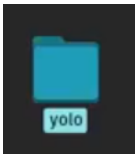
Yolo V3 Implementation in Linux

Open this link: https://pjreddie.com/darknet/yolo/

Create a new folder name called "**Yolo**" in any location.



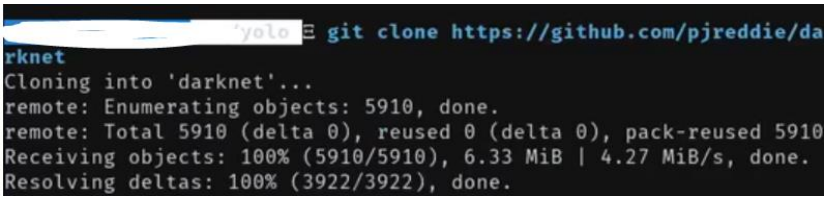
Open the terminal where the Yolo folder is residing.

Clone:

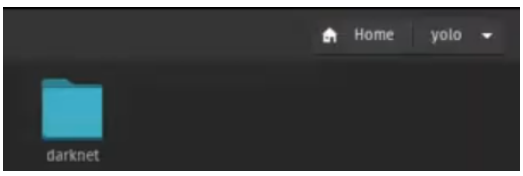
Now I have to clone the contents from official Git Hub repo to our local system Yolo folder.

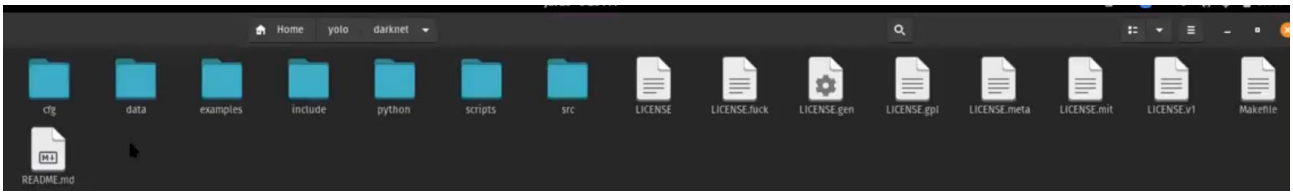
Command: git clone https://github.com/pjreddie/darknet

In case of TFOD we have downloaded the models repo and Here we are downloading the darknet repo.



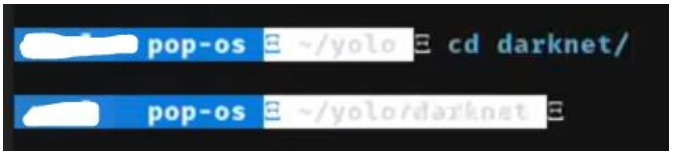
Now the cloning has done and the darknet repo has created



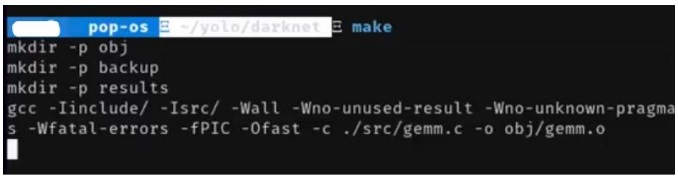


Now go to the terminal and navigate to the folder **cd darknet**

Command: cd darknet

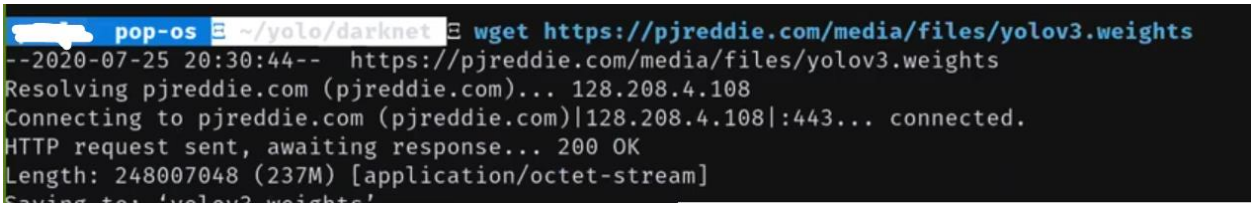


Then open the **make** file.

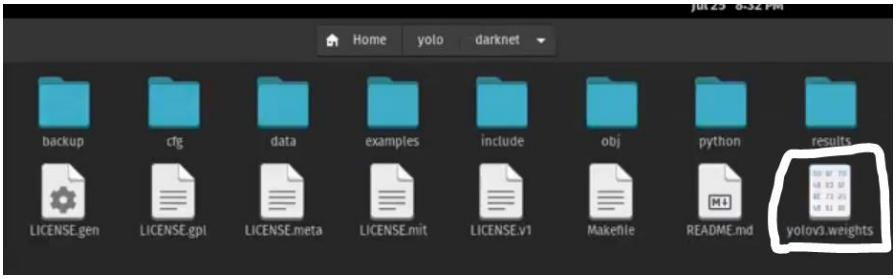


Now we need to **download the pre-trained weights file.**

Command: wget https://pjreddie.com/media/files/yolov3.weights

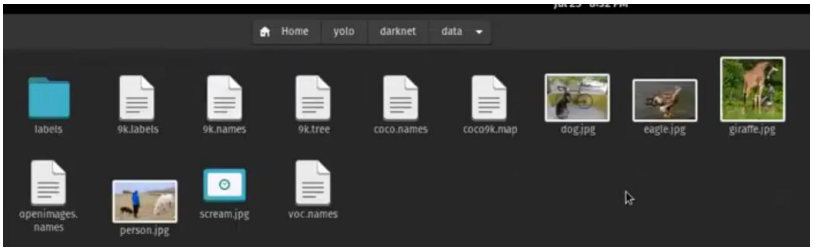


Once the weights are downloaded, then we need to move that file in to our Yolo folder

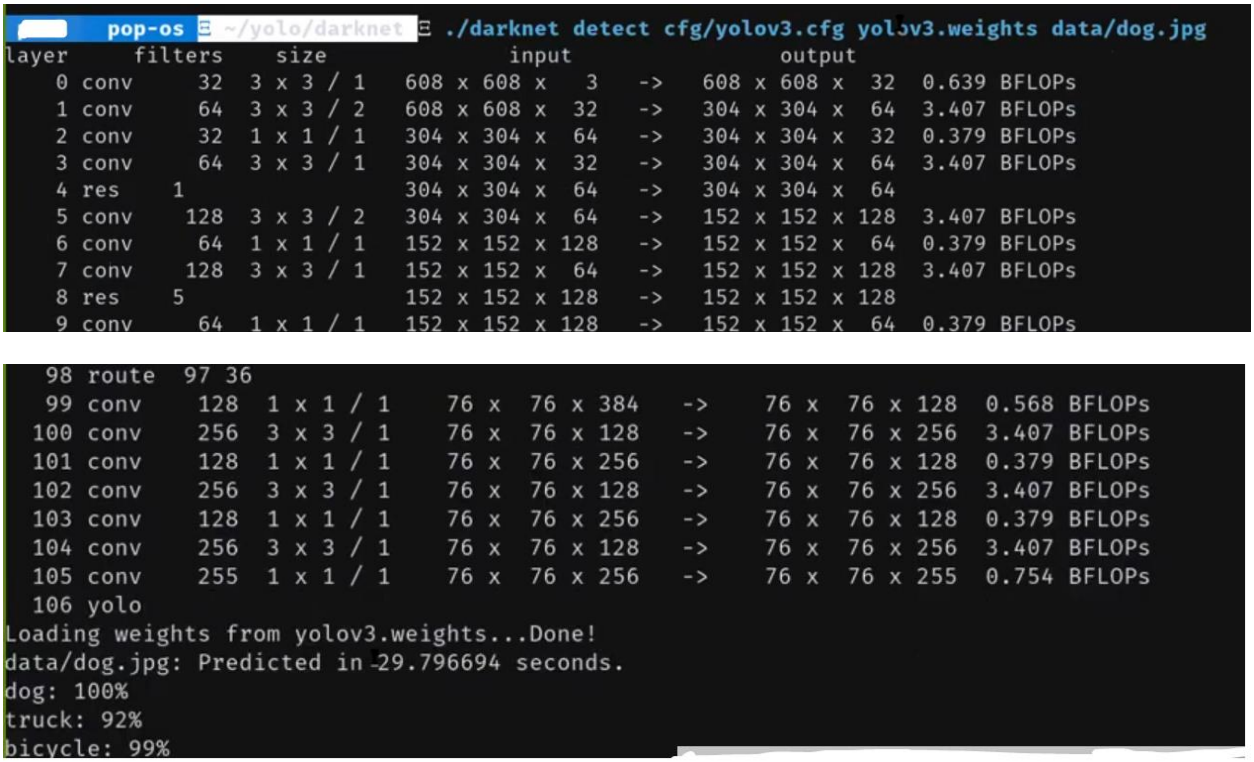


Now I'm going to detect with some image

Under this location, they have provided some sample images for testing purpose.



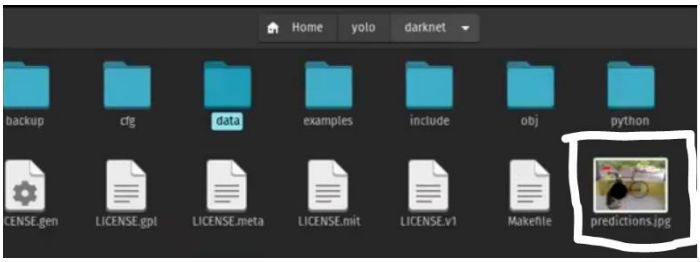
Command: ./darknet detect cfg/yolov3.cfg yolov3.weights data/dog.jpg



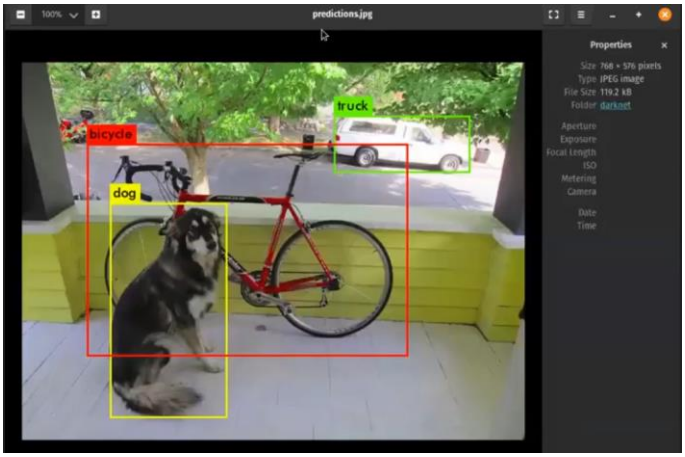
If you noticed over here. For prediction it took almost 30 seconds

Now we are going to view the Image

Yolo folder ---- > darknet ---- > then you see the image under the name of “predictions.jpg”



If you open that file the it looks like below



Even you can test with many sample images

```
pop-os ~/yolo/darknet ./darknet detect cfg/yolov3.cfg yolov3.weights data/giraffe.jpg
  101 conv    128  1 x 1 / 1    76 x  76 x 256   ->    76 x  76 x 128  0.379 BFLOPs
  102 conv    256  3 x 3 / 1    76 x  76 x 128   ->    76 x  76 x 256  3.407 BFLOPs
  103 conv    128  1 x 1 / 1    76 x  76 x 256   ->    76 x  76 x 128  0.379 BFLOPs
  104 conv    256  3 x 3 / 1    76 x  76 x 128   ->    76 x  76 x 256  3.407 BFLOPs
  105 conv    255  1 x 1 / 1    76 x  76 x 256   ->    76 x  76 x 255  0.754 BFLOPs
  106 yolo
Loading weights from yolov3.weights...Done!
data/giraffe.jpg: Predicted in 30.805723 seconds.
giraffe: 98%
zebra: 98%
```

Tiny Yolo:

Download the tiny yolo weights first:

Command: wget https://pjreddie.com/media/files/yolov3-tiny.weights

Once the weights are downloaded then copy the tiny yolo weights into our darknet folder.

Detection:

Command: ./darknet detect cfg/yolov3-tiny.cfg yolov3-tiny.weights data/dog.jpg

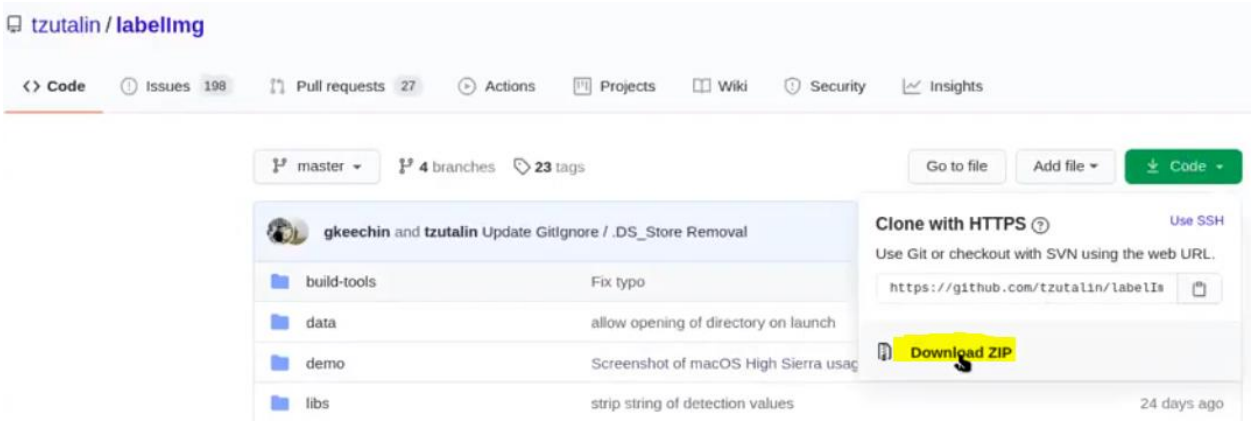
```
pop-os ~/yolo/darknet ./darknet detect cfg/yolov3-tiny.cfg yolov3-tiny.weights data/dog.jpg
```

```
  16 yolo
  17 route  13
  18 conv    128  1 x 1 / 1    13 x  13 x 256   ->    13 x  13 x 128  0.011 BFLOPs
  19 upsample            2x    13 x  13 x 128   ->    26 x  26 x 128
  20 route  19 8
  21 conv    256  3 x 3 / 1    26 x  26 x 384   ->    26 x  26 x 256  1.196 BFLOPs
  22 conv    255  1 x 1 / 1    26 x  26 x 256   ->    26 x  26 x 255  0.088 BFLOPs
  23 yolo
Loading weights from yolov3-tiny.weights...Done!
data/dog.jpg: Predicted in 1.293594 seconds.
dog: 57%
car: 52%
truck: 56%
car: 62%
bicycle: 59%
```

Custom Training:

We need to install the labelImg in our Linux machine

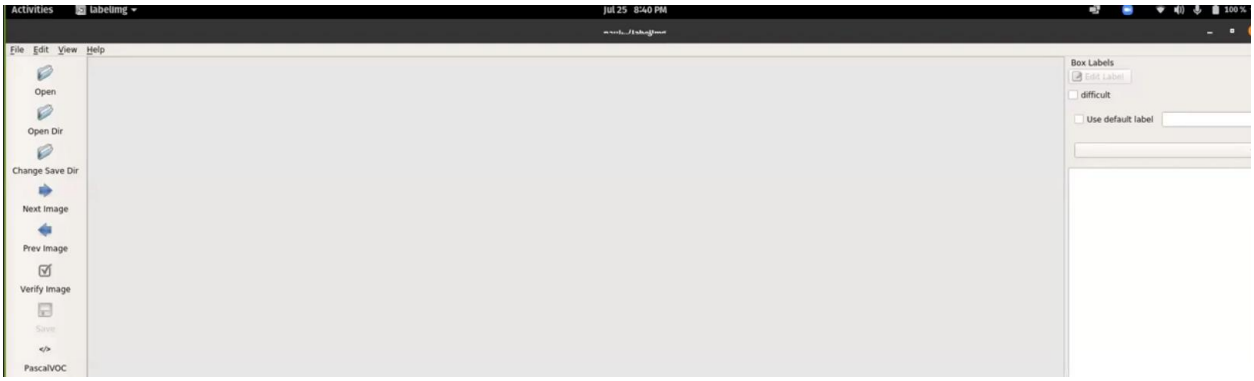
Link to download and Instructions to Install: https://github.com/tzutalin/labelImg



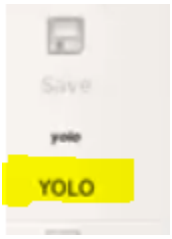
Go to the specific directory where our labelImg is located and then open the terminal and execute the below steps.

sudo apt-get install pyqt5-dev-tools
sudo pip3 install -r requirements/requirements-linux-python3.txt
make qt5py3
python3 labelImg.py
python3 labelImg.py [IMAGE_PATH] [PRE-DEFINED CLASS FILE]

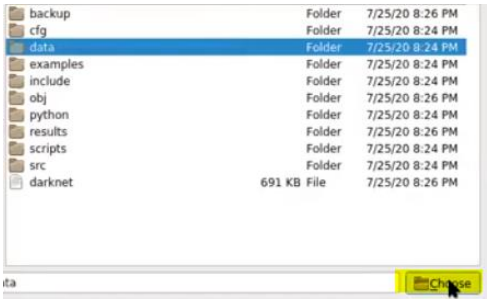
Here I'm going to execute the step to open the lableimg file



Now we need to change the PascalVOC to Yolo



Now open dir --- >Yolo --- > darknet --- > data (consider our dataset is located in this folder)



Now draw the bounding box for the different objects which are present in the picture.
Click Create RectBox and draw the bounding box and label them with class names.
Once the annotation is done then save each and every image in the dataset and the .txt file is created for each and every respective image file.

Open Yolo -- > darknet -- > data -- > then we will we able see many txt files

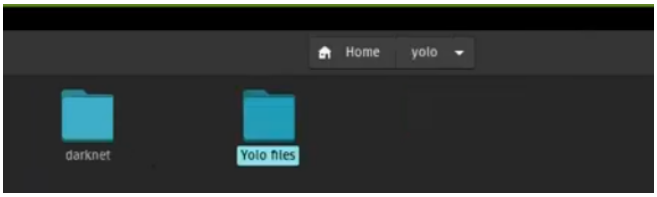
Sample txt file

Assume I have the new dataset under the folder name called " Yolo files" and I have done the annotation part.
Yolo files -- > Dataset -- > Images and labels folder

In this images folder I have the images and in the labels folder .txt files are available for corresponding each and every image in the images folder.

In TFOD our parent folder is the research folder but in case of Yolo our parent folder is darknet.

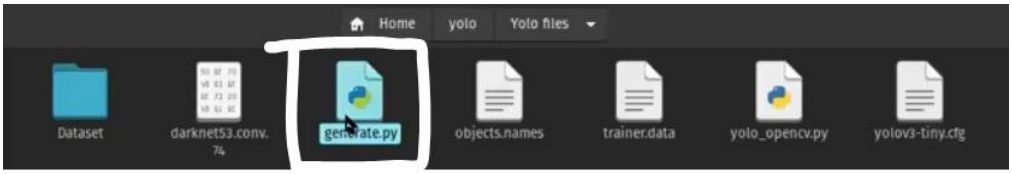
Copy the Yolo files to Yolo directory



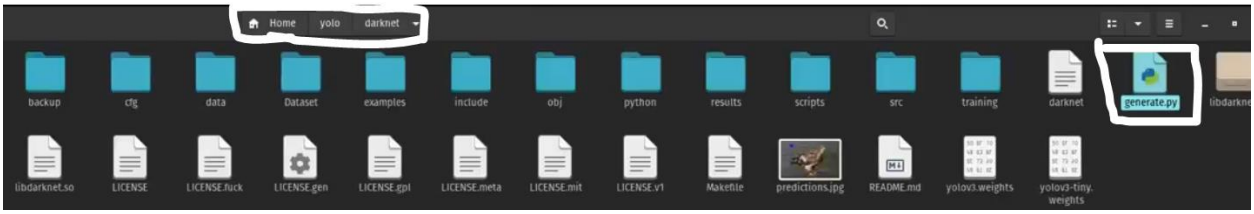
Now I'm going to create a new folder called **"training"** and in this folder I have to move some important files for our custom training.

Now I'm going to move the dataset (Yolo files -- > Dataset --- > copy --- > paste -- > Yolo --- > darknet.

Splitting the dataset (Train and Test):



In order to split the dataset, there is a separate script(geneate.py) is available inside the Yolo files -- > generate.py -- > copy -- > paste -- > Yolo -- > darknet

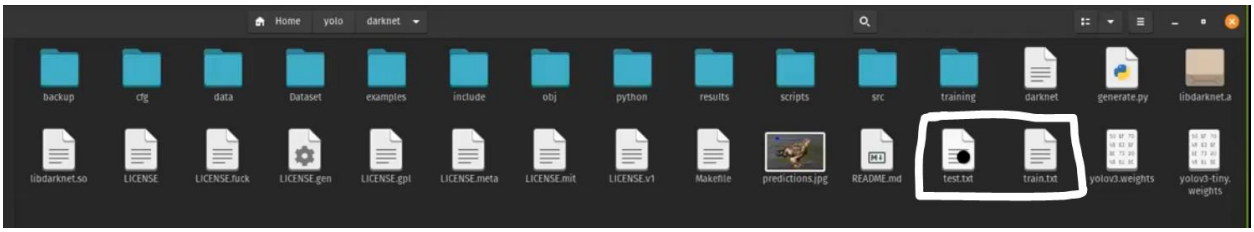


Open the generate.py file

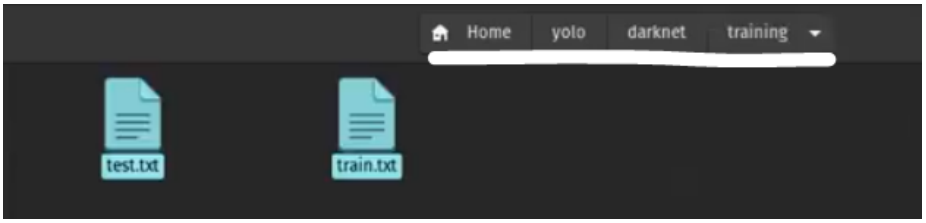
```python
import glob, os


dataset_path = '/home [redacted]/yolo/darknet/Dataset/images'

# Percentage of images to be used for the test set
percentage_test = 10;

# Create and/or truncate train.txt and test.txt
file_train = open('train.txt', 'w')
file_test = open('test.txt', 'w')

# Populate train.txt and test.txt
counter = 1
index_test = round(100 / percentage_test)
for pathAndFilename in glob.iglob(os.path.join(dataset_path, "*.jpg")):
    title, ext = os.path.splitext(os.path.basename(pathAndFilename))

    if counter == index_test+1:
        counter = 1
        file_test.write(dataset_path + "/" + title + '.jpg' + "\n")
    else:
        file_train.write(dataset_path + "/" + title + '.jpg' + "\n")
        counter = counter + 1
```

If you done any changes with respect to dataset path or percentage_test then save the file.
Now we need to execute that file(generate.py) in the terminal

Now the command got executed successfully and inside the darknet folder two files will be created 1. Train.txt 2. Test.txt

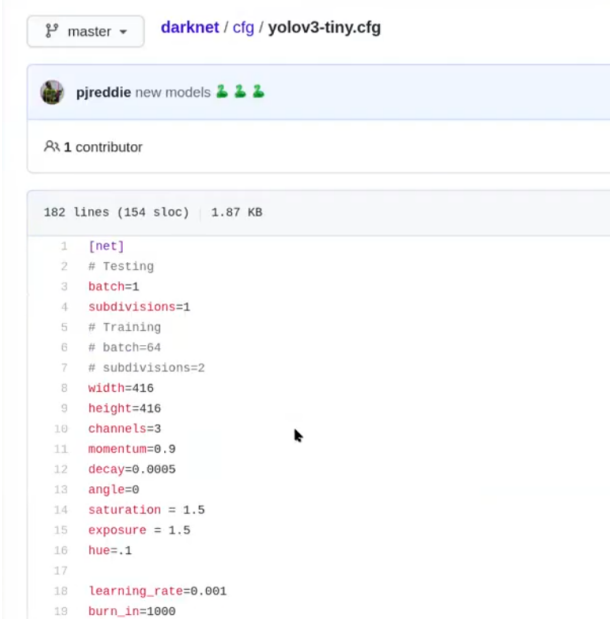


Now move train and test.txt to the training folder

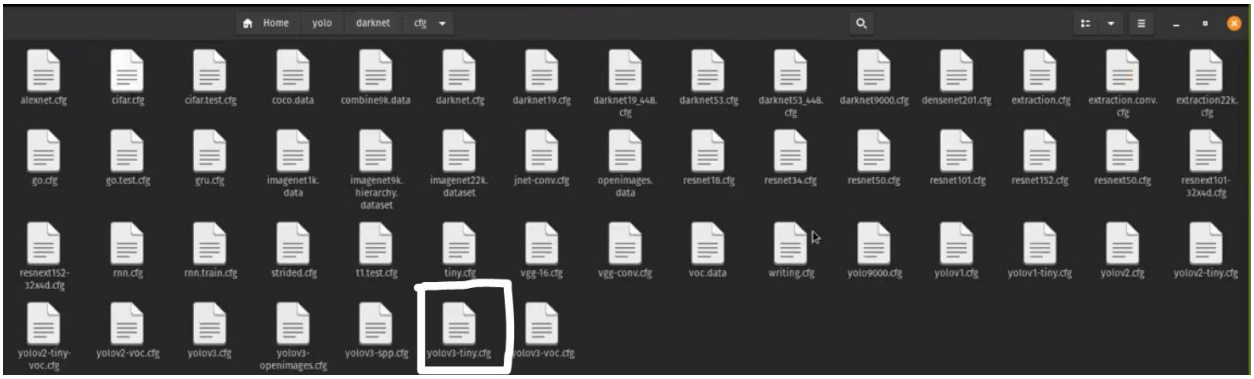


Config file:
Link to download the config file – Yolov3- tiny:
https://github.com/pjreddie/darknet/blob/master/cfg/yolov3-tiny.cfg

The config file looks like this.



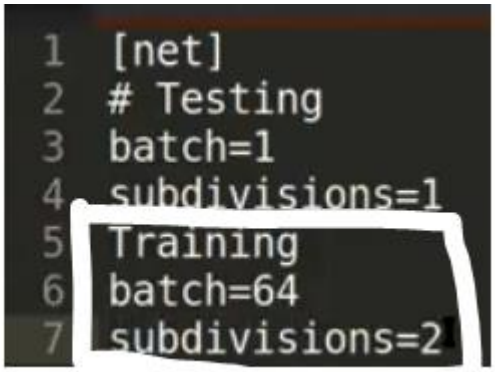
Whenever we download (darknet) we get the config files by default in our system
Yolo -- > darknet -- > cfg



Changes 1 (Uncomment some lines):

Open the Yolo tiny config file. Whenever we do the training then we need to remove the comments for specific lines like below (line no: 5 to 7) and change the batch = 64 and subdivision = 2

```
1  [net]
2  # Testing
3  batch=1
4  subdivisions=1
5  # Training
6  # batch=64
7  # subdivisions=2
8  width=416
9  height=416
10 channels=3
11 momentum=0.9
12 decay=0.0005
13 angle=0
14 saturation = 1.5
15 exposure = 1.5
16 hue=.1
17
18 learning_rate=0.001
19 burn_in=1000
20 max_batches = 500200
21 policy=steps
22 steps=400000,450000
23 scales=.1,.1
24
25 [convolutional]
26 batch_normalize=1
27 filters=16
28 size=3
29 stride=1
30 pad=1
31 activation=leaky
```



Change 2(No of Filters):

In the config file we need to do some changes before the Yolo layers
In that config layers we have two yolo layers and we need to do some changes on the convolution layer and it is residing before the yolo layer.

```
[convolutional]
size=1
stride=1
pad=1
filters=255
activation=linear


[yolo]
mask = 3,4,5
anchors = 10,14,  23,27,  37,58,  81,82,  135,169,  344,319
classes=80
num=6
jitter=.3
ignore_thresh = .7
truth_thresh = 1
random=1
```

Change the number of filters for convolution layer (just before the 2 Yolo part) based on the number of classes that we have.

Number of filters = 3 X (5 + no of classes) = 3 X (5 + 1) = 18

Change the filter 255 to 18 (line no : 127 and 171)

```
123 [convolutional]
124 size=1
125 stride=1
126 pad=1
127 filters=18
128 activation=linear
129
130
131
132 [yolo]
133 mask = 3,4,5
134 anchors = 10,14,  23,27,  37,58,  81,82,  135,169,  344,319
135 classes=80
136 num=6
137 jitter=.3
138 ignore_thresh = .7
139 truth_thresh = 1
140 random=1
```

```
167 [convolutional]
168 size=1
169 stride=1
170 pad=1
171 filters=18
172 activation=linear
173
174 [yolo]
175 mask = 0,1,2
176 anchors = 10,14,  23,27,  37,58,  81,82,  135,169,  344,319
177 classes=80
178 num=6
179 jitter=.3
180 ignore_thresh = .7
181 truth_thresh = 1
182 random=1
```

Change 3 (No of Classes):

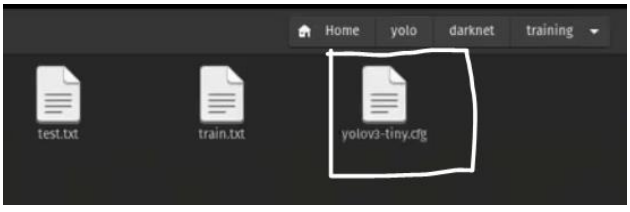
We have two yolo parts in our config files and we need to change the number of classes (line no: 135 and 177)

Change 80 to 1

```
132 [yolo]
133 mask = 3,4,5
134 anchors = 10,14,  23,27,  37,58,  81,82,  135,169,  344,319
135 classes=1
136 num=6
137 jitter=.3
138 ignore_thresh = .7
139 truth_thresh = 1
140 random=1
```

```
174 [yolo]
175 mask = 0,1,2
176 anchors = 10,14,  23,27,  37,58,  81,82,  135,169,  344,319
177 classes=1
178 num=6
179 jitter=.3
180 ignore_thresh = .7
181 truth_thresh = 1
182 random=1
```

Save the changes and copy the yolo-tiny config file and paste in to the training folder.

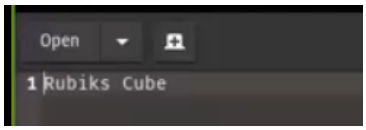


Files Needed:

Basically this object.name is found in Yolo -- > Yolo files -- > object.name

1. Create a file name called **"object.names"**

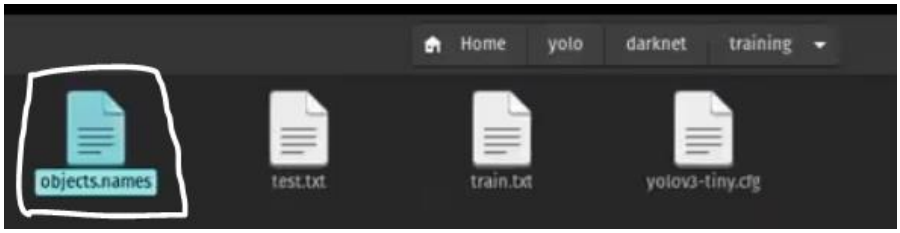
   This object.names file contains how many classes we and we need to specify only the class name inside this file



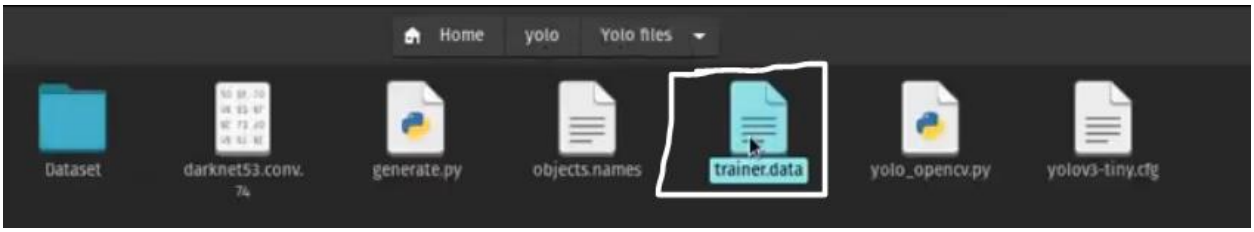
   While annotation I gave this class name and we need to mention the same name inside the object.name file. If we have multiple classes and we need to specify those classes in a same manner that we have in the labelImg tool and in the object.name file just hit enter and mention the class name.

   Now move the object.file to the training folder.

   Yolo -- > darknet -- > training -- > paste the object.file



2. Next thing is another file and where will be mentioning that the path of the file, images. The name of the file is "**trainer.data"**
   Yolo -- > yolo files -- > trainer.data



   The file looks like below

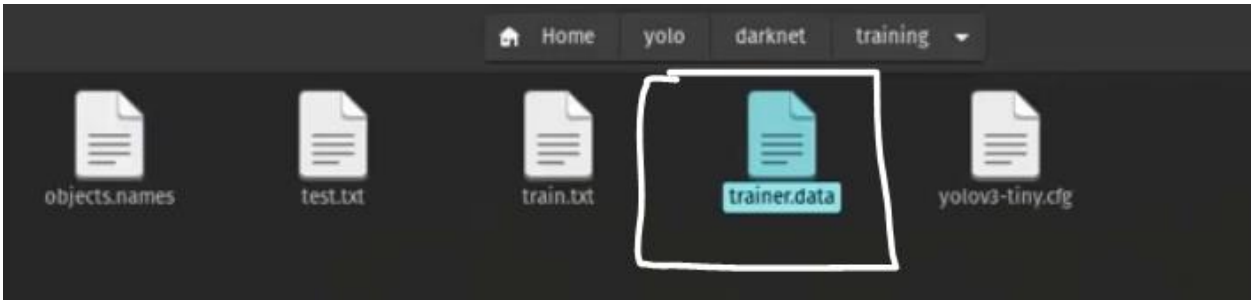
```
1 classes= 1
2 train  = custom/train.txt
3 valid  = custom/test.txt
4 names = custom/objects.names
5 backup = backup/
6
```

```
classes= 1
train  = training/train.txt
valid  = training/test.txt
names = training/objects.names
backup = backup/
```

Here we need to specify or do some changes on the total number of classes and we need to specify the train path, valid or test path and names.

After the changes are done save the file.

Now copy the file trainer.data -- > Yolo -- > darknet -- > training -- > paste the file



3. Pretrained models and this file size is around 155MB and this is the model file for the yolo framework (transfer learning approach) and this model is trained on COCO dataset
   Link to download: https://pjreddie.com/media/files/darknet53.conv.74

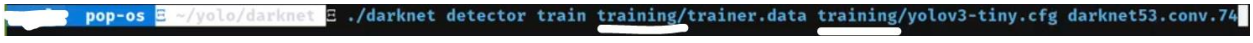
   Once the file is downloaded then move that file in to the darknet folder.

Start the training:

```
Training Command :- ./darknet detector train custom/trainer.data custom/yolov3-tiny.cfg darknet53.conv.74
```

Instead of custom we need to give our folder name called “training”
Command: ./darknet detector train training/trainer.data training/yolov3-tiny.cfg darknet53.conv.74

Navigate to the darknet folder in the terminal

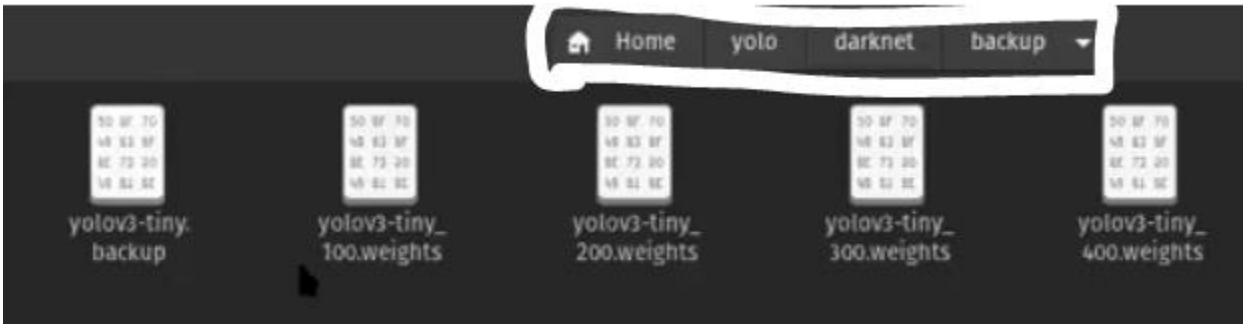


```
pop-os ~/yolo/darknet ./darknet detector train training/trainer.data training/yolov3-tiny.cfg darknet53.conv.74
yolov3-tiny
layer     filters    size              input                output
    0 conv     16  3 x 3 / 1   416 x 416 x   3   ->   416 x 416 x  16  0.150 BFLOPs
    1 max          2 x 2 / 2   416 x 416 x  16   ->   208 x 208 x  16
    2 conv     32  3 x 3 / 1   208 x 208 x  16   ->   208 x 208 x  32  0.399 BFLOPs
    3 max          2 x 2 / 2   208 x 208 x  32   ->   104 x 104 x  32
    4 conv     64  3 x 3 / 1   104 x 104 x  32   ->   104 x 104 x  64  0.399 BFLOPs
    5 max          2 x 2 / 2   104 x 104 x  64   ->    52 x  52 x  64
    6 conv    128  3 x 3 / 1    52 x  52 x  64   ->    52 x  52 x 128  0.399 BFLOPs
    7 max          2 x 2 / 2    52 x  52 x 128   ->    26 x  26 x 128
    8 conv    256  3 x 3 / 1    26 x  26 x 128   ->    26 x  26 x 256  0.399 BFLOPs
    9 max          2 x 2 / 2    26 x  26 x 256   ->    13 x  13 x 256
   10 conv    512  3 x 3 / 1    13 x  13 x 256   ->    13 x  13 x 512  0.399 BFLOPs
   11 max          2 x 2 / 1    13 x  13 x 512   ->    13 x  13 x 512
   12 conv   1024  3 x 3 / 1    13 x  13 x 512   ->    13 x  13 x1024  1.595 BFLOPs
   13 conv    256  1 x 1 / 1    13 x  13 x1024   ->    13 x  13 x 256  0.089 BFLOPs
   14 conv    512  3 x 3 / 1    13 x  13 x 256   ->    13 x  13 x 512  0.399 BFLOPs
   15 conv     18  1 x 1 / 1    13 x  13 x 512   ->    13 x  13 x  18  0.003 BFLOPs
   16 yolo
   17 route  13
   18 conv    128  1 x 1 / 1    13 x  13 x 256   ->    13 x  13 x 128  0.011 BFLOPs
   19 upsample            2x    13 x  13 x 128   ->    26 x  26 x 128
   20 route  19 8
   21 conv    256  3 x 3 / 1    26 x  26 x 384   ->    26 x  26 x 256  1.196 BFLOPs
   22 conv     18  1 x 1 / 1    26 x  26 x 256   ->    26 x  26 x  18  0.006 BFLOPs
   23 yolo
Loading weights from darknet53.conv.74...Done!
Learning Rate: 0.001, Momentum: 0.9, Decay: 0.0005
Resizing
608
Loaded: 0.044022 seconds
```

```
Loading weights from darknet53.conv.74...Done!
Learning Rate: 0.001, Momentum: 0.9, Decay: 0.0005
Resizing
608
Loaded: 0.044022 seconds
Region 16 Avg IOU: 0.545788, Class: 0.499931, Obj: 0.499777, No Obj: 0.499833, .5R: 1.000000, .75R: 0.000000,  count: 1
Region 23 Avg IOU: -nan, Class: -nan, Obj: -nan, No Obj: 0.498697, .5R: -nan, .75R: -nan,  count: 0
1: 674.829407, 674.829407 avg, 0.000000 rate, 8.345487 seconds, 1 images
Loaded: 0.000094 seconds
Region 16 Avg IOU: 0.547513, Class: 0.501211, Obj: 0.499340, No Obj: 0.499834, .5R: 1.000000, .75R: 0.000000,  count: 1
Region 23 Avg IOU: -nan, Class: -nan, Obj: -nan, No Obj: 0.498692, .5R: -nan, .75R: -nan,  count: 0
2: 674.596436, 674.806091 avg, 0.000000 rate, 8.171865 seconds, 2 images
Loaded: 0.000073 seconds
Region 16 Avg IOU: 0.682104, Class: 0.501204, Obj: 0.499346, No Obj: 0.499833, .5R: 1.000000, .75R: 0.000000,  count: 1
Region 23 Avg IOU: -nan, Class: -nan, Obj: -nan, No Obj: 0.498693, .5R: -nan, .75R: -nan,  count: 0
3: 674.759766, 674.801453 avg, 0.000000 rate, 8.190158 seconds, 3 images
Loaded: 0.000049 seconds
```

This avg is basically the average loss value.

Once the training done, in backup folder we will be able to see the weights file like below.



Prediction Part:

To do the next thing the module in the opencv and which is responsible for read the DNN module (neural network modules).

Yolo -- > Yolo files -- > open yolo_opencv.py

Change the path of config, weights and classes for live webcam detection

```
import cv2
import argparse
import numpy as np

ap = argparse.ArgumentParser()
ap.add_argument('-c', '--config',
                help = 'path to config file', default="/path/to/yolov3-tiny.cfg")
ap.add_argument('-w', '--weights',
                help = 'path to pre-trained weights', default="/path/to/yolov3-tiny_final.weights")
ap.add_argument('-cl', '--classes',
                help = 'path to objects.names',default="/path/to/objects.names")
args = ap.parse_args()


# Get names of output layers, output for YOLOv3 is ['yolo_16', 'yolo_23']
def getOutputsNames(net):
    layersNames = net.getLayerNames()
    return [layersNames[i[0] - 1] for i in net.getUnconnectedOutLayers()]


# Darw a rectangle surrounding the object and its class name
def draw_pred(img, class_id, confidence, x, y, x_plus_w, y_plus_h):

    label = str(classes[class_id])

    color = COLORS[class_id]

    cv2.rectangle(img, (x,y), (x_plus_w,y_plus_h), color, 2)

    cv2.putText(img, label, (x-10,y-10), cv2.FONT_HERSHEY_SIMPLEX, 0.5, color, 2)

# Define a window to show the cam stream on it
window_title= "Rubiks Detector"
cv2.namedWindow(window_title, cv2.WINDOW_NORMAL)


# Load names classes
classes = None
with open(args.classes, 'r') as f:
    classes = [line.strip() for line in f.readlines()]
print(classes)

#Generate color for each class randomly
COLORS = np.random.uniform(0, 255, size=(len(classes), 3))

# Define network from configuration file and load the weights from the given weights file
net = cv2.dnn.readNet(args.weights,args.config)

# Define video capture for default cam
cap = cv2.VideoCapture(0)


while cv2.waitKey(1) < 0 or False:

```

```
Do the prediction/testing via live cam
conda create --name yolo python=3.6.10
conda activate opencv
#install the opencv package
conda install -c conda-forge opencv
or
pip install opencv-python

cd to project_directory

python yolo_opencv.py -c /path/to/yolov3-tiny.cfg -w /path/to/yolov3-tiny_finally.weights -cl /path/to/objects.names
```

Note:

- Resuming the training process from specific iterations like TFOD is not possible in Yolo.
- The weights will be saved in the backup folder
- Weights will be saved for every 100 iterations till 900 and then it saves the weight for every 10000.
- Stop or kill the training process once the average loss is reduced to 0.06 or once the average loss is no longer decreases.
- Right now, we have implemented the CPU version. If you have GPU do some changes like GPU=1 on the Makefile
- If you have installed cuda then CUDNN=1
- If you build opencv from scratch then do changes like OPENCV=1
- For prediction comment the line no 5,6,7 and uncomment the testing in yolo-tiny.cfg (config file) (Yolo -- > darknet -- > cfg)