Running a YOLOv4 Object Detector with Darknet in the Cloud! (GPU ENABLED)

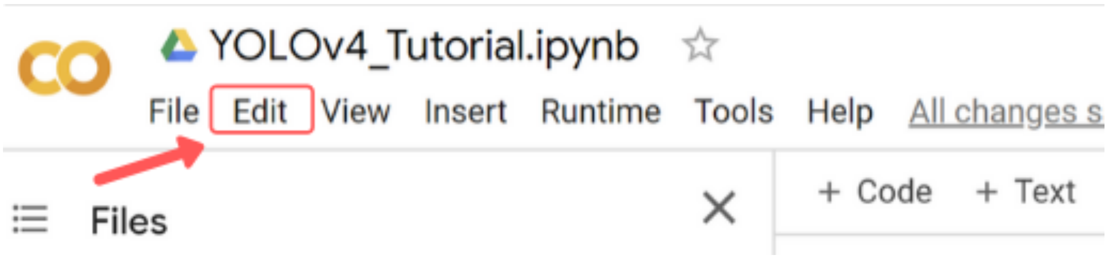
This tutorial will help you build YOLOv4 easily in the cloud with GPU enabled so that you can run object detections in milliseconds!

Step 1: Enabling GPU within your notebook

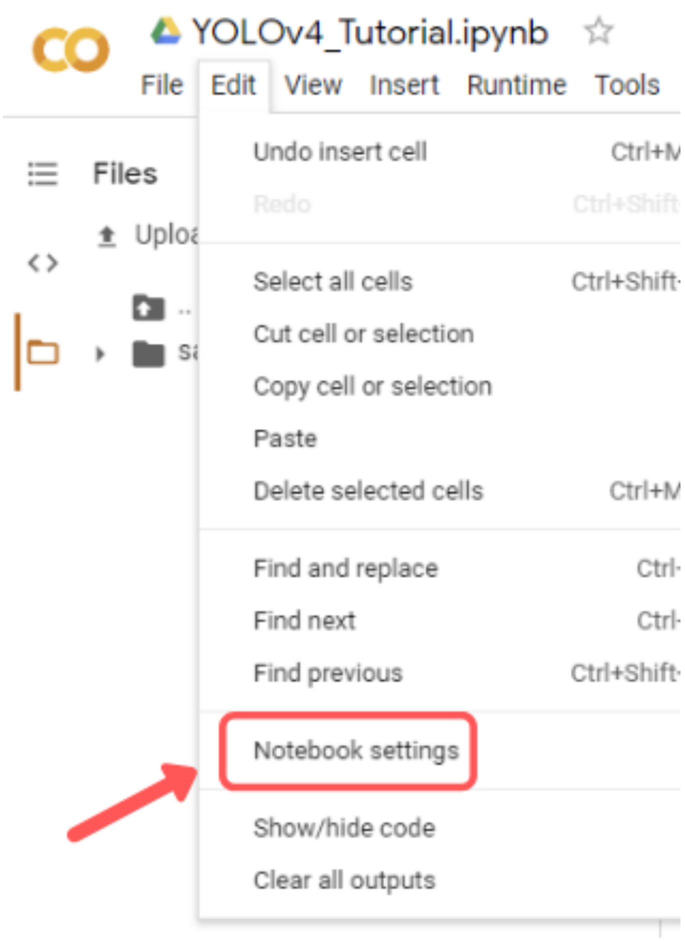
You will want to enable GPU acceleration within your Colab notebook so that your YOLOv4 system will be able to process detections over 100 times faster than CPU.

Steps:

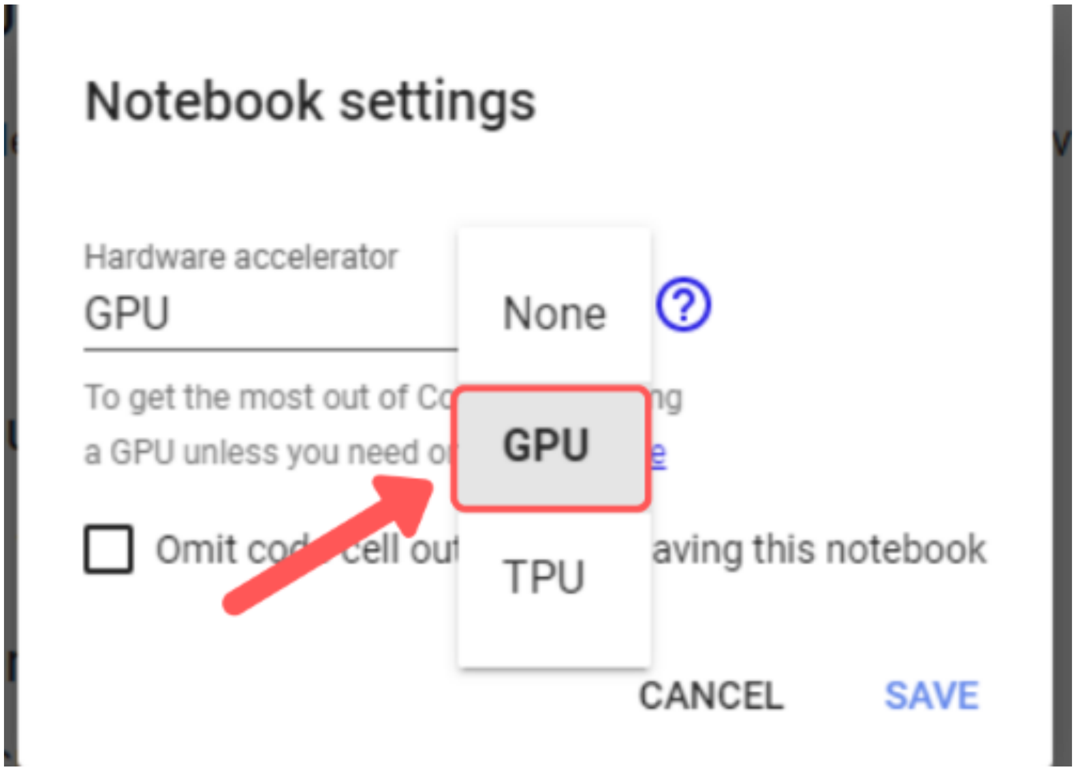
i) Click **Edit** at top left of your notebook



ii) Click **Notebook Settings** within dropdown



iii) Under 'Hardware Accelerator' select **GPU** and then hit **Save**



Your notebook should now have GPU enabled!

Step 2: Cloning and Building Darknet

The following cells will clone darknet from AlexeyAB's famous repository, adjust the Makefile to enable OPENCV and GPU for darknet and then build darknet.

Do not worry about any warnings when you run the '!make' cell!

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

```
Cloning into 'darknet'...
remote: Enumerating objects: 15873, done.
remote: Counting objects: 100% (10/10), done.
remote: Compressing objects: 100% (10/10), done.
remote: Total 15873 (delta 3), reused 0 (delta 0), pack-reused 15863 (from 3)
Receiving objects: 100% (15873/15873), 14.47 MiB | 6.16 MiB/s, done.
Resolving deltas: 100% (10686/10686), 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
!/usr/local/cuda/bin/nvcc --version
```

```
nvcc: NVIDIA (R) Cuda compiler driver
Copyright (c) 2005-2024 NVIDIA Corporation
Built on Thu_Jun__6_02:18:23_PDT_2024
Cuda compilation tools, release 12.5, V12.5.82
Build cuda_12.5.r12.5/compiler.34385749_0
```

```
# make darknet (builds darknet so that you can then use the darknet executable fi
!make
```

```
In file included from /usr/local/cuda/include/cuda_runtime.h:95,
                 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:5668:60: note: expected ‘void **’
```

```
 5668 | extern __host__ cudaError_t CUDARTAPI cudaHostAlloc(void **pHost, si
      |                                                      ~~~~~~~^~~~~
./src/gaussian_yolo_layer.c:77:38: warning: passing argument 1 of ‘cudaHostA
   77 |     if (cudaSuccess == cudaHostAlloc(&l.delta, batch*l.outputs * siz
      |                                      ^~~~~~~~
      |                                      |
      |                                      float **
In file included from /usr/local/cuda/include/cuda_runtime.h:95,
                 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:5668:60: note: expected ‘void **’
 5668 | extern __host__ cudaError_t CUDARTAPI cudaHostAlloc(void **pHost, si
      |                                                      ~~~~~~~^~~~~
./src/gaussian_yolo_layer.c: In function ‘resize_gaussian_yolo_layer’:
./src/gaussian_yolo_layer.c:109:42: warning: passing argument 1 of ‘cudaHost
  109 |         if (cudaSuccess != cudaHostAlloc(&l->output, l->batch*l->out
      |                                          ^~~~~~~~~~
      |                                          |
      |                                          float **
In file included from /usr/local/cuda/include/cuda_runtime.h:95,
                 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:5668:60: note: expected ‘void **’
 5668 | extern __host__ cudaError_t CUDARTAPI cudaHostAlloc(void **pHost, si
      |                                                      ~~~~~~~^~~~~
./src/gaussian_yolo_layer.c:118:42: warning: passing argument 1 of ‘cudaHost
  118 |         if (cudaSuccess != cudaHostAlloc(&l->delta, l->batch*l->outp
      |                                          ^~~~~~~~~
      |                                          |
      |                                          float **
In file included from /usr/local/cuda/include/cuda_runtime.h:95,
                 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:5668:60: note: expected ‘void **’
 5668 | extern __host__ cudaError_t CUDARTAPI cudaHostAlloc(void **pHost, si
      |                                                      ~~~~~~~^~~~~
gcc -Iinclude/ -I3rdparty/stb/include -DOPENCV `pkg-config --cflags opencv4
gcc -Iinclude/ -I3rdparty/stb/include -DOPENCV `pkg-config --cflags opencv4
gcc -Iinclude/ -I3rdparty/stb/include -DOPENCV `pkg-config --cflags opencv4
gcc -Iinclude/ -I3rdparty/stb/include -DOPENCV `pkg-config --cflags opencv4
gcc -Iinclude/ -I3rdparty/stb/include -DOPENCV `pkg-config --cflags opencv4
nvcc -gencode arch=compute_50,code=[sm_50,compute_50] -gencode arch=compute_
nvcc -gencode arch=compute_50,code=[sm_50,compute_50] -gencode arch=compute_
nvcc -gencode arch=compute_50,code=[sm_50,compute_50] -gencode arch=compute_
nvcc -gencode arch=compute_50,code=[sm_50,compute_50] -gencode arch=compute_
nvcc -gencode arch=compute_50,code=[sm_50,compute_50] -gencode arch=compute_
nvcc -gencode arch=compute_50,code=[sm_50,compute_50] -gencode arch=compute_
nvcc -gencode arch=compute_50,code=[sm_50,compute_50] -gencode arch=compute_
nvcc -gencode arch=compute_50,code=[sm_50,compute_50] -gencode arch=compute_
nvcc -gencode arch=compute_50,code=[sm_50,compute_50] -gencode arch=compute_
```

Step 3: Download pre-trained YOLOv4 weights

YOLOv4 has been trained already on the coco dataset which has 80 classes that it can predict. We will grab these pretrained weights so that we can run YOLOv4 on these pretrained classes and get detections.

```
!wget https://github.com/AlexeyAB/darknet/releases/download/darknet_yolo_v3_optim
```

```
--2025-03-08 01:34:12--  https://github.com/AlexeyAB/darknet/releases/downloa
Resolving github.com (github.com)... 140.82.112.3
Connecting to github.com (github.com)|140.82.112.3|:443... connected.
HTTP request sent, awaiting response... 302 Found
Location: https://objects.githubusercontent.com/github-production-release-asse
--2025-03-08 01:34:12--  https://objects.githubusercontent.com/github-product
Resolving objects.githubusercontent.com (objects.githubusercontent.com)... 18
Connecting to objects.githubusercontent.com (objects.githubusercontent.com)|1
HTTP request sent, awaiting response... 200 OK
Length: 257717640 (246M) [application/octet-stream]
Saving to: ‘yolov4.weights’

yolov4.weights      100%[===================>] 245.78M  98.1MB/s    in 2.5s

2025-03-08 01:34:15 (98.1 MB/s) - ‘yolov4.weights’ saved [257717640/257717640
```

Step 4: Define Helper Functions

These three functions are helper functions that will allow you to show the image in your Colab Notebook after running your detections, as well as upload and download images to and from your Cloud VM.

```
# define helper functions
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

  fig = plt.gcf()
  fig.set_size_inches(18, 10)
  plt.axis("off")
  plt.imshow(cv2.cvtColor(resized_image, cv2.COLOR_BGR2RGB))
  plt.show()

# use this to upload files
def upload():
  from google.colab import files
  uploaded = files.upload()
  for name, data in uploaded.items():
    with open(name, 'wb') as f:
      f.write(data)
      print ('saved file', name)

# use this to download a file
def download(path):
  from google.colab import files
  files.download(path)
```

Step 5: Run Your Detections with Darknet and YOLOv4!

Darknet is now built and ready to run detections using YOLOv4 in the cloud! You can find out which sorts of classes the pre-trained YOLOv4 weights can detect by clicking here. COCO CLASSES

The object detector can be run using the following command

```
!./darknet detector test <path to .data file> <path to config> <path to weights> <path to
```

Darknet comes with a few images already installed in the darknet/data/ folder.

Note: After running detections OpenCV can't open the image instantly in the cloud so we must run:

```
imShow('predictions.jpg')
```

This will output the image with the detections shown. The most recent detections are always saved to 'predictions.jpg'

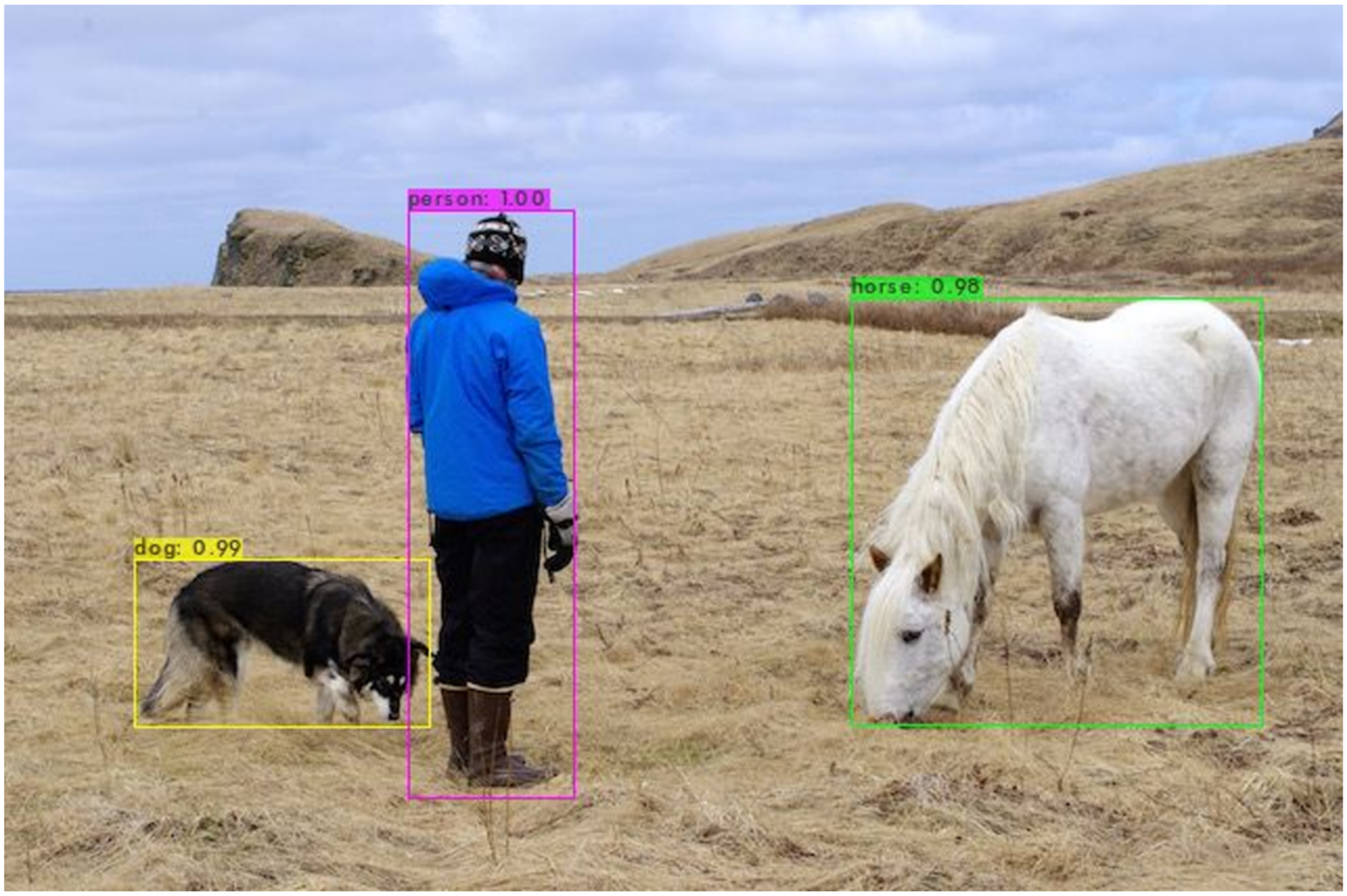
Try out the examples below for yourself!

```
# run darknet detection on test images
!./darknet detector test cfg/coco.data cfg/yolov4.cfg yolov4.weights data/person.
```

```
 122 conv    256       1 x 1/ 1     38 x  38 x 512 ->   38 x  38 x 256 0.37
 123 conv    512       3 x 3/ 1     38 x  38 x 256 ->   38 x  38 x 512 3.40
 124 conv    256       1 x 1/ 1     38 x  38 x 512 ->   38 x  38 x 256 0.37
 125 conv    512       3 x 3/ 1     38 x  38 x 256 ->   38 x  38 x 512 3.40
 126 conv    256       1 x 1/ 1     38 x  38 x 512 ->   38 x  38 x 256 0.37
 127 conv    128       1 x 1/ 1     38 x  38 x 256 ->   38 x  38 x 128 0.09
 128 upsample                 2x    38 x  38 x 128 ->   76 x  76 x 128
 129 route  54                                     ->   76 x  76 x 256
 130 conv    128       1 x 1/ 1     76 x  76 x 256 ->   76 x  76 x 128 0.37
 131 route  130 128                                ->   76 x  76 x 256
 132 conv    128       1 x 1/ 1     76 x  76 x 256 ->   76 x  76 x 128 0.37
 133 conv    256       3 x 3/ 1     76 x  76 x 128 ->   76 x  76 x 256 3.40
 134 conv    128       1 x 1/ 1     76 x  76 x 256 ->   76 x  76 x 128 0.37
 135 conv    256       3 x 3/ 1     76 x  76 x 128 ->   76 x  76 x 256 3.40
 136 conv    128       1 x 1/ 1     76 x  76 x 256 ->   76 x  76 x 128 0.37
 137 conv    256       3 x 3/ 1     76 x  76 x 128 ->   76 x  76 x 256 3.40
 138 conv    255       1 x 1/ 1     76 x  76 x 256 ->   76 x  76 x 255 0.75
 139 yolo
```

```
[yolo] params: iou loss: ciou (4), iou_norm: 0.07, obj_norm: 1.00, cls_norm:
nms_kind: greedynms (1), beta = 0.600000
 140 route  136                                             ->   76 x  76 x 1
 141 conv    256       3 x 3/ 2     76 x  76 x 128 ->   38 x  38 x 256 0.852
 142 route  141 126                                 ->   38 x  38 x 512
 143 conv    256       1 x 1/ 1     38 x  38 x 512 ->   38 x  38 x 256 0.379
 144 conv    512       3 x 3/ 1     38 x  38 x 256 ->   38 x  38 x 512 3.407
 145 conv    256       1 x 1/ 1     38 x  38 x 512 ->   38 x  38 x 256 0.379
 146 conv    512       3 x 3/ 1     38 x  38 x 256 ->   38 x  38 x 512 3.407
 147 conv    256       1 x 1/ 1     38 x  38 x 512 ->   38 x  38 x 256 0.379
 148 conv    512       3 x 3/ 1     38 x  38 x 256 ->   38 x  38 x 512 3.407
 149 conv    255       1 x 1/ 1     38 x  38 x 512 ->   38 x  38 x 255 0.377
 150 yolo
[yolo] params: iou loss: ciou (4), iou_norm: 0.07, obj_norm: 1.00, cls_norm:
nms_kind: greedynms (1), beta = 0.600000
 151 route  147                                             ->   38 x  38 x 2
 152 conv    512       3 x 3/ 2     38 x  38 x 256 ->   19 x  19 x 512 0.852
 153 route  152 116                                 ->   19 x  19 x1024
 154 conv    512       1 x 1/ 1     19 x  19 x1024 ->   19 x  19 x 512 0.379
 155 conv   1024       3 x 3/ 1     19 x  19 x 512 ->   19 x  19 x1024 3.407
 156 conv    512       1 x 1/ 1     19 x  19 x1024 ->   19 x  19 x 512 0.379
 157 conv   1024       3 x 3/ 1     19 x  19 x 512 ->   19 x  19 x1024 3.407
 158 conv    512       1 x 1/ 1     19 x  19 x1024 ->   19 x  19 x 512 0.379
 159 conv   1024       3 x 3/ 1     19 x  19 x 512 ->   19 x  19 x1024 3.407
 160 conv    255       1 x 1/ 1     19 x  19 x1024 ->   19 x  19 x 255 0.189
 161 yolo
[yolo] params: iou loss: ciou (4), iou_norm: 0.07, obj_norm: 1.00, cls_norm:
nms_kind: greedynms (1), beta = 0.600000
Total BFLOPS 128.459
avg_outputs = 1068395
 Allocate additional workspace_size = 52.44 MB
Loading weights from yolov4.weights...
 seen 64, trained: 32032 K-images (500 Kilo-batches_64)
Done! Loaded 162 layers from weights-file
 Detection layer: 139 - type = 28
 Detection layer: 150 - type = 28
 Detection layer: 161 - type = 28
data/person.jpg: Predicted in 234.291000 milli-seconds.
dog: 99%
person: 100%
```

```
# show image using our helper function
imShow('predictions.jpg')
```



```
# This stops 'Run all' at this cell by causing an error
assert False
```

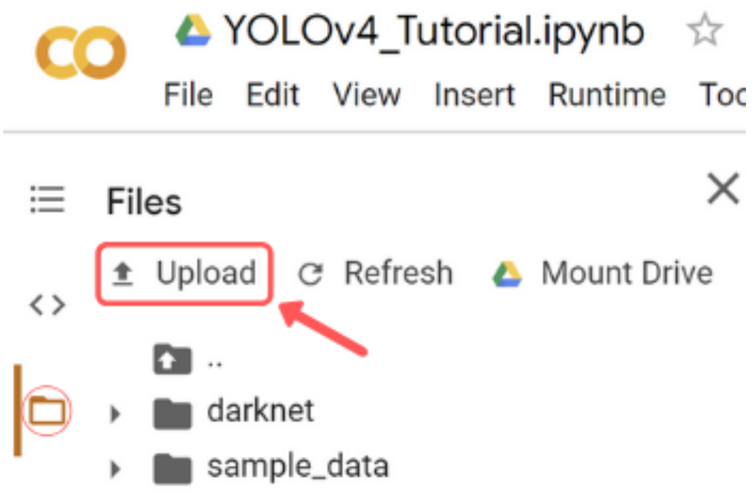
Step 6: Uploading Local or Google Drive Files to Use

You may want to run detections on more than just the images within the darknet/data/ folder. This step will show you how to upload local or Google Drive files to the cloud VM and run detections on them!

Method 1: Local Files

To upload local files just use our helper function by running 'upload()' as seen below. Click **Choose Files** and select the image from your local machine that you want to upload to the cloud VM.

If this function doesn't work for you then click the **Upload** button in the File Explorer on the left



side of your notebook.

The image should save to the root directory of the cloud VM so that you can access it from the darknet command by running.

```
!./darknet detector test cfg/coco.data cfg/yolov4.cfg yolov4.weights ../<your image name>
```

```
# try out the upload helper function! (I uploaded an image called highway.jpg, up
%cd ..
upload()
%cd darknet
```

```
# run darknet with YOLOv4 on your personal image! (note yours will not be called
!./darknet detector test cfg/coco.data cfg/yolov4.cfg yolov4.weights ../highway.j
imShow('predictions.jpg')
```

Method 2: Google Drive

Images can also be uploaded from your Google Drive and easily have YOLOv4 detections run on them.

You will want to run the below cell to mount your google drive into the cloud VM so that you can access its contents. It is that easy!

NOTE: We will be creating a symbolic link between '/content/gdrive/My\ Drive/' and '/mydrive.

This means we are just creating a shortcut '/mydrive' to map to the contents within the folder '/content/gdrive/My\ Drive/'.

The reason for this is that sometime having the space in 'My Drive' folder path can cause issues when running certain commands. This symbolic link will stop this from happening!

Now you can run YOLOv4 with images from Google Drive using the darknet command:

```
!./darknet detector test cfg/coco.data cfg/yolov4.cfg yolov4.weights /mydrive/<path to im
```

I recommend saving images within a folder called 'images' at the root level of your Google Drive.

```
%cd ..
from google.colab import drive
drive.mount('/content/gdrive')
```

```
# this creates a symbolic link so that now the path /content/gdrive/My\ Drive/ is
!ln -s /content/gdrive/My\ Drive/ /mydrive
!ls /mydrive
```

```
# cd back into the darknet folder to run detections
%cd darknet
```

```
# run detections on image within your Google Drive!
!./darknet detector test cfg/coco.data cfg/yolov4.cfg yolov4.weights /mydrive/ima
imShow('predictions.jpg')
```

Download Files to Local Machine or Google Drive from Cloud VM

You can also easily download images from your cloud VM to save to your local machine or Google Drive.

Method 1: Local Machine

You can do it easily by using our helper function 'download()' or by right clicking the image in the File Explorer on the left side of your notebook and hitting **Download**. Files will be saved to your *Downloads* folder.

This is useful if you want to download the **'predictions.jpg'** images that the object detector outputs.

Method 2: Google Drive

A simple copy command can copy over any file to your Google Drive as it is already mounted. (you must run the mount command above if you have not already)

```
!cp <file to download> <destination to save file>
```

See example of each below!

```
# LOCAL MACHINE DOWNLOAD
# if you get an error first run then run it again and it should work
download('predictions.jpg')
```

```
# GOOGLE DRIVE DOWNLOAD
# note that I can change what the image name is saved as (I am saving it as detec
!cp predictions.jpg /mydrive/images/detection1.jpg
```

Step 7: Running YOLOv4 on Video in the Cloud!

You can also run YOLOv4 object detections on video in your Google Colab VM. Follow the cells below to see how to run videos from your local machine or from your Google Drive!

Local Machine Video

Here is how to upload video from local machine, run detector and then download video showing detections.

```
# upload the video of your choosing! (Feel free to use the same video I do, it is
upload()
```

```
!./darknet detector demo cfg/coco.data cfg/yolov4.cfg yolov4.weights -dont_show t
```

```
# download the video with detections shown
download('results.avi')
```

Google Drive Video

Here is how to run detector on video stored in Google Drive and save video straight to Google drive as well!

Note: You will have to change the paths to where your video is stored within your Google Drive and to where you want the resulting video stored. I have a videos folder in the home directory of my Google Drive.

```
!./darknet detector demo cfg/coco.data cfg/yolov4.cfg yolov4.weights -dont_show /
```

Step 8: Customize YOLOv4 with the different command line flags.

Darknet and YOLOv4 have a lot of command line flags you can add to your '!./darknet detector ...' to allow it to be customizeable and flexible.

I will show a few examples of these flags that you can take advantage of! Feel free to mix and match them together to customize your detections in any way you want.

Threshold Flag

There is a flag '-thresh' you can use to add a threshold for confidences on the detections. Only detections with a confidence level above the threshold you set will be returned.

In the example below we run darknet with YOLOv4 without a threshold on the test image dog.jpg. The image returns four detections, the lowest confidence being on the pottedplant with 33%.

If we add the '-thresh 0.5' flag this will only output three detections as now pottedplant falls below the threshold and is ignored.

Check it out below!

```
# this is ran without the threshold flag set
!./darknet detector test cfg/coco.data cfg/yolov4.cfg yolov4.weights data/dog.jpg
imShow('predictions.jpg')
```

```
# same detections but ran with the threshold flag set to 0.5 (pottedplant is no l
!./darknet detector test cfg/coco.data cfg/yolov4.cfg yolov4.weights data/dog.jpg
imShow('predictions.jpg')
```

Output Bounding Box Coordinates

You can output bounding box coordinates for each detection with the flag '-ext_output'. This external outputs flag will give you a few extra details about each detection within an image.

Check it out below!

```
# darknet run with external output flag to print bounding box coordinates
!./darknet detector test cfg/coco.data cfg/yolov4.cfg yolov4.weights data/person.
imShow('predictions.jpg')
```

Don't Show Image

You can add the flag '-dont_show' to not have the image outputted after running darknet. This doesn't really affect anything when running in Colab as the image is unable to output properly straight from darknet anyways. However, by adding the -dont_show flag you will get rid of the following warning from showing.

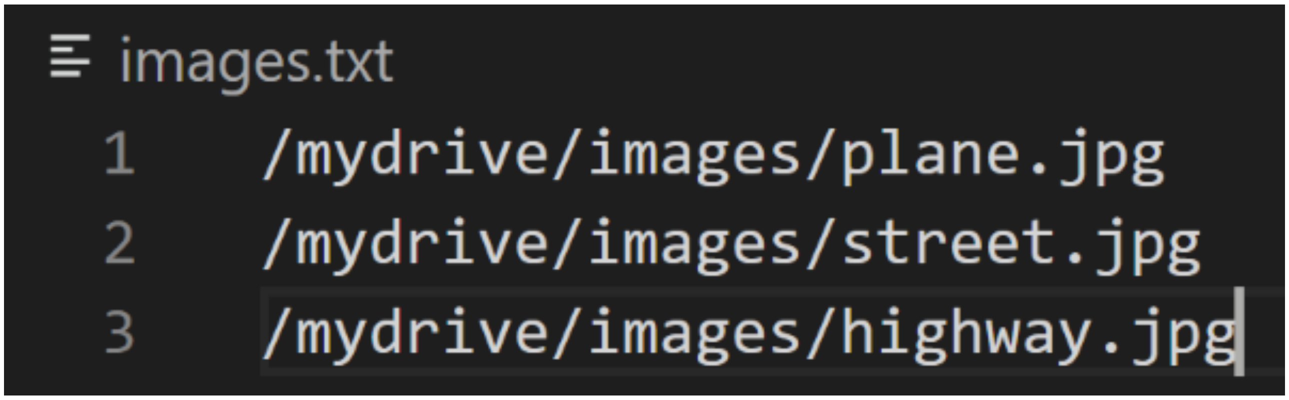
```
Unable to init server: Could not connect: Connection refused

(predictions:1850): Gtk-WARNING **: 17:01:00.687: cannot open display:
```

This is an important flag to have when running darknet with YOLOv4 on video as it will suppress having the video shown.

```
# running darknet with dont show flag set (no longer get warnings)
!./darknet detector test cfg/coco.data cfg/yolov4.cfg yolov4.weights data/person.
```

Step 9: Multiple Images at Once

YOLOv4 object detections can be run on multiple images at once. This is done through having a text file which has the paths to several images that you want to have the detector run on.



images.txt

```
/mydrive/images/plane.jpg
/mydrive/images/street.jpg
/mydrive/images/highway.jpg
```

The .txt file should be in this format. One path to an image per line.

This file is stored to my Google Drive root directory and holds the path to three images within my Google Drive images folder.

Save Results to .JSON File

Here is an example of saving the multiple image detections to a .JSON file.

```
!./darknet detector test cfg/coco.data cfg/yolov4.cfg yolov4.weights -ext_output
```

```
download('result.json')
```

Saving Results to a .txt File

You can also save the results of running multiple images to a text file.

```
!./darknet detector test cfg/coco.data cfg/yolov4.cfg yolov4.weights -dont_show -
```

```
download('result.txt')
```

How to Train Your Own YOLOv4 Custom Object Detector!

Now comes the time to create your own custom YOLOv4 object detector to recognize any classes/objects you want!

This requires a couple tricks and tips so make sure to follow along closely with the rest of the tutorial.

In order to create a custom YOLOv4 detector we will need the following:

- Labeled Custom Dataset
- Custom .cfg file
- obj.data and obj.names files
- train.txt file (test.txt is optional here as well)

Step 1: Gathering and Labeling a Custom Dataset

In order to create a custom object detector you need a good dataset of images and labels so that the detector can be efficiently trained to detect objects.

This can be done in two ways. through or through using Google images or creating your own dataset and using an annotation tool to manually draw labels. **(I recommend the first way!)**

Method 1: Using Google's Open Images Dataset (RECOMMENDED)

This method is the method I recommend as you can gather thousands of images and auto-generate their labels within minutes! Gathering a dataset from Google's Open Images Dataset and using OIDv4 toolkit to generate labels is easy and time efficient. The dataset contains labeled images for over 600 classes! Explore the Dataset Here!

Watch one of my previous youtube videos on how to do use this approach to create your own custom dataset!

Creating a Custom YOLOv3 Dataset (Video) **(images and labels are same format for YOLOv4 so video still works)**

Here is a link to my github repository for the OIDv4 toolkit! Github Repo

For this tutorial I will be creating a license plate object detector using data from Google's Open Images Dataset.

Training Dataset

I ran the following commands within the toolkit to create my custom training dataset. I chose to use 1500 images within my dataset but the general rule is the more images the better for higher accuracy of your model.

```
python main.py downloader --classes 'Vehicle registration plate' --type_csv train --limit
```

Note: If you are training a detector for multiple classes add '--multiclasses 1' to the command above to have all your training images saved together in one folder.

This creates a folder with 1500 images of 'Vehicle registration plates' (license plates) for us to train our custom object detector with.

Validation Dataset

Note: Having a validation dataset is optional but highly recommended.

We want to have a validation dataset in order to properly test our custom object detector after it is trained. We can take advantage of the OIDv4_Toolkit as it has seperate images for validation to make sure we have different images and not ones that are in our training dataset.

We can run the same commands above but for validation. I recommend having a validation dataset size that is roughly 20-30% of your training dataset size. Since I chose 1500 images for my training data this means I will have a validation size of 300 images. (20% of 1500 = 300)

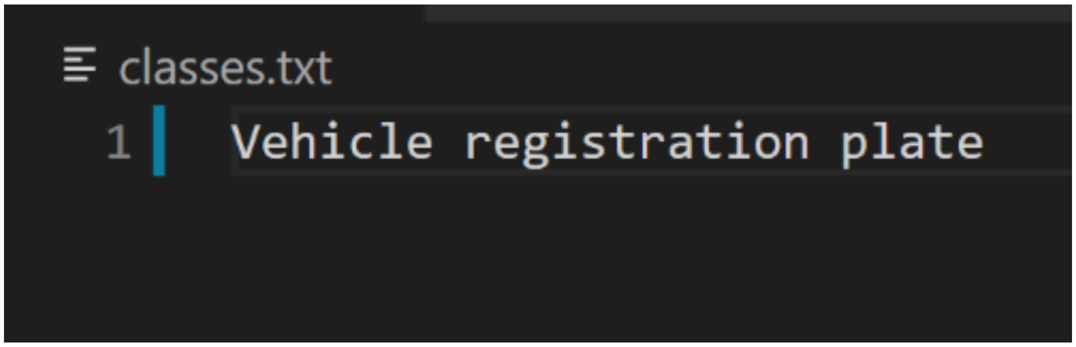
```
python main.py downloader --classes 'Vehicle registration plate' --type_csv validation --
```

Now you have a proper validation dataset to test your custom object detector after training!

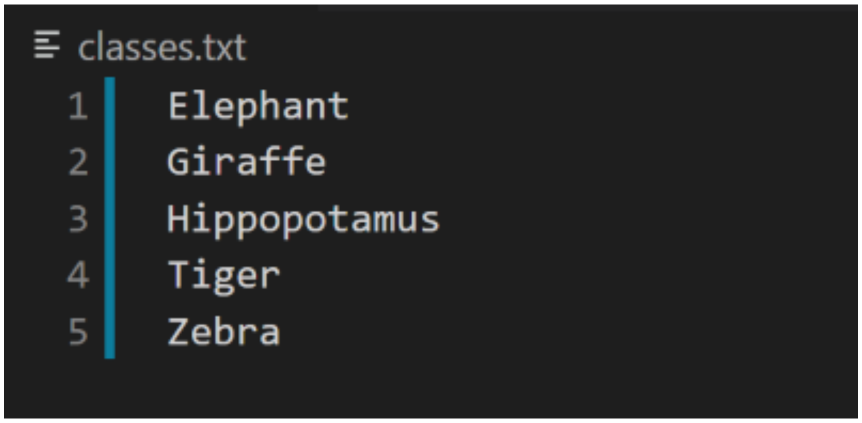
Converting Labels to YOLOv4 Format

The labels that we get from the toolkit are not in the proper YOLOv4 format. Luckily, with a few simple commands we can convert them to the proper format.

Within the root OIDv4_ToolKit folder open the file classes.txt and edit it to have the classes you just downloaded, one per line.


```
classes.txt
Vehicle registration plate
```

Example of classes.txt for multiclass object detector:


```
classes.txt
Elephant
Giraffe
Hippopotamus
Tiger
Zebra
```

Now run this command to convert the labels generated by OIDv4_Toolkit into YOLOv4 labels (this should convert both train and validation dataset labels):

```
python convert_annotations.py
```

This converts all labels to YOLOv4 format which can now be used by darknet to properly train our custom object detector.

Remove the old 'Label' folder in the train and validation folders which contains the non YOLOv4 formatted labels by running the following: (your file path will have a different name for 'Vehicle registration plate'... depending on which classes you downloaded.

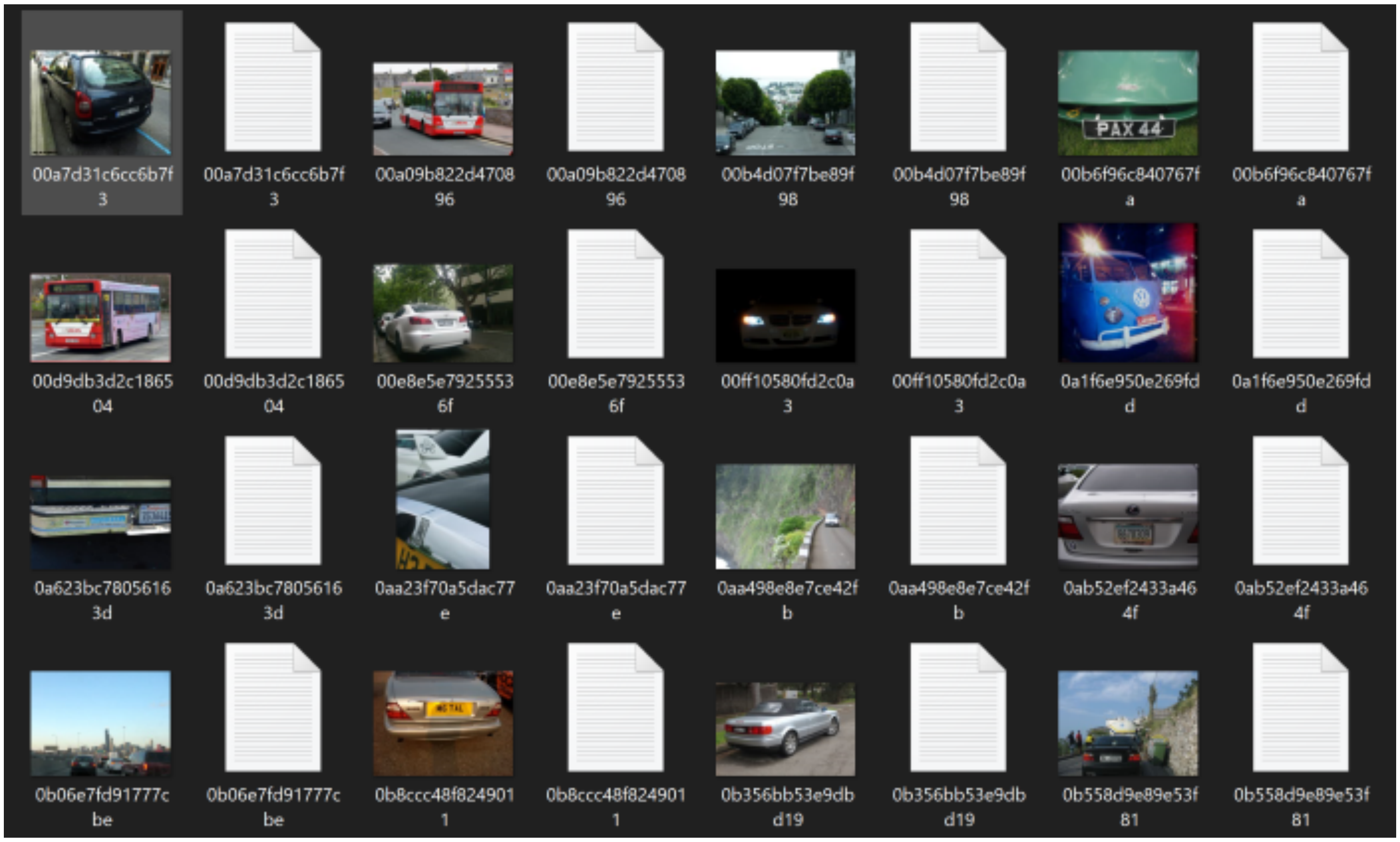
```
rm -r OID/Dataset/train/'Vehicle registration plate'/Label/
```

And finally delete the old labels for validation as well.

```
rm -r OID/Dataset/validation/'Vehicle registration plate'/Label/
```

If this command doesn't work on your machine then just go to the folder with 'Label' and right click and hit Delete to manually delete it.

The folders with all your images and annotations should now look like this. Each image should have a text file with the same name beside (or below) it.



You now have your Custom YOLOv4 Training and Validation Datasets with proper labels!

Method 2: Manually Labeling Images with Annotation Tool

If you can't find the proper images or classes within Google's Open Images Dataset then you will have to use an annotation tool to manually draw your labels which can be a tiresome process.

I have created a previous video where I walk through how to mass download images from Google Images along with how to use LabelImg, an annotation tool, in order to create a custom dataset for YOLOv4. Hit link below to learn how.

Create Labels and Annotations for Custom YOLOv3 Dataset (Video) **(same format for YOLOv4)**

After following the tutorial video you should be able to have a folder with images and text files as your training dataset as well as one for your validation dataset like the one above.

You now have your Custom YOLOv4 Datasets!

Step 2: Moving Your Custom Datasets Into Your Cloud VM

So now that you have your datasets properly formatted to be used for training and validation, we need to move them into this cloud VM so that when it comes the time we can actually train and validate our model.

I recommend renaming the trainin dataset folder with your images and text files on your local machine to be called '**obj**' and then creating a .zip folder of the 'obj' folder. Then I recommend uploading the zip to your Google Drive. So you should now have obj.zip someplace in your Google drive.

Do the same with your validation dataset but name it '**test**'. So you should now have **test.zip** also uploaded to your Google Drive.

This will **greatly reduce** the time it takes to transfer our dataset into our cloud VM.

Now we can copy in the zips and unzip them in your cloud VM.

```
# this is where my datasets are stored within my Google Drive (I created a yolov4
!ls /mydrive/yolov4
```

```
# copy over both datasets into the root directory of the Colab VM (comment out te
!cp /mydrive/yolov4/obj.zip ../
!cp /mydrive/yolov4/test.zip ../
```

```
# unzip the datasets and their contents so that they are now in /darknet/data/ fo
!unzip ../obj.zip -d data/
!unzip ../test.zip -d data/
```

Step 3: Configuring Files for Training

This step involves properly configuring your custom .cfg, obj.data, obj.names, train.txt and test.txt files.

It is important to configure all these files with extreme caution as typos or small errors can cause major problems with your custom training.

i) Cfg File

Copy over the yolov4.cfg to your Google Drive by running the cell below. This will allow us to edit it in a text editor.

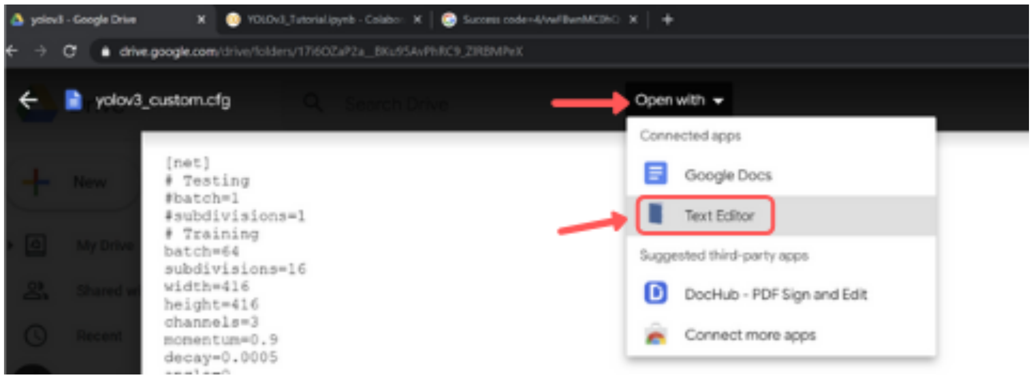
```
# download cfg to google drive and change its name
!cp cfg/yolov4-custom.cfg /mydrive/yolov4/yolov4-obj.cfg
```

```
# to download to local machine (change its name to yolov4-obj.cfg once you downlo
download('cfg/yolov4-custom.cfg')
```

Now you need to edit the .cfg to fit your needs based on your object detector. Open it up in a code or text editor to do so.

If you downloaded cfg to google drive you can use the built in **Text Editor** by going to your google drive and double clicking on yolov4-obj.cfg and then clicking on the **Open with** drop down and selectin **Text Editor**.

(Image from previous tutorial so don't mind different file name)


yolov3 - Google Drive
YOLOv3_Tutorial.ipynb - Colabo
drive.google.com/drive/folders/17i6OZaP2a_BKu95AvPhRC9_ZIRBMPeX
yolov3_custom.cfg
Open with
Connected apps
Google Docs
Text Editor
Suggested third-party apps
DocHub - PDF Sign and Edit
Connect more apps
New
My Drive
Recent

```
[net]
# Testing
#batch=1
#subdivisions=1
# Training
batch=64
subdivisions=16
width=416
height=416
channels=3
momentum=0.9
decay=0.0005
```

I recommend having **batch = 64** and **subdivisions = 16** for ultimate results. If you run into any issues then up subdivisions to 32.

Make the rest of the changes to the cfg based on how many classes you are training your detector on.

Note: I set my **max_batches = 6000**, **steps = 4800, 5400**, I changed the **classes = 1** in the three YOLO layers and **filters = 18** in the three convolutional layers before the YOLO layers.

How to Configure Your Variables:

width = 416

height = 416 **(these can be any multiple of 32, 416 is standard, you can sometimes improve results by making value larger like 608 but will slow down training)**

max_batches = (# of classes) * 2000 **(but no less than 6000 so if you are training for 1, 2, or 3 classes it will be 6000, however detector for 5 classes would have max_batches=10000)**

steps = (80% of max_batches), (90% of max_batches) **(so if your max_batches = 10000, then steps = 8000, 9000)**

filters = (# of classes + 5) * 3 **(so if you are training for one class then your filters = 18, but if you are training for 4 classes then your filters = 27)**

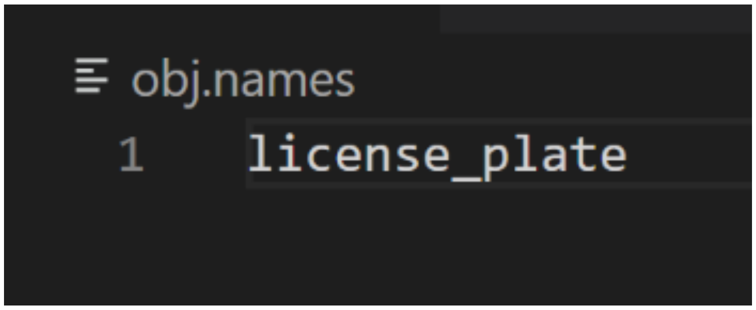
Optional: If you run into memory issues or find the training taking a super long time. In each of the three yolo layers in the cfg, change one line from random = 1 to **random = 0** to speed up training but slightly reduce accuracy of model. Will also help save memory if you run into any memory issues.

```
# upload the custom .cfg back to cloud VM from Google Drive
!cp /mydrive/yolov4/yolov4-obj.cfg ./cfg
```

ii) obj.names and obj.data

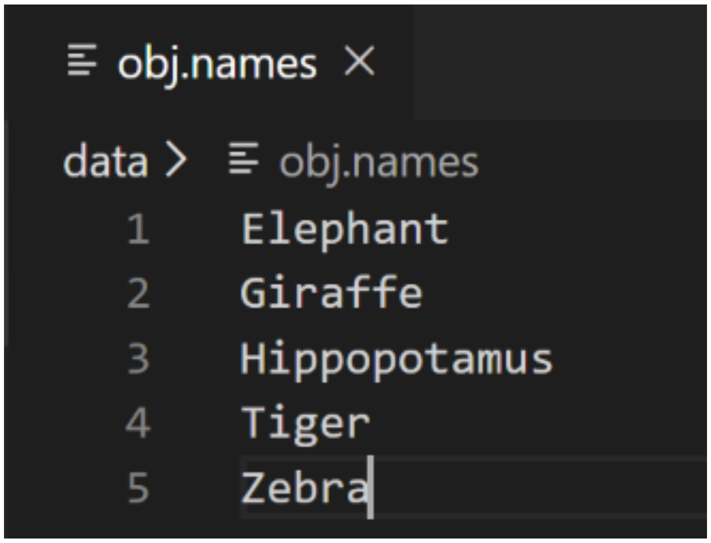
Create a new file within a code or text editor called **obj.names** where you will have one class name per line in the same order as your classes.txt from the dataset generation step.

NOTE: You do not want to have spaces in your class name. For this reason I changed "Vehicle Registration Plate" to **license_plate**.



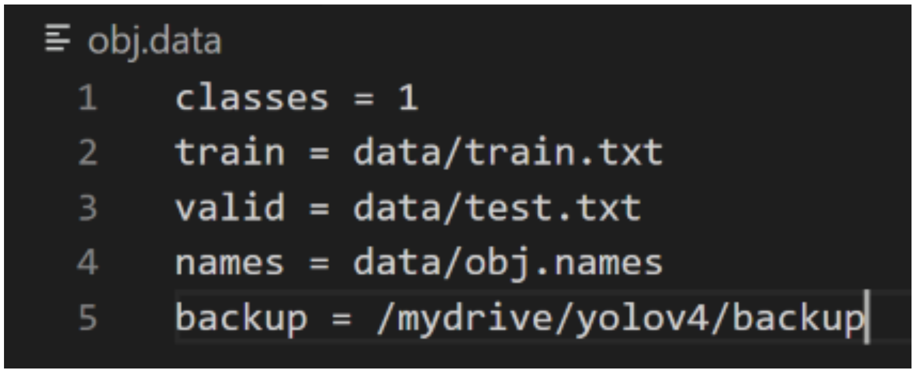
```
license_plate
```

Example for multiclass obj.names file:



```
Elephant
Giraffe
Hippopotamus
Tiger
Zebra
```

You will also create a **obj.data** file and fill it in like this (change your number of classes accordingly, as well as your backup location)



```
classes = 1
train = data/train.txt
valid = data/test.txt
names = data/obj.names
backup = /mydrive/yolov4/backup
```

This backup path is where we will save the weights to of our model throughout training. Create a backup folder in your google drive and put its correct path in this file.

```
# upload the obj.names and obj.data files to cloud VM from Google Drive
!cp /mydrive/yolov4/obj.names ./data
!cp /mydrive/yolov4/obj.data  ./data
```

iii) Generating train.txt and test.txt

The last configuration files needed before we can begin to train our custom detector are the train.txt and test.txt files which hold the relative paths to all our training images and valdidation images.

Luckily I have created scripts that eaily generate these two files withe proper paths to all images.

The scripts can be accessed from the Github Repo

Just download the two files to your local machine and upload them to your Google Drive so we can use them in the Colab Notebook.

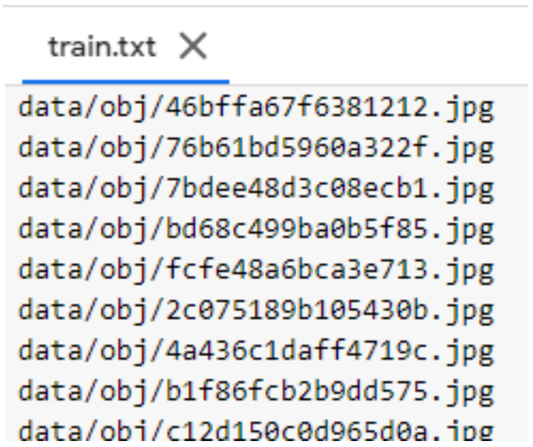
```
# upload the generate_train.py and generate_test.py script to cloud VM from Googl
!cp /mydrive/yolov4/generate_train.py ./
!cp /mydrive/yolov4/generate_test.py ./
```

Now simply run both scripts to do the work for you of generating the two txt files.

```
!python generate_train.py
!python generate_test.py
```

```
# verify that the newly generated train.txt and test.txt can be seen in our darkn
!ls data/
```

If you are uncertain whether generating the files worked and want to double check that everything went as planned, double click on **train.txt** on the left side File Explorer and it should look like this.



```
data/obj/46bffa67f6381212.jpg
data/obj/76b61bd5960a322f.jpg
data/obj/7bdee48d3c08ecb1.jpg
data/obj/bd68c499ba0b5f85.jpg
data/obj/fcfe48a6bca3e713.jpg
data/obj/2c075189b105430b.jpg
data/obj/4a436c1daff4719c.jpg
data/obj/b1f86fcb2b9dd575.jpg
data/obj/c12d150c0d965d0a.jpg
```

It will contain one line for each training image path.

Step 4: Download pre-trained weights for the convolutional layers.

This step downloads the weights for the convolutional layers of the YOLOv4 network. By using these weights it helps your custom object detector to be way more accurate and not have to train as long. You don't have to use these weights but trust me it will help your modle converge and be accurate way faster. USE IT!

```
!wget https://github.com/AlexeyAB/darknet/releases/download/darknet_yolo_v3_optim
```

Step 5: Train Your Custom Object Detector!

The time has finally come! You have made it to the moment of truth! You are now ready to train your custom YOLOv4 object detector on whatever crazy classes you have decided on. So run the following command. (-dont_show flag stops chart from popping up since Colab Notebook can't open images on the spot, -map flag overlays mean average precision on chart to see how accuracy of your model is, only add map flag if you have a validation dataset)

```
 !./darknet detector train <path to obj.data> <path to custom config> yolov4.conv.137 -don
```

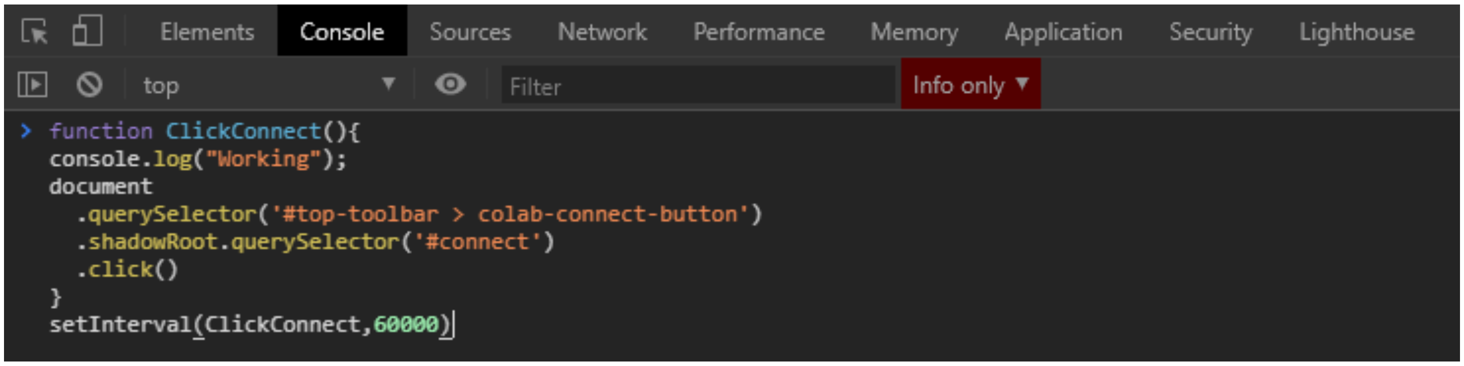
TIP: This training could take several hours depending on how many iterations you chose in the .cfg file. You will want to let this run as you sleep or go to work for the day, etc. However, Colab Cloud Service kicks you off it's VMs if you are idle for too long (30-90 mins).

To avoid this hold (CTRL + SHIFT + i) at the same time to open up the inspector view on your browser.

Paste the following code into your console window and hit **Enter**

```
function ClickConnect(){
console.log("Working");
document
  .querySelector('#top-toolbar > colab-connect-button')
  .shadowRoot.querySelector('#connect')
  .click()
}
setInterval(ClickConnect,60000)
```

Looks like this, it will click the screen every 10 minutes so that you don't get kicked off for being idle! HACKS!



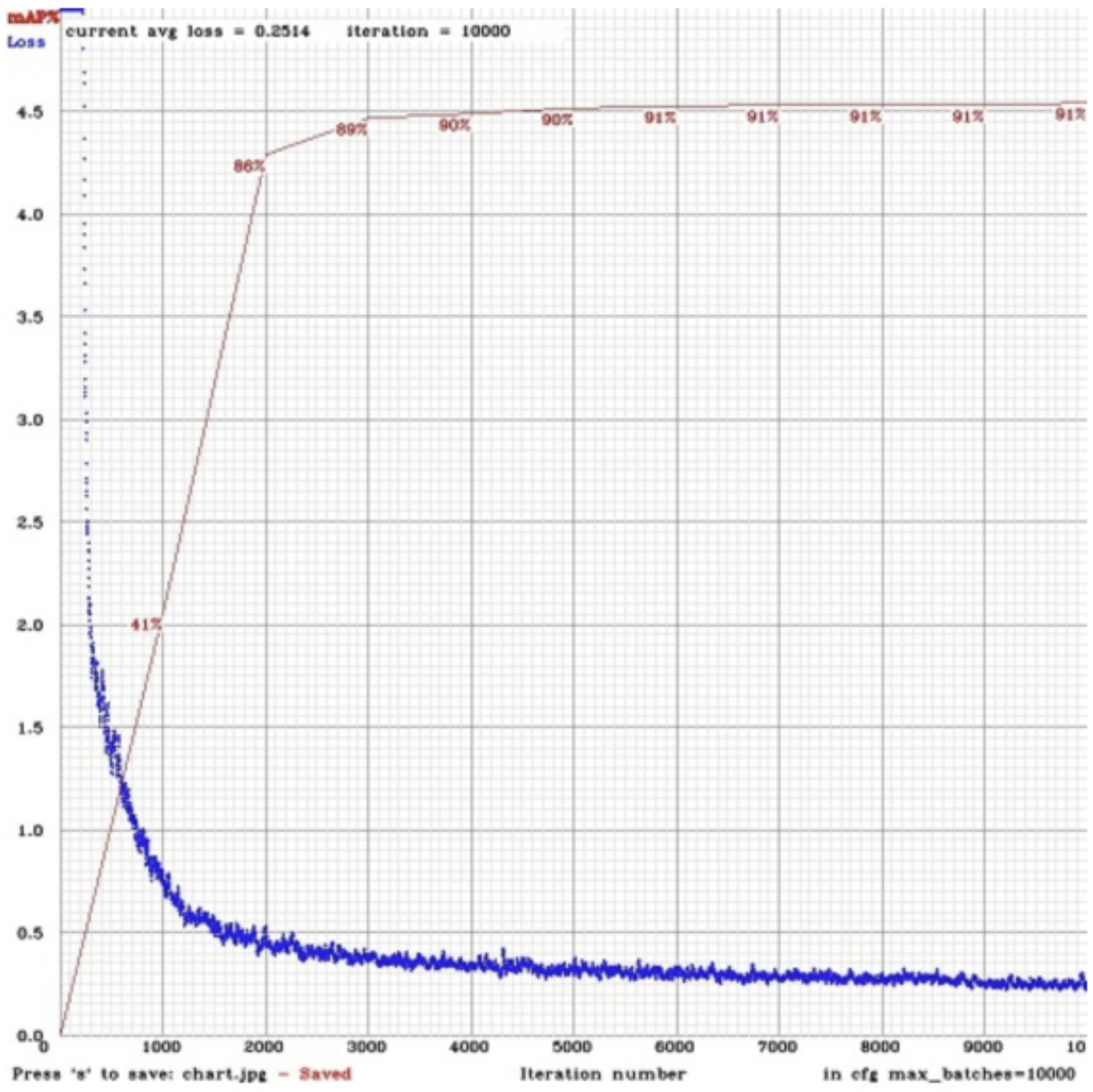
```
> function ClickConnect(){
  console.log("Working");
  document
    .querySelector('#top-toolbar > colab-connect-button')
    .shadowRoot.querySelector('#connect')
    .click()
  }
  setInterval(ClickConnect,60000)
```

```
# train your custom detector! (uncomment %%capture below if you run into memory i
# %%capture
!./darknet detector train data/obj.data cfg/yolov4-obj.cfg yolov4.conv.137 -dont_
```

After training, you can observe a chart of how your model did throughout the training process by running the below command. It shows a chart of your average loss vs. iterations. For your model to be 'accurate' you should aim for a loss under 2.

```
# show chart.png of how custom object detector did with training
imShow('chart.png')
```

Here is what the chart.png should look like after an uninterrupted training! If you stop training or it crashes during training your chart will look like the above one but don't worry you can still check accuracy of your model in the next steps.



TRICK: If for some reason you get an error or your Colab goes idle during training, you have not lost your partially trained model and weights! Every 100 iterations a weights file called **yolov4-obj_last.weights** is saved to **mydrive/yolov4/backup/** folder (wherever your backup folder is). This is why we created this folder in our Google drive and not on the cloud VM. If your runtime crashes and your backup folder was in your cloud VM you would lose your weights and your training progress.

We can kick off training from our last saved weights file so that we don't have to restart! WOOHOO! Just run the following command but with your backup location.

```
!./darknet detector train data/obj.data cfg/yolov4-obj.cfg /mydrive/yolov4/backup/yolov4-
```

```
# kick off training from where it last saved
!./darknet detector train data/obj.data cfg/yolov4-obj.cfg /mydrive/yolov4/backup
```

Step 6: Checking the Mean Average Precision (mAP) of Your Model

If you didn't run the training with the '-map- flag added then you can still find out the mAP of your model after training. Run the following command on any of the saved weights from the training to see the mAP value for that specific weight's file. I would suggest to run it on multiple of the saved weights to compare and find the weights with the highest mAP as that is the most accurate one!

NOTE: If you think your final weights file has overfitted then it is important to run these mAP commands to see if one of the previously saved weights is a more accurate model for your classes.

```
!./darknet detector map data/obj.data cfg/yolov4-obj.cfg /mydrive/yolov4/backup/y
```

Step 7: Run Your Custom Object Detector!!!

You have done it! You now have a custom object detector to make your very own detections. Time to test it out and have some fun!

```
# need to set our custom cfg to test mode
%cd cfg
!sed -i 's/batch=64/batch=1/' yolov4-obj.cfg
!sed -i 's/subdivisions=16/subdivisions=1/' yolov4-obj.cfg
%cd ..
```

```
# run your custom detector with this command (upload an image to your google driv
!./darknet detector test data/obj.data cfg/yolov4-obj.cfg /mydrive/yolov4/backup/
imShow('predictions.jpg')
```

Thanks!

Thanks for following along this tutorial, I hope it worked well for all of you!

If you enjoy YOLOv3 or YOLOv4 check out my other videos and subscribe to my channel so you don't miss out on any other amazing YOLO content!

The AI Guy