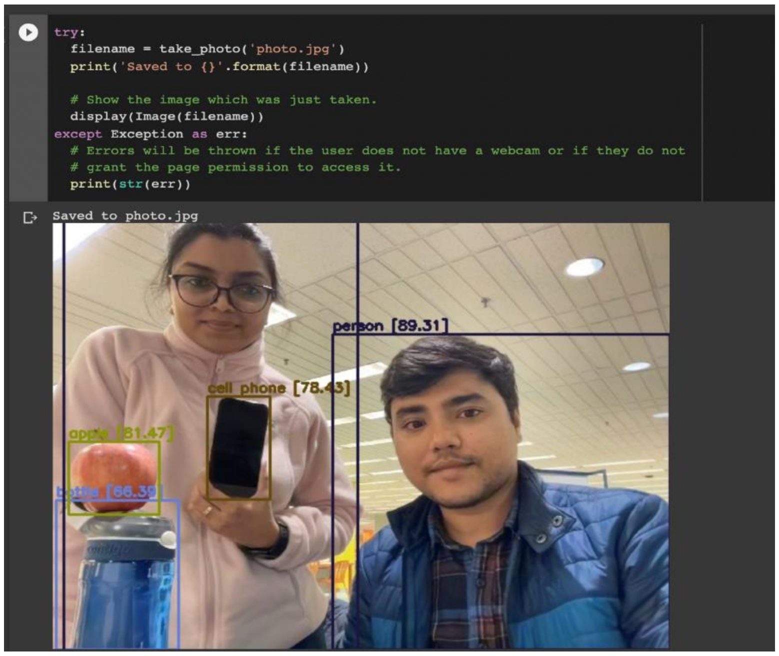


```
!./darknet detector test cfg/coco.data cfg/yolov4.cfg yolov4.weights data/person.jpg
```

```
 CUDA-version: 11010 (11020), cuDNN: 7.6.5, CUDNN_HALF=1, GPU count: 1
 CUDNN_HALF=1
 OpenCV version: 3.2.0
 0 : compute_capability = 600, cudnn_half = 0, GPU: Tesla P100-PCIE-16GB
net.optimized_memory = 0
mini_batch = 1, batch = 8, time_steps = 1, train = 0
   layer   filters  size/strd(dil)      input                output
   0 Create CUDA-stream - 0
 Create cudnn-handle 0
conv     32       3 x 3/ 1    608 x 608 x   3 ->  608 x 608 x  32 0.639 BF
   1 conv     64       3 x 3/ 2    608 x 608 x  32 ->  304 x 304 x  64 3.407 BF
   2 conv     64       1 x 1/ 1    304 x 304 x  64 ->  304 x 304 x  64 0.757 BF
   3 route  1                                    ->  304 x 304 x  64
   4 conv     64       1 x 1/ 1    304 x 304 x  64 ->  304 x 304 x  64 0.757 BF
   5 conv     32       1 x 1/ 1    304 x 304 x  64 ->  304 x 304 x  32 0.379 BF
   6 conv     64       3 x 3/ 1    304 x 304 x  32 ->  304 x 304 x  64 3.407 BF
   7 Shortcut Layer: 4,  wt = 0, wn = 0, outputs: 304 x 304 x  64 0.006 BF
   8 conv     64       1 x 1/ 1    304 x 304 x  64 ->  304 x 304 x  64 0.757 BF
   9 route  8 2                                  ->  304 x 304 x 128
  10 conv     64       1 x 1/ 1    304 x 304 x 128 ->  304 x 304 x  64 1.514 BF
  11 conv    128       3 x 3/ 2    304 x 304 x  64 ->  152 x 152 x 128 3.407 BF
  12 conv     64       1 x 1/ 1    152 x 152 x 128 ->  152 x 152 x  64 0.379 BF
  13 route  11                                   ->  152 x 152 x 128
  14 conv     64       1 x 1/ 1    152 x 152 x 128 ->  152 x 152 x  64 0.379 BF
  15 conv     64       1 x 1/ 1    152 x 152 x  64 ->  152 x 152 x  64 0.189 BF
  16 conv     64       3 x 3/ 1    152 x 152 x  64 ->  152 x 152 x  64 1.703 BF
  17 Shortcut Layer: 14,  wt = 0, wn = 0, outputs: 152 x 152 x  64 0.001 BF
  18 conv     64       1 x 1/ 1    152 x 152 x  64 ->  152 x 152 x  64 0.189 BF
  19 conv     64       3 x 3/ 1    152 x 152 x  64 ->  152 x 152 x  64 1.703 BF
  20 Shortcut Layer: 17,  wt = 0, wn = 0, outputs: 152 x 152 x  64 0.001 BF
  21 conv     64       1 x 1/ 1    152 x 152 x  64 ->  152 x 152 x  64 0.189 BF
  22 route  21 12                                ->  152 x 152 x 128
  23 conv    128       1 x 1/ 1    152 x 152 x 128 ->  152 x 152 x 128 0.757 BF
  24 conv    256       3 x 3/ 2    152 x 152 x 128 ->   76 x  76 x 256 3.407 BF
  25 conv    128       1 x 1/ 1     76 x  76 x 256 ->   76 x  76 x 128 0.379 BF
  26 route  24                                   ->   76 x  76 x 256
```

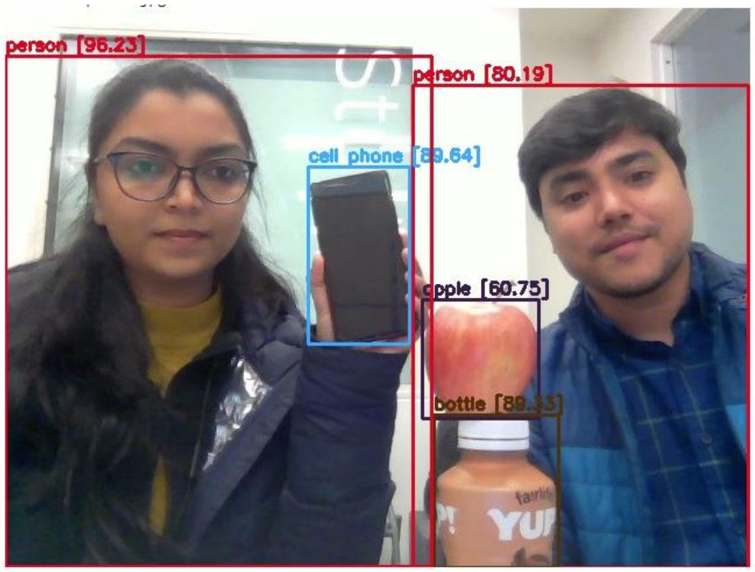


**Step 8: YoloV3 implementation on images taken via Web-Cam-**



```
try:
  filename = take_photo('photo.jpg')
  print('Saved to {}'.format(filename))

  # Show the image which was just taken.
  display(Image(filename))
except Exception as err:
  # Errors will be thrown if the user does not have a webcam or if they do not
  # grant the page permission to access it.
  print(str(err))
```

```
Saved to photo.jpg
```



**Conclusion:**

After implementing YoloV3 using Darknet framework on TensorFlow's COCO dataset using pretrained models, the model can classify the objects shown in image taken via web cam with the accuracy mentioned above the respective bounding box.

## Custom Object detection using YOLOV3

Similar to TensorFlow's COCO dataset, we can customize our own dataset by creating labels and adding new classes other than what coco dataset is offering. For the demonstration purpose, we have implemented Data Annotation techniques for two classes – 'Headphones' and 'Masks'.

### 1. Data Collection

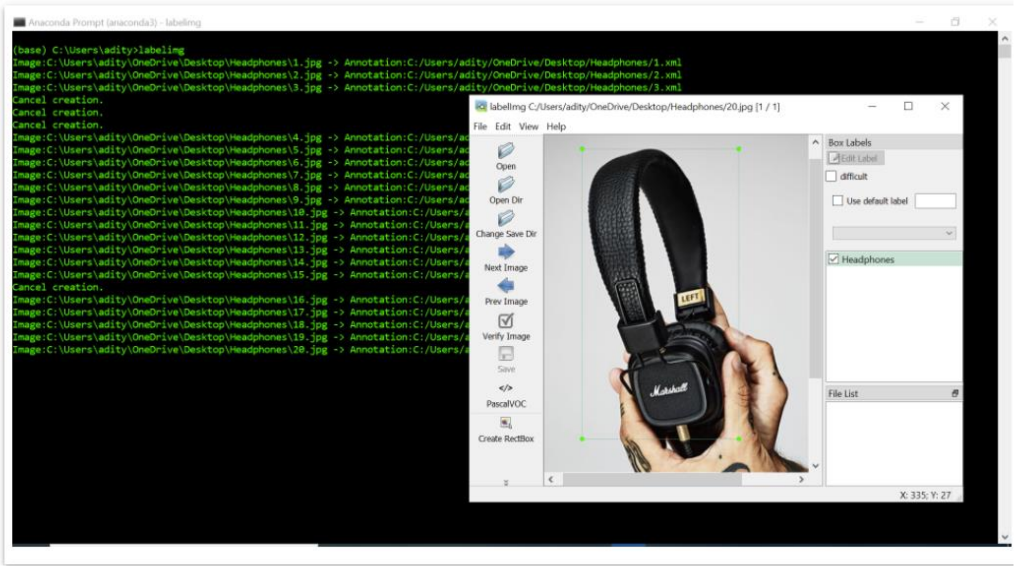
There are different image Data Collection techniques, namely –

a. Web Scrapping using Beautiful Soup 4
b. Simple Image Downloader Tool
c. Google open source, etc.

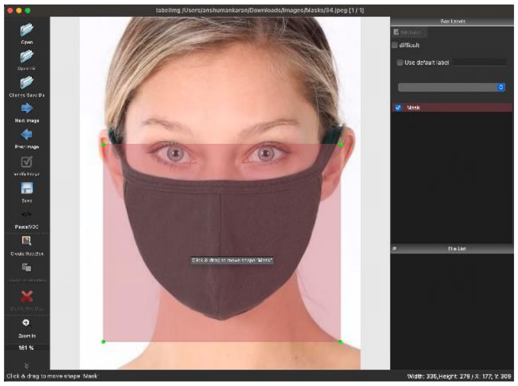
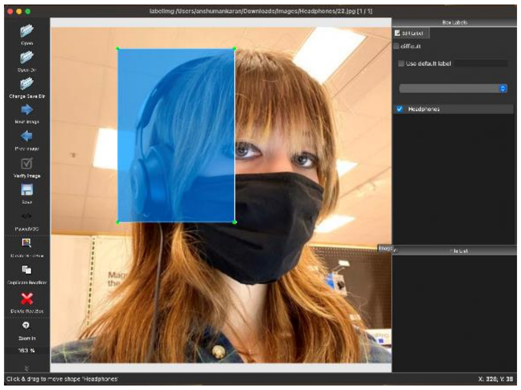
To collect the image dataset of objects, we have downloaded images manually from google.

### 2. Data Annotation:

- There are various ways to perform image data collection for custom dataset like web-scrapping, manually clicking images, downloading the images from sites like google or printe rest.
- Due to time constraints, we have downloaded the images of two labels - headphones and m asks (which are not present in COCO Dataset) from google and performed annotation using ' labelimg' tool.
- In this way, we have created a dataset of 50 images of each of these two classes.
- We have made a bounding box using 'Create Reactbox' function in labelimg tool around the object that we are going to detect. In our case, it was 'Headphones' and 'masks'.
- Once these annotations were completed, we have splitted our image dataset with xml file (this xml file will have coordinates for the location of bounding box) in train and test folders.
- Then, as shown in the figure below, we have used 'labelimg' tool to assign a bounding box on the object in the image.



```
(base) C:\Users\adity>labelimg
Image:C:\Users\adity\OneDrive\Desktop\Headphones\1.jpg -> Annotation:C:/Users/adity/OneDrive/Desktop/Headphones/1.xml
Image:C:\Users\adity\OneDrive\Desktop\Headphones\2.jpg -> Annotation:C:/Users/adity/OneDrive/Desktop/Headphones/2.xml
Image:C:\Users\adity\OneDrive\Desktop\Headphones\3.jpg -> Annotation:C:/Users/adity/OneDrive/Desktop/Headphones/3.xml
Cancel creation.
Cancel creation.
Cancel creation.
Image:C:\Users\adity\OneDrive\Desktop\Headphones\4.jpg -> Annotation:C:/Users/ad
Image:C:\Users\adity\OneDrive\Desktop\Headphones\5.jpg -> Annotation:C:/Users/ad
Image:C:\Users\adity\OneDrive\Desktop\Headphones\6.jpg -> Annotation:C:/Users/ad
Image:C:\Users\adity\OneDrive\Desktop\Headphones\7.jpg -> Annotation:C:/Users/ad
Image:C:\Users\adity\OneDrive\Desktop\Headphones\8.jpg -> Annotation:C:/Users/ad
Image:C:\Users\adity\OneDrive\Desktop\Headphones\9.jpg -> Annotation:C:/Users/ad
Image:C:\Users\adity\OneDrive\Desktop\Headphones\10.jpg -> Annotation:C:/Users/a
Image:C:\Users\adity\OneDrive\Desktop\Headphones\11.jpg -> Annotation:C:/Users/a
Image:C:\Users\adity\OneDrive\Desktop\Headphones\12.jpg -> Annotation:C:/Users/a
Image:C:\Users\adity\OneDrive\Desktop\Headphones\13.jpg -> Annotation:C:/Users/a
Image:C:\Users\adity\OneDrive\Desktop\Headphones\14.jpg -> Annotation:C:/Users/a
Image:C:\Users\adity\OneDrive\Desktop\Headphones\15.jpg -> Annotation:C:/Users/a
Cancel creation.
Image:C:\Users\adity\OneDrive\Desktop\Headphones\16.jpg -> Annotation:C:/Users/a
Image:C:\Users\adity\OneDrive\Desktop\Headphones\17.jpg -> Annotation:C:/Users/a
Image:C:\Users\adity\OneDrive\Desktop\Headphones\18.jpg -> Annotation:C:/Users/a
Image:C:\Users\adity\OneDrive\Desktop\Headphones\19.jpg -> Annotation:C:/Users/a
Image:C:\Users\adity\OneDrive\Desktop\Headphones\20.jpg -> Annotation:C:/Users/a
```

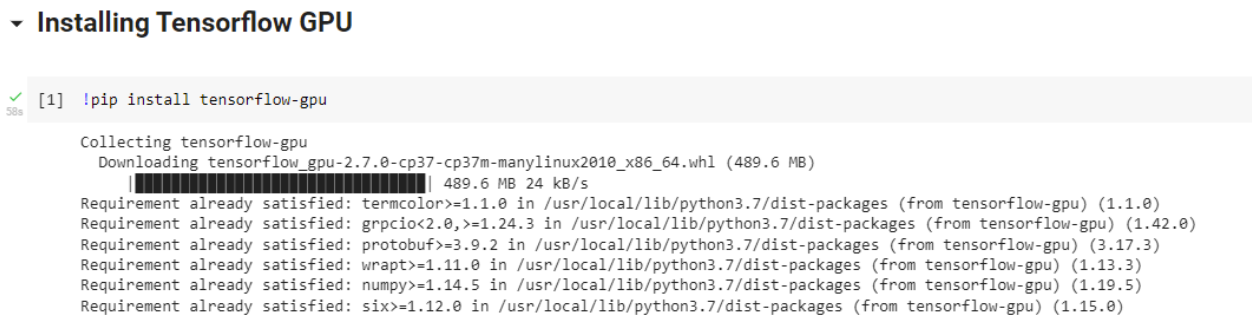
**2. Custom Object Detection Implementation:**

Now that our dataset is ready, we need to implement Yolov3 using darknet framework on it. I have used TensorFlow’s official document to implement this in practice [7]. The link to the API Tutorial guide is - https://tensorflow-object-detection-api-tutorial.readthedocs.io/en/latest/

**3. Step by step implementation of custom Object Detection in python:**

**a. Installing Tensorflow’s GPU:**

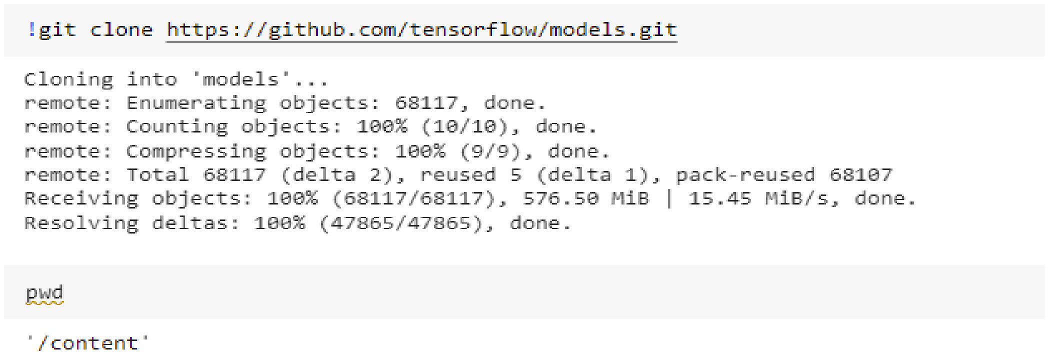
We have used Tensorflow’s Coco – dataset to create custom labels as a reference to train our model.Hence, GPU installation is necessary.



▾ Installing Tensorflow GPU

```
[1] !pip install tensorflow-gpu

Collecting tensorflow-gpu
  Downloading tensorflow_gpu-2.7.0-cp37-cp37m-manylinux2010_x86_64.whl (489.6 MB)
    |████████████████████████████████| 489.6 MB 24 kB/s
Requirement already satisfied: termcolor>=1.1.0 in /usr/local/lib/python3.7/dist-packages (from tensorflow-gpu) (1.1.0)
Requirement already satisfied: grpcio<2.0,>=1.24.3 in /usr/local/lib/python3.7/dist-packages (from tensorflow-gpu) (1.42.0)
Requirement already satisfied: protobuf>=3.9.2 in /usr/local/lib/python3.7/dist-packages (from tensorflow-gpu) (3.17.3)
Requirement already satisfied: wrapt>=1.11.0 in /usr/local/lib/python3.7/dist-packages (from tensorflow-gpu) (1.13.3)
Requirement already satisfied: numpy>=1.14.5 in /usr/local/lib/python3.7/dist-packages (from tensorflow-gpu) (1.19.5)
Requirement already satisfied: six>=1.12.0 in /usr/local/lib/python3.7/dist-packages (from tensorflow-gpu) (1.15.0)
```

**b. Cloning TFOD 2.0 Github Repository**

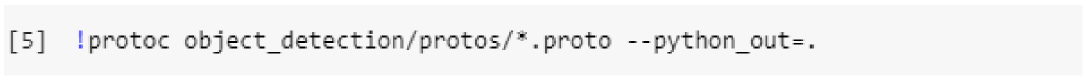


```
!git clone https://github.com/tensorflow/models.git

Cloning into 'models'...
remote: Enumerating objects: 68117, done.
remote: Counting objects: 100% (10/10), done.
remote: Compressing objects: 100% (9/9), done.
remote: Total 68117 (delta 2), reused 5 (delta 1), pack-reused 68107
Receiving objects: 100% (68117/68117), 576.50 MiB | 15.45 MiB/s, done.
Resolving deltas: 100% (47865/47865), done.
```

```
pwd

'/content'
```

**c. Protobufs to configure model**:

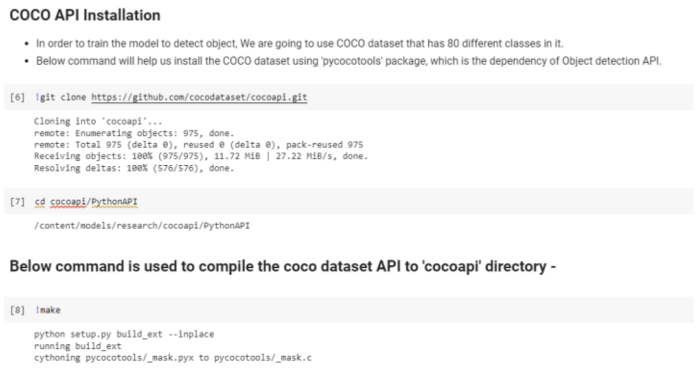
TensorFlow Object Detection API uses **Protobufs to configure model** and training parameters. Before the framework can be used, the Protobufs libraries must be downloaded and compiled.



```
[5]  !protoc object_detection/protos/*.proto --python_out=.
```

**d. COCO API Installation:**

The pycocotools package is listed as a dependency of the Object Detection. Hence, it becomes mandatory to install this API otherwise we might face failure in execution later on.



**COCO API Installation**

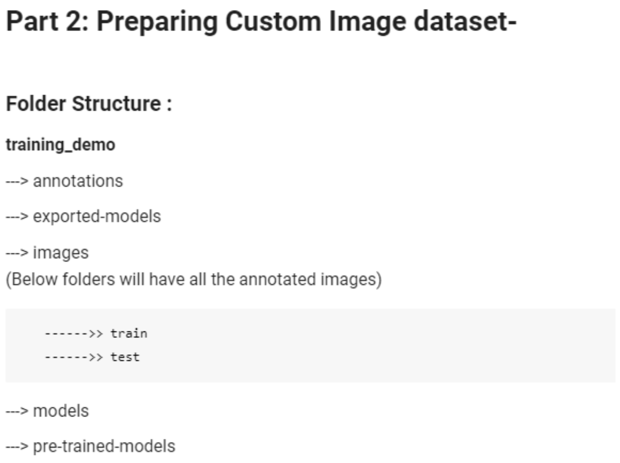
- In order to train the model to detect object, We are going to use COCO dataset that has 80 different classes in it.
- Below command will help us install the COCO dataset using 'pycocotools' package, which is the dependency of Object detection API.

```
[6]  !git clone https://github.com/cocodataset/cocoapi.git

     Cloning into 'cocoapi'...
     remote: Enumerating objects: 975, done.
     remote: Total 975 (delta 0), reused 0 (delta 0), pack-reused 975
     Receiving objects: 100% (975/975), 11.72 MiB | 27.22 MiB/s, done.
     Resolving deltas: 100% (576/576), done.

[7]  cd cocoapi/PythonAPI

     /content/models/research/cocoapi/PythonAPI
```

**Below command is used to compile the coco dataset API to 'cocoapi' directory -**

```
[8]  !make

     python setup.py build_ext --inplace
     running build_ext
     cythoning pycocotools/_mask.pyx to pycocotools/_mask.c
```

**e. Folder Structure for Custom Image Dataset:**



**Part 2: Preparing Custom Image dataset-**

**Folder Structure :**

**training_demo**

---> annotations

---> exported-models

---> images
(Below folders will have all the annotated images)

```
    ------>> train
    ------>> test
```

---> models

---> pre-trained-models

**f. Configuring the training model:**

First, we need to first create a folder in 'models' directory and  name it as 'my_ssd_resnet101_v1_fpn' and upload this 'pipeline.config' file from 'pre-trained-models' folder in that folder.

**Challenges Faced:**

1. Due to hardware constraints, the label creation process faced hurdles and the model kept running until the GPU session timed out.
2. Along with this, higher time required to train a model by hyperparameter tuning in order to improve accuracy of the labels we created.

**Conclusion**:

We found that the yolo model used in this is successful to identify the classes labeled in the COCO dataset with accuracy of objects as mentioned below –

Person 1 – 96.23
Person 2 – 89.31
Bottle – 66.39
Cell Phone – 78.43
Apple – 81.47

**Future Scope:**

- Applications of object detection using live surveillance is the most trending topic in computer vision because of the advent in technologies like YOLOv3 and Darknet.
- This project can be extended further for a specific category of people such as for – Pregnant Women, Old-age people and children on road. Once the object is detected, people driving vehicles in that locality should get a warning on their devices to be careful while driving in order to avoid accidents.

**References:**

1. YOLO Official Website – **https://pjreddie.com/darknet/yolo/**
2. Official Yolo v4 paper - https://arxiv.org/abs/2004.10934
3. https://dev.to/afrozchakure/all-you-need-to-know-about-yolo-v3-you-only-look-once-e4m
4. https://pjreddie.com/darknet/yolo/
5. https://github.com/pjreddie/darknet
6. https://tensorflow-object-detection-api-tutorial.readthedocs.io/en/latest/
7. https://machinelearningmastery.com/object-recognition-with-deep-learning/
8. https://www.cv-foundation.org/openaccess/content_cvpr_2016/papers/Redmon_You_Only_Look_CVPR_2016_paper.pdf
9. https://www.pyimagesearch.com/2020/02/03/how-to-use-opencvs-dnn-module-with-nvidia-gpus-cuda-and-cudnn/
10. https://github.com/theAIGuysCode/YOLOv4-Cloud-Tutorial
11. https://medium.com/analytics-vidhya/train-a-custom-yolov4-object-detector-using-google-colab-61a659d4868