

HOW TO RUN THIS CODE FOR “BIRD DEECTION”

Download Python IDLE

- Go to this site: <https://www.python.org/downloads/>
- Download the latest version for Windows 3.11.1

Install Python IDLE

- Simply install the Python 3.11.1

Open Command Prompt

Install all required libraries

- pip3 install numpy
- pip3 install matplotlib
- pip3 install pandas
- pip3 install scikit-learn
- pip3 install cv2
- pip3 install numpy
- pip3 install glob
- pip3 install random

Open Python IDLE

Open the code "`Bird_Detection.py`"

Set the train model "`testing.cfg`"

Set the built model "`training_bird.weights`"

Run the code "`Bird_Detection.py`"

[Next How to build a custom model for Bird Detection](#)

Building an Object Detector

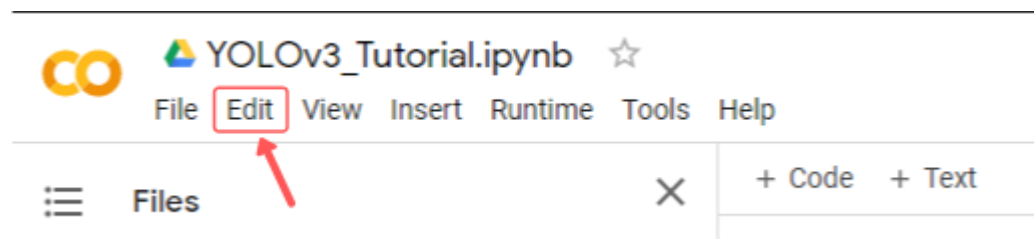
This code will help you build YOLOv3 easily in the cloud with GPU enabled so that you can run real-time object detections as well as train your very own custom object detector!

Step 1: Enabling GPU within your notebook

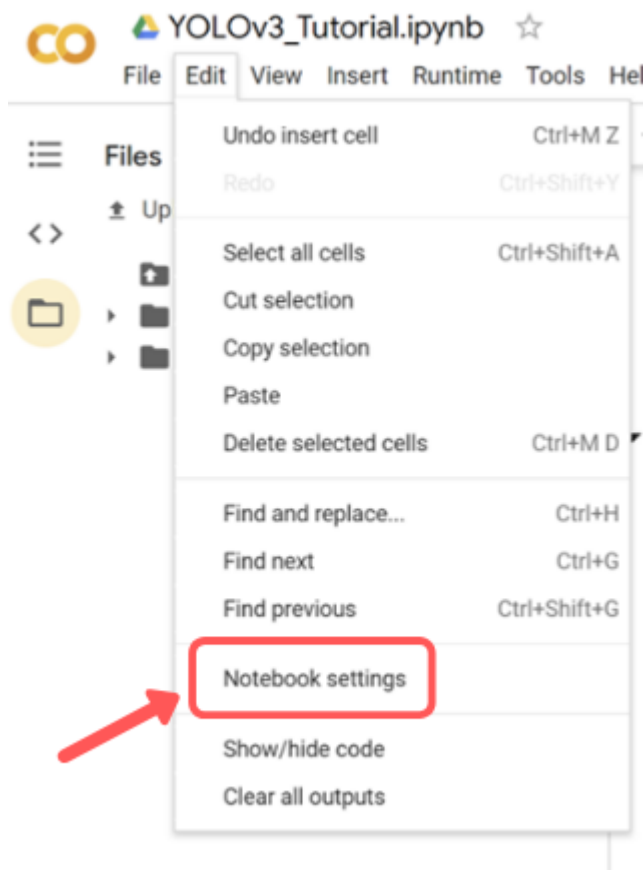
You need to enable GPU acceleration within your Colab notebook so that your YOLOv3 system will be able to process detections over 100 faster than CPU.

Steps:

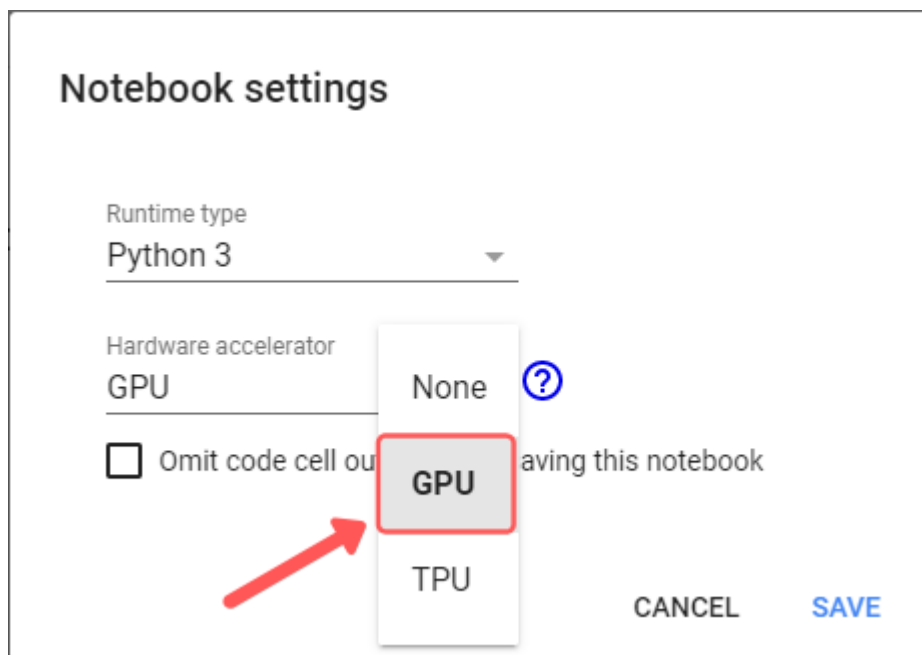
i) Click **Edit** at top left of your notebook



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.

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

# 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

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

# make darknet (build)
!make
```

▼ Step 3: Download pretrained YOLOv3 weights

YOLOv3 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 YOLOv3 on these pretrained classes and get detections.

```
# get yolov3 pretrained coco dataset weights
!wget https://pjreddie.com/media/files/yolov3.weights

# 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_CUBIC)

    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 4: Run Detections with Darknet and YOLOv3!

Darknet is now built and ready to run detections using YOLOv3 in the cloud! You can find out which sorts of classes the pretrained YOLOv3 weights can detect by clicking here. [COCO CLASSES](#)

The object detector can be run using the following command

```
!./darknet detect <path to config> <path to weights> <path to image>
```

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
!./darknet detect cfg/yolov3.cfg yolov3.weights data/bird1.jpg

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



```
# look we can run another detection!  
!./darknet detect cfg/yolov3.cfg yolov3.weights data/bird2.jpg  
imshow('predictions.jpg')
```

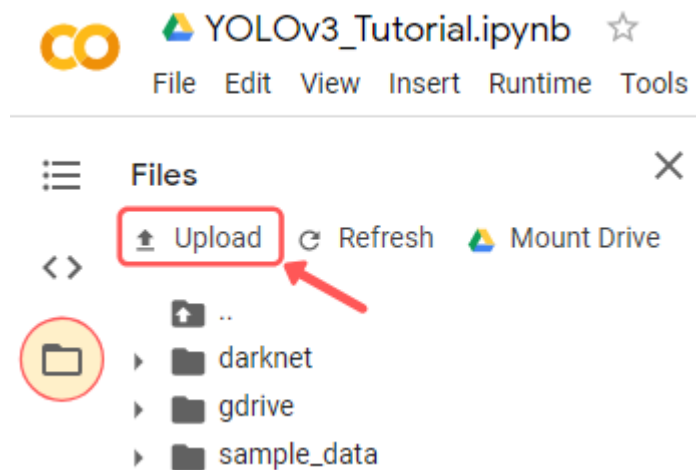
▼ Step 5: Uploading Local or Google Drive Files to Use

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 your root directory so that you can access it from your darknet command by running.

```
!./darknet detect cfg/yolov3.cfg yolov3.weights ../<your image name>
```

```
# upload an image to root directory (I uploaded an image called street.jpg, you can upload ar
%cd ..
upload()
```

```
# make sure you are in the darknet folder to run the detections command!
%cd darknet
!./darknet detect cfg/yolov3.cfg yolov3.weights ../bird3.jpg
imshow('predictions.jpg')
```

▼ Method 2: Google Drive

Images can also be uploaded from your Google Drive and easily have 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 YOLOv3 with images from Google Drive using the darknet command:


```
!./darknet detect cfg/yolov3.cfg yolov3.weights /mydrive/<path to image>
```

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 equal to /n
!ln -s /content/gdrive/My\ Drive/ /mydrive
!ls /mydrive

# run darknet command with google drive image (my image path is /images/plane.jpg)
%cd darknet
!./darknet detect cfg/yolov3.cfg yolov3.weights /mydrive/images/bird4.jpg
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 detection1.jpg)  
!cp predictions.jpg /mydrive/images/detection1.jpg
```

Training a Custom YOLOv3 Object Detector in the Cloud!

WOW! YOU ARE KILLING IT SO FAR!

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 YOLOv3 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

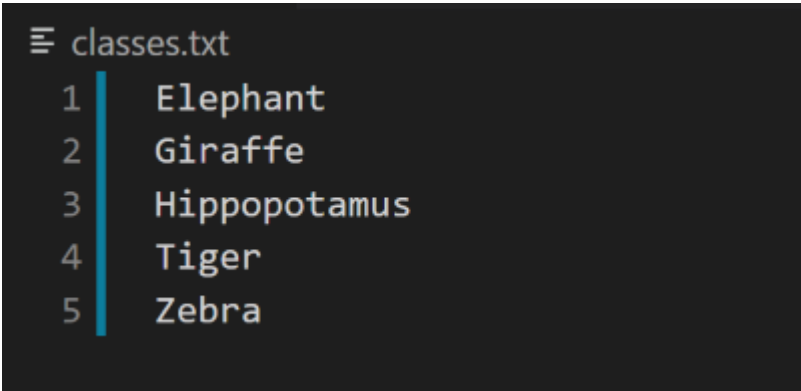
In order to create a custom object detector you need a good dataset of images and labels so that the detector can be efficiently trained to detect objects.

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 OLDv4 toolkit to generate labels is easy and time efficient. The dataset contains labeled images for over 600 classes! [Explore the Dataset Here!](#)

Within the root OIDv4_ToolKit folder open the file classes.txt and edit it to have the classes you just



```
≡ classes.txt
1 Elephant
2 Giraffe
3 Hippopotamus
4 Tiger
5 Zebra
```

downloaded, one per line.

Now convert the image annotations:

```
python convert_annotations.py
```

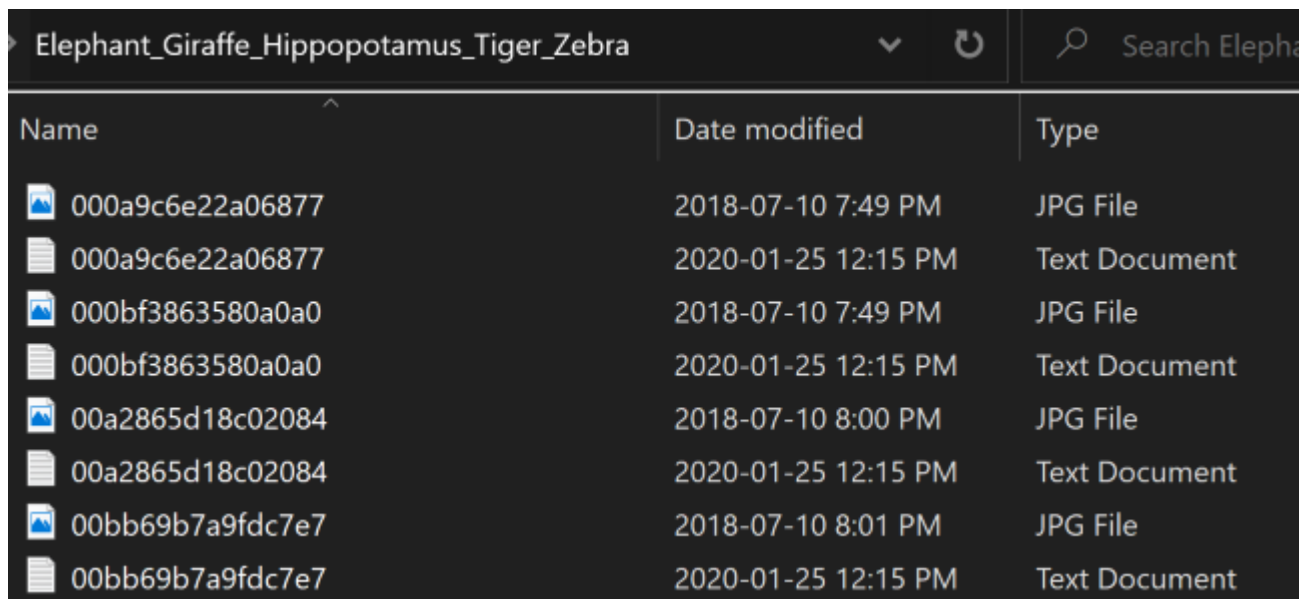
This converts all labels to YOLOv3 format which can now be used by darknet to properly train our custom object detector.









Remove the old 'Label' folder in the OIDv4 toolkit which contains the non YOLOv3 formatted labels by running: (your file path will have a different name for Elephant_Giraffe.. depending on which classes you downloaded).

```
rm -r OID/Dataset/train/Elephant_Giraffe_Hippopotamus_Tiger_Zebra/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 folder with all your images and annotations should now look like this. Each image should have a text file with the same name.



Elephant_Giraffe_Hippopotamus_Tiger_Zebra			Search Elephant_Giraffe_Hippopotamus_Tiger_Zebra
Name	Date modified	Type	
 000a9c6e22a06877	2018-07-10 7:49 PM	JPG File	
 000a9c6e22a06877	2020-01-25 12:15 PM	Text Document	
 000bf3863580a0a0	2018-07-10 7:49 PM	JPG File	
 000bf3863580a0a0	2020-01-25 12:15 PM	Text Document	
 00a2865d18c02084	2018-07-10 8:00 PM	JPG File	
 00a2865d18c02084	2020-01-25 12:15 PM	Text Document	
 00bb69b7a9fdc7e7	2018-07-10 8:01 PM	JPG File	
 00bb69b7a9fdc7e7	2020-01-25 12:15 PM	Text Document	

You have successfully generated a custom YOLOv3 dataset!

Congrats!

▼ Step 2: Moving Your Custom Dataset Into Your Cloud VM

So now that you have your dataset properly formatted to be used for training we need to move it into this cloud VM so that when it comes the time we can actually use it for training.

I recommend renaming the 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.

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

Now we can copy in the zip and unzip it on your cloud VM.

```
# this is where my zip is stored (I created a yolov3 folder where I will get my required files)
!ls /mydrive/yolov3
```

```
# copy the .zip file into the root directory of cloud VM
!cp /mydrive/yolov3/obj.zip ../
```

```
# unzip the zip file and its contents should now be in /darknet/data/obj
!unzip ../obj.zip -d data/obj/
```

▼ Step 3: Configuring Files for Training

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

i) Cfg File

Copy over the yolov3.cfg to edit by running the cell below.

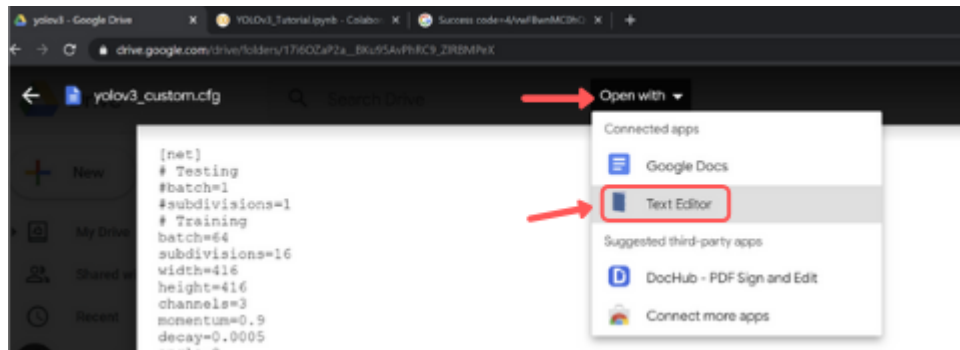
```
# download cfg to google drive and change its name
!cp cfg/yolov3.cfg /mydrive/yolov3/yolov3_custom_bird1.cfg
```

```
# to download to local machine (change its name to yolov3_custom.cfg once you download)
download('cfg/yolov3.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 yolov3_custom.cfg and then clicking on the **Open with** drop down and

selectin **Text Editor**.



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 = 10000**, **steps = 8000, 9000**, I changed the **classes = 5** in the three YOLO layers and **filters = 30** in the three convolutional layers before the YOLO layers.

Optional: 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/yolov3/yolov3_custom_bird1.cfg ./cfg
```

```
# upload the custom .cfg back to cloud VM from local machine (uncomment to use)
#%cd cfg
#upload()
#%cd ..
```

▼ ii) obj.names and obj.data

Create a new file within a code or text editor called **obj.names** and you will make this file exactly the same as your classes.txt in the dataset generation step.

```
≡ obj.names ×  
data > ≡ obj.names  
1    Elephant  
2    Giraffe  
3    Hippopotamus  
4    Tiger  
5    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)

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.

```
≡ obj.data ●  
data > ≡ obj.data  
1    classes = 5  
2    train = data/train.txt  
3    valid = data/test.txt  
4    names = data/obj.names  
5    backup = /mydrive/yolov3/backup/
```

```
# upload the obj.names and obj.data files to cloud VM from Google Drive
```

```
!cp /mydrive/yolov3/obj.names ./data
```

```
!cp /mydrive/yolov3/obj.data ./data
```

```
# upload the obj.names and obj.data files to cloud VM from local machine (uncomment to use)
```

```
##cd data
```

```
#upload()
```

```
##cd ..
```

▼ iii) Generating train.txt

The last configuration file needed before we can begin to train our custom detector is the train.txt file which hold the relative paths to all our training images.

```
# upload the generate_train.py script to cloud VM from Google Drive
```

```
!cp /mydrive/yolov3/generate_train.py ./
```

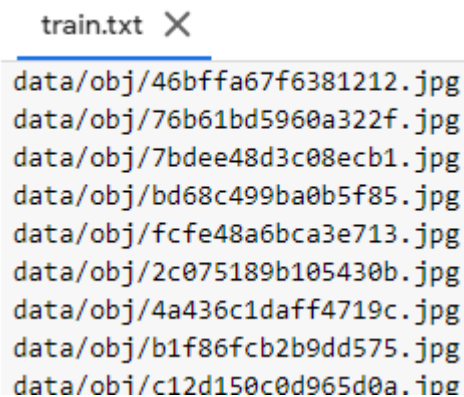
```
# upload the generate_train.py script to cloud VM from local machine (uncomment to use)
#upload()
```

Now we simply run the python script to do all the work for us.

```
!python generate_train.py
```

```
# verify train.txt can be seen in our darknet/data folder
!ls data/
```

If everything went as planned double click on **train.txt** on the left side File Explorer and it should



```
train.txt X
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
```

look like this.

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 YOLOv3 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 model converge and be accurate way faster. USE IT!

```
# upload pretrained convolutional layer weights
!wget http://pjreddie.com/media/files/darknet53.conv.74
```

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 YOLOv3 object detector on whatever crazy classes you have decided on. So run the following command. (dont_show flag stops a chart from popping up since cloud can't open images on the spot)

```
!./darknet detector train <path to obj.data> <path to custom config> darknet53.conv.74 -dont_show
```

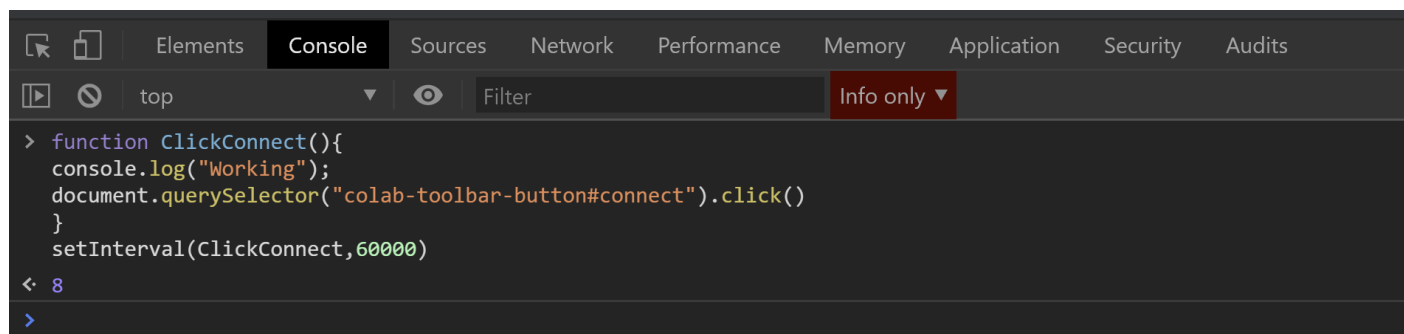
TIP: This training could take several hours depending on how many iterations you chose in the .cfg file. You will want to let this run as you sleep or go to work for the day, etc. However, Colab Cloud Service kicks you off it's VMs if you are idle for too long (30-90 mins).

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("colab-toolbar-button#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!



```
# train your custom detector
```

```
!./darknet detector train data/obj.data cfg/yolov3_custom_bird1.cfg darknet53.conv.74 -dont_s
```

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 would aim for a loss under 2.

```
imshow('chart.png')
```