DMA PROJECT REPORT

ON

OBJECT DETECTION USING YOLOv3 AND OPENCV

Submitted by:

ANMOL CHHABRA
(LCO17378)
VINAY KUMAR PRASHAR
(LCO17378)
RAJAT KUTHIAL
(LCO16374)

CHANDIGARH COLLEGE OF ENGINEERING AND TECHNOLOGY
(DEGREE WING)
(Government Institute Under UT Administration | Affiliated to Punjab University, Chandigarh)
Sector 26, Chandigarh, PIN-160019
JUNE, 2020

OBJECT DETECTION USING YOLOv3 AND OPENCV

1. ABSTRACT

In this project we are detecting an image which contains different objects like man, woman, animal, things etc. using different concepts and technique of images used in Computer Vision. We are also detecting objects in the video by dividing the video into the frames and applying object detection on frames, whole video part is done by using the Open CV.

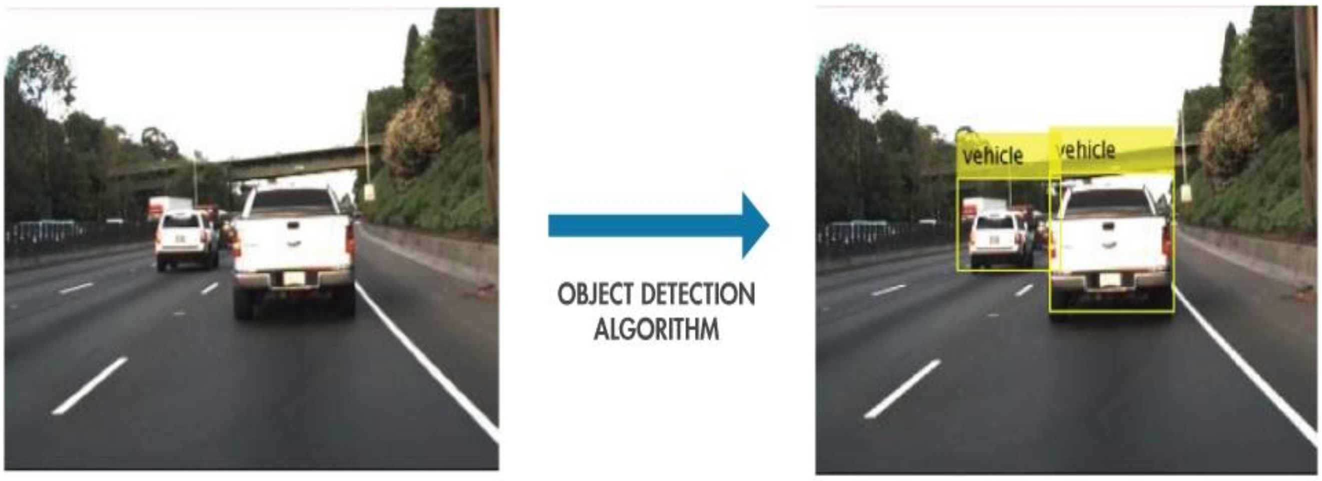
Throughout our project in computer vision field, we were a lot confused by terms explained in this, but we decided to break these terminologies which were able to help us understand the differences between each of the terms. We took different images and saw different objects in it, therefore we started classifying the objects in it. By performing the classifications, we understood the meaning of Image Classification. We got more instances and carried our deep learning into this field, and took it to next approach where we had an image which consists of different objects in it. We got to know that for this we would be using the technique of Object Detection and since we started making efforts, we came through a lot of challenges. Since such an image can have multiple objects, we won't be able to implement the Object Detection technique. Facing this challenge, we got to know about the term Image Segmentation in which we can make groups and turn them into segments. After making segments it would be easy for us to configure/train our model which could understand what type of image is it, what are the objects identified by the model.

2. INTRODUCTION

What is Object Detection?

Object Detection is a common Computer Vision problem which deals with identifying and locating object of certain classes in the image. Interpreting the object localization can be done in various ways, including creating a bounding box around the object or marking every pixel in the image which contains the object (called segmentation).

Object detection is a key technology behind advanced driver assistance systems (ADAS) that enable cars to detect driving lanes or perform pedestrian detection to improve road safety. Object detection is also useful in applications such as video surveillance or image retrieval systems.



(Example of Object Detection Algorithm)

Difference between Object detection and image classification?

In the field of Computer Vision, the doubt which is most common we have is what is the difference between the classification of image, detection of object and segmentation of image. Let's take an image of a dog as shown below, we start making classifications of it to which it belongs (like a dog, which is an instance here). And that what is known as the Image Classification.

As we go through, we see that there was only a particular object, a dog. We can easily make use of such image classification model and try predicting that we have an image of a dog. But, just imagine the fact if we have double images like one of cat and other of dog, how would we classify them?

For this problem we need to train a multi-label classifier. By doing this we would see another problem that we aren't aware of the location of either animal/object in the image.

Therefore, we make use of Image Localization which helps to identify the object location in that particular image. In that case, if we have multiple objects in an image, we go for the next step known as Object Detection. Now we are able to predict the location and the class for each object in an image using Object Detection.

In order to detect the objects and classifying the images, we first require the understanding of what an image comprises of. Now comes the role of Image Segmentation. Either we can divide or make partitions of an image into parts known to be segments. It won't be reliable to make a processing of an entire image at the very same time because there will be some regions where we do not contain information of the image. By making divisions in the image and converting it into segments, we can pick some of the important segments for processing of an image. This is how Image Segmentation has its working.

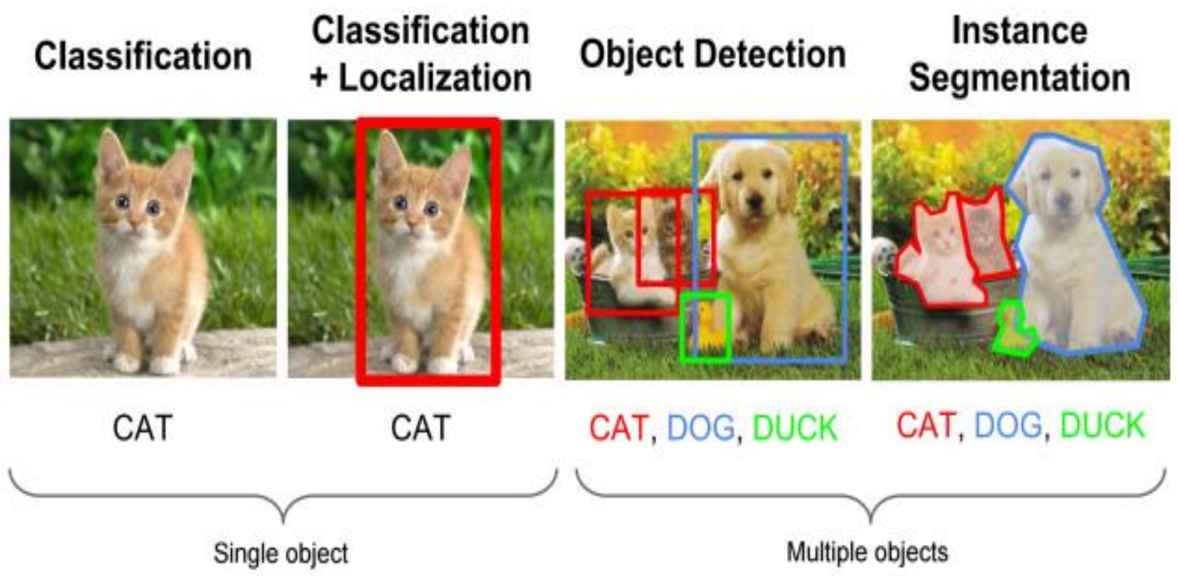
Knowing the definition of the image that it is a collection of pixels; we start making groups of the pixels which have similar attributes using the Image Segmentation method.

After applying Object Detection method, we are only able to build up a bounding box which is corresponding to each of the class in an image. But it won't be letting us know anything about the shape of the image since the bounding boxes maybe rectangular or square in shape type.

The models of image segmentation on the other hand will be able to create a pixel-wise mask for each type of object in an image. Such type of technique gives us far more granular way of understanding the object(s) in an image.

Now summarizing each method, we observed:

1. **Image Classification** helps in classification of what is contained in a particular image
2. **Image Localization** does the specification of location of a single object in a particular image
3. **Object Detection** specifies the location of multiple images
4. **Image Segmentation** does creation of pixels wise mask of each object in a particular image



(Difference between Image Classification and Classification + localization and Object Detection and Instance segmentation)

What is YOLO?

You only look once (YOLO) is a state-of-the-art, real-time object detection system. Object detection is one of the classical problems in computer vision where you work to recognize *what* and *where* — specifically what objects are inside a given image and also where they are in the image. The problem of object detection is more complex than classification, which also can recognize objects but doesn't indicate where the object is located in the image. In addition, classification doesn't work on images containing more than one object.

YOLO uses a totally different approach. YOLO is a clever convolutional neural network (CNN) for doing object detection in real-time. The algorithm applies a single neural network to the full image, and then divides the image into regions and predicts bounding boxes and probabilities for each region. These bounding boxes are weighted by the predicted probabilities.

YOLO is popular because it achieves high accuracy while also being able to run in real-time.

YOLOv3

YOLOv3 is extremely fast and accurate. In mAP measured at .5 IOU YOLOv3 is on par with Focal Loss but about 4x faster. Moreover, you can easily tradeoff between speed and accuracy simply by changing the size of the model, no retraining required!

How it works?

Prior detection systems repurpose classifiers or localizers to perform detection. They apply the model to an image at multiple locations and scales. High scoring regions of the image are considered detections.

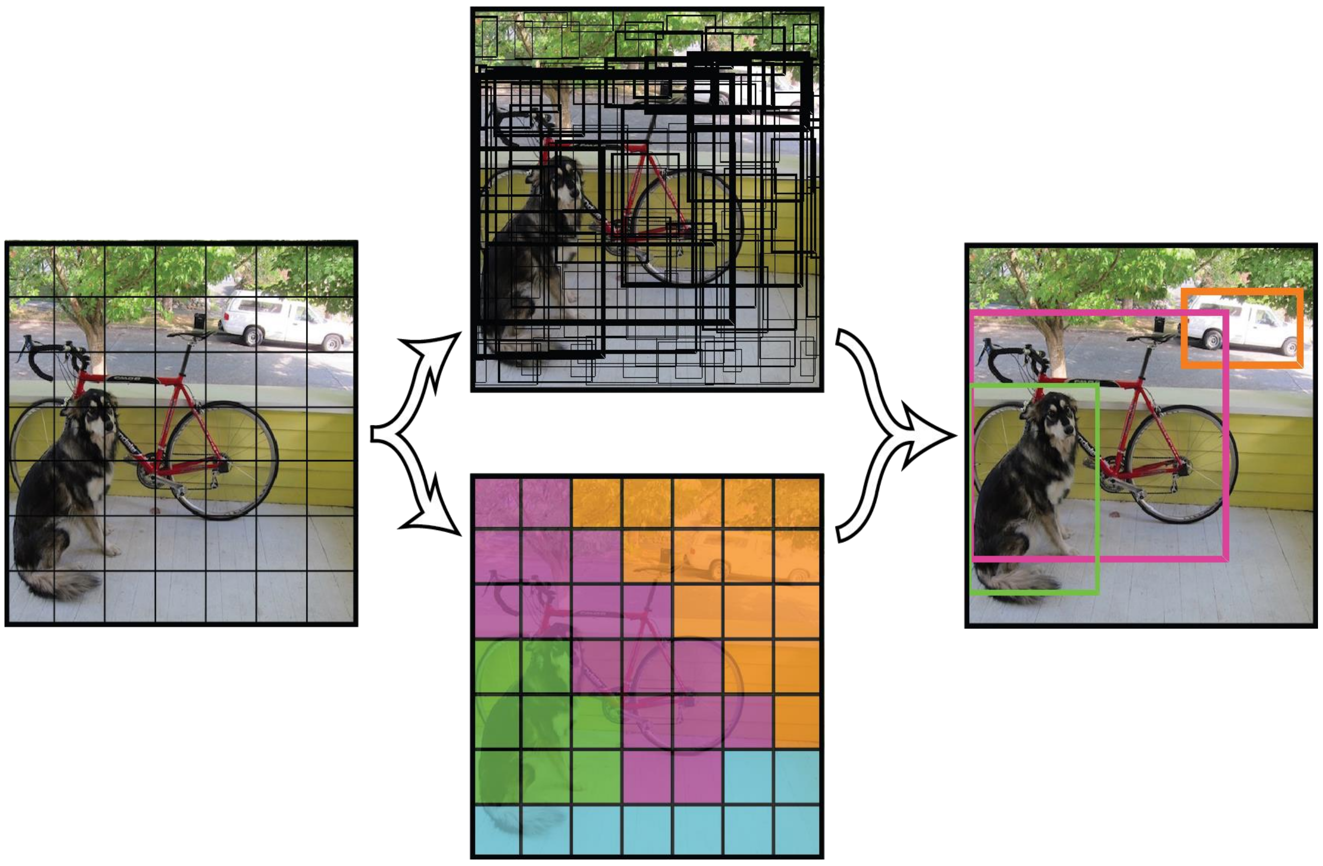
YOLOv3 uses a few tricks to improve training and increase performance, including: multi-scale predictions, a better backbone classifier, and more.

Feature Extractor Network (Darknet-53):

DarkNet-53 is a convolutional neural network that is 53 layers deep. You can load a pretrained version of the network trained on more than a million images from the ImageNet database [1]. The pretrained network can classify images into 1000 object categories, such as keyboard, mouse, pencil, and many animals. As a result, the network has learned rich feature representations for a wide range of images. The network has an image input size of 256-by-256. DarkNet-53 is often used as the foundation for object detection problems and YOLO workflows.

YOLO v3 also uses a variant of Darknet, for the task of detection, 53 more layers are stacked onto it, giving us a **106 layer fully convolutional underlying architecture for YOLO v3**. This is the reason behind the slowness of YOLO v3 compared to YOLO v2. Here is how the architecture of YOLO now looks like.

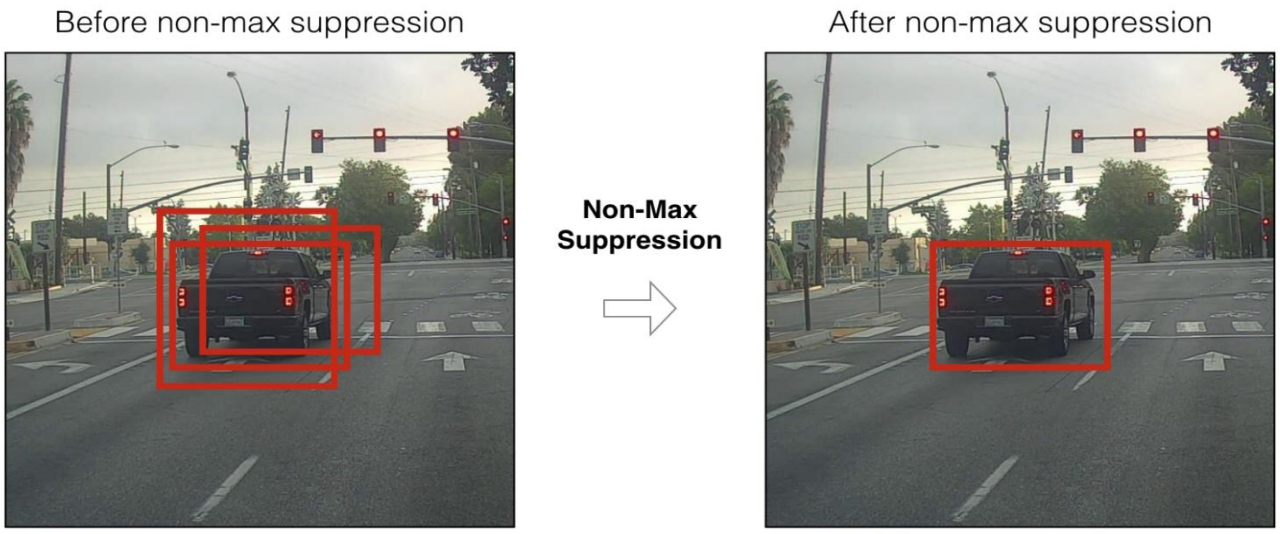
Example:



(Example of How Yolov3 Detects objects)

Non-maximum suppression:

We have also used the non-maximum suppression in this. Non-Maximum Suppression (NMS) is a technique used in many computer vision algorithms. It is a class of algorithms to select one entity (e.g. bounding boxes) out of many overlapping entities. The selection criteria can be chosen to arrive at particular results. Most commonly, the criteria are some form of probability number along with some form of overlap measure



(Example of Non-Maximum suppression)

3. REQUIREMENT

Following are the requirements that we need in this project and their description (as per there documentation):

- **OpenCV 3.4:** OpenCV (Open Source Computer Vision Library) is an open source computer vision and machine learning software library. OpenCV was built to provide a common infrastructure for computer vision applications and to accelerate the use of machine perception in the commercial products. Being a BSD-licensed product, OpenCV makes it easy for businesses to utilize and modify the code.
- **Python 3.6:** Python is a popular programming language. It was created by Guido van Rossum, and released in 1991.

  It is used for:
  - web development (server-side),
  - software development,
  - mathematics,
  - system scripting.
- **Tensorflow-gpu 1.5.0:** TensorFlow is an open source software library for high performance numerical computation. Its flexible architecture allows easy deployment of computation across a variety of platforms (CPUs, GPUs, TPUs), and from desktops to clusters of servers to mobile and edge devices.

- **Keras 2.1.3:** Keras is an open-source neural-network library written in Python. Designed to enable fast experimentation with deep neural networks, it focuses on being user-friendly, modular, and extensible. It was developed as part of the research effort of project ONEIROS (Open-ended Neuro-Electronic Intelligent Robot Operating System), and its primary author and maintainer is François Chollet, a Google engineer. Keras contains numerous implementations of commonly used neural-network building blocks such as layers, objectives, activation functions, optimizers, and a host of tools to make working with image and text data easier to simplify the coding necessary for writing deep neural network code. The code is hosted on GitHub, and community support forums include the GitHub issues page, and a Slack channel.
- **Flask:** Flask is a micro web framework written in Python. It is classified as a microframework because it does not require particular tools or libraries.[3] It has no database abstraction layer, form validation, or any other components where pre-existing third-party libraries provide common functions. However, Flask supports extensions that can add application features as if they were implemented in Flask itself. Extensions exist for object-relational mappers, form validation, upload handling, various open authentication technologies and several common framework related tools.

4. DATASET:

YOLOv3 model uses pre-trained weights for standard object detection problems. So, for training YOLOv3 we use convolutional weights that are pre-trained on **Imagenet**. We use weights from the darknet53 model.
We can just download the wights for the YOLOv3 from the given link:
https://pjreddie.com/media/files/yolov3.weights
After downloading the weights, we will just put the weight file in the main directory.

Imagenet:

ImageNet is an image database organized according to the WordNet hierarchy (currently only the nouns), in which each node of the hierarchy is depicted by hundreds and thousands of images. Currently we have an average of over five hundred images per node.
We can say that ImageNet is a large dataset of annotated photographs intended for computer vision research. ImageNet is still an ongoing research to provide researchers around the world an easily accessible image database.

ImageNet consists of 14,197,122 images organized into 21,841 subcategories. These subcategories can be considered as sub-trees of 27 high-level categories. Thus, ImageNet is a well-organized hierarchy that makes it useful for supervised machine learning tasks.

Pre-trained Models using Imagenet:

A model trained on ImageNet has essentially learned to identify both low-level and high-level features in images. However, in a real-world application such as medical image analysis or handwriting recognition, models have to be trained from data drawn from those application domains. This is time consuming and sometimes impossible due to lack of sufficient annotated training data.

Limitations of Imagenet:

- Though ImageNet has a large number of classes, most of them don't represent everyday entities.
- Images are not uniformly distributed across subcategories. One research team found that by considering 200 subcategories, they found that the top 11 had 50% of the images, followed by a long tail.
- When classifying people, ImageNet uses labels that are racist, misogynist and offensive. People are treated as objects. Their photos have been used without their knowledge.
- One study noted that ImageNet lacks geodiversity. Most of the data represents North America and Europe. China and India are represented in only 1% and 2.1% of the images respectively. This implies that models trained on ImageNet will not work well when applied for the developing world.
- Another study from 2016 found that 30% of ImageNet's image URLs are broken. This is about 4.4 million annotations lost.

5. IMPLEMENTATION

Making the .h5 from the yolo weights:

An H5 file is a data file saved in the Hierarchical Data Format (HDF). It contains multidimensional arrays of scientific data.

The yolo.weights (YOLOv3 Weights) are converted to yolo.h5 using the following command:

python yolov3.weights data\yolo.h5

Getting the YOLO Configuration File:

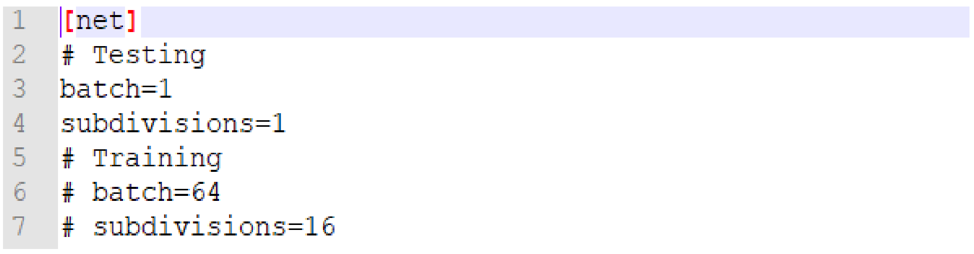
YOLOv3 also needs a configuration file **darknet-yolov3.cfg**. It is also included in our code base. It is based on the demo configuration file, yolov3-voc.cfg (comes with darknet code), which was used to train on the VOC dataset. All the important training parameters are stored in this configuration file.

To get the yolov3.cfg file we will look into the github repository of the "***pjreddie***":

https://github.com/pjreddie/darknet/blob/master/cfg/yolov3.cfg

Different Parameters defined in YOLOv3 Configuration File:

- **Batch hyper-parameter in YOLOv3:** The **batch** parameter indicates the **batch size** used during training. It is impractical (and unnecessary) to use all images in the training set at once to update the weights. So, a small subset of images is used in one iteration, and this subset is called the **batch size**.

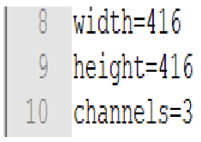
  ```
  [net]
  # Testing
  batch=1
  subdivisions=1
  # Training
  # batch=64
  # subdivisions=16
  ```

  When the batch size is set to 64, it means 64 images are used in one iteration to update the parameters of the neural network.

- **Subdivisions configuration parameter in YOLOv3:** Darknet allows you to specify a variable called **subdivisions** that lets you process a fraction of the batch size at one time on your GPU.

  Example: You can start the training with subdivisions=1, and if you get an *Out of memory error*, increase the subdivisions parameter by multiples of 2(e.g. 2, 4, 8, 16) till the training proceeds successfully. The GPU will process $batch/subdivision$ number of images at any time, but the full batch or iteration would be complete only after all the 64 (as set above) images are processed.

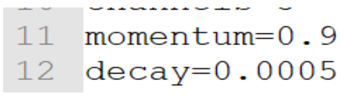
- **Width, Height, Channels:** These configuration parameters specify the input image size and the number of channels.

  ```
  width=416
  height=416
  channels=3
  ```

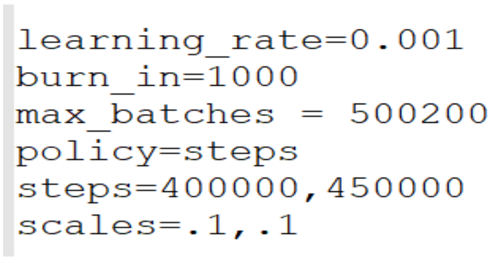
  The input training images are first resized to **width** x **height** before training. Here we use the default values of 416×416. The results might improve if we increase it to 608×608, but it would take longer to train too. **channels**=3 indicates that we would be processing 3-channel RGB input images.

- **Momentum and Decay:** The configuration file contains a few parameters that control how the weight is updated. **Momentum** is used to penalize large weight changes between iterations. The parameter **decay** controls this penalty term.



```
momentum=0.9
decay=0.0005
```

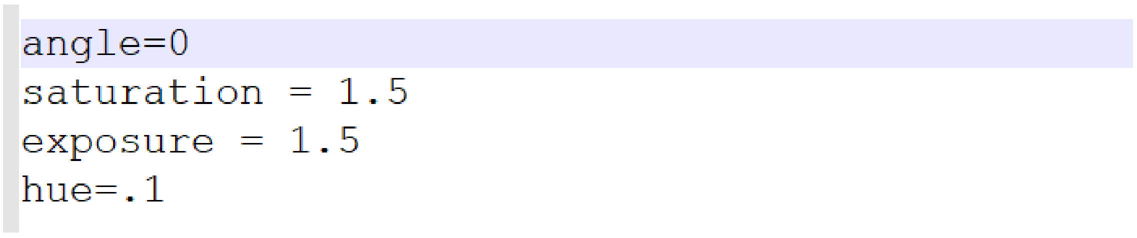
- **Learning Rate, Steps, Scales, Burn In:** The parameter **learning rate** controls how aggressively we should learn based on the current batch of data. Typically, this is a number between 0.01 and 0.0001.



```
learning_rate=0.001
burn_in=1000
max_batches = 500200
policy=steps
steps=400000,450000
scales=.1,.1
```

At the beginning of the training process, we are starting with zero information and so the learning rate needs to be high. But as the neural network sees a lot of data, the weights need to change less aggressively. In other words, the learning rate needs to be decreased over time. In the configuration file, this decrease in learning rate is accomplished by first specifying that our learning rate decreasing **policy** is **steps**. In the above example, the learning rate will start from 0.001 and remain constant for 3800 iterations, and then it will multiply by **scales** to get the new learning rate.
It has been empirically found that the training speed tends to increase if we have a lower learning rate for a short period of time at the very beginning. This is controlled by the **burn_in** parameter.

- **Data augmentation:** Data augmentation is a strategy that enables practitioners to significantly increase the diversity of data available for training models, without actually collecting new data. Data augmentation techniques such as cropping, padding, and horizontal flipping are commonly used to train large neural networks.
The **angle** parameter in the configuration file allows you to randomly rotate the given image by ± angle.
We can also transform the colors of the entire picture using **saturation**, **exposure**, and **hue**



```
angle=0
saturation = 1.5
exposure = 1.5
hue=.1
```

- **Number of iterations:** Finally, we need to specify how many iterations should the training process be run for.

  Max_batches=500200

  For multi-class object detectors, the max_batches number is higher, i.e. we need to run for a greater number of batches

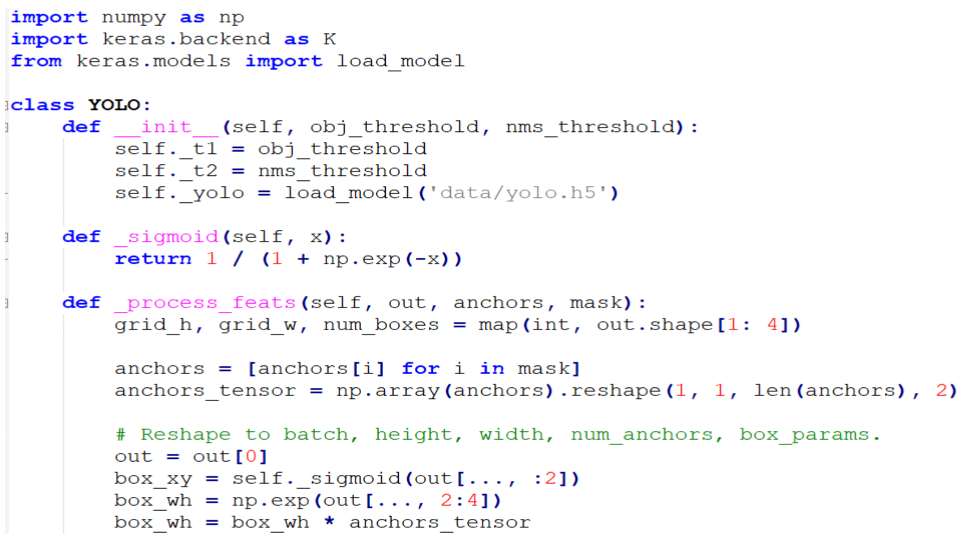
Building Models:

We have made a model yolo_model.py using the keras.models.

This model will help us in detecting the object and creating the boxes and the layout on them.

Model is build using several functions such as:

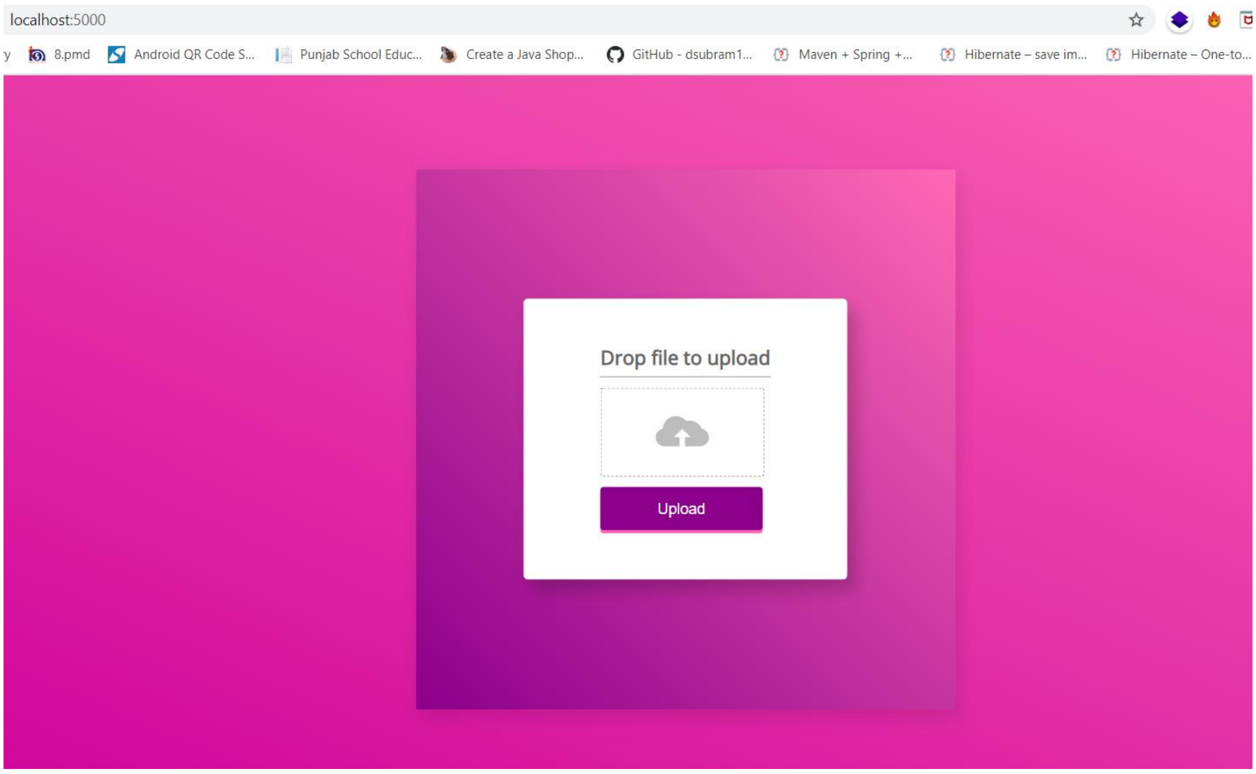
- Yolo_output: This function will help is predicting the output and several methods will be called from this method.
- _process_feats: This function will the boxes to be drawn with the box confidence and the probable class to which the box object belongs to.
- _filter_boxes: It will help in filtering the boxes thar are to be drawn around the objects.
- _nms_boxes: This function will suppress the non-maximum boxes. In Layman terms, "It removes the overlapping of the boxes."
- Yolo_predict: This function is used to predict the object detection on the image or on the particular video file.



```
import numpy as np
import keras.backend as K
from keras.models import load_model

class YOLO:
    def __init__(self, obj_threshold, nms_threshold):
        self._t1 = obj_threshold
        self._t2 = nms_threshold
        self._yolo = load_model('data/yolo.h5')

    def _sigmoid(self, x):
        return 1 / (1 + np.exp(-x))

    def _process_feats(self, out, anchors, mask):
        grid_h, grid_w, num_boxes = map(int, out.shape[1: 4])

        anchors = [anchors[i] for i in mask]
        anchors_tensor = np.array(anchors).reshape(1, 1, len(anchors), 2)

        # Reshape to batch, height, width, num_anchors, box_params.
        out = out[0]
        box_xy = self._sigmoid(out[..., :2])
        box_wh = np.exp(out[..., 2:4])
        box_wh = box_wh * anchors_tensor
```

Expected Input:

The input is expected in the form of the video file upload through an html file.



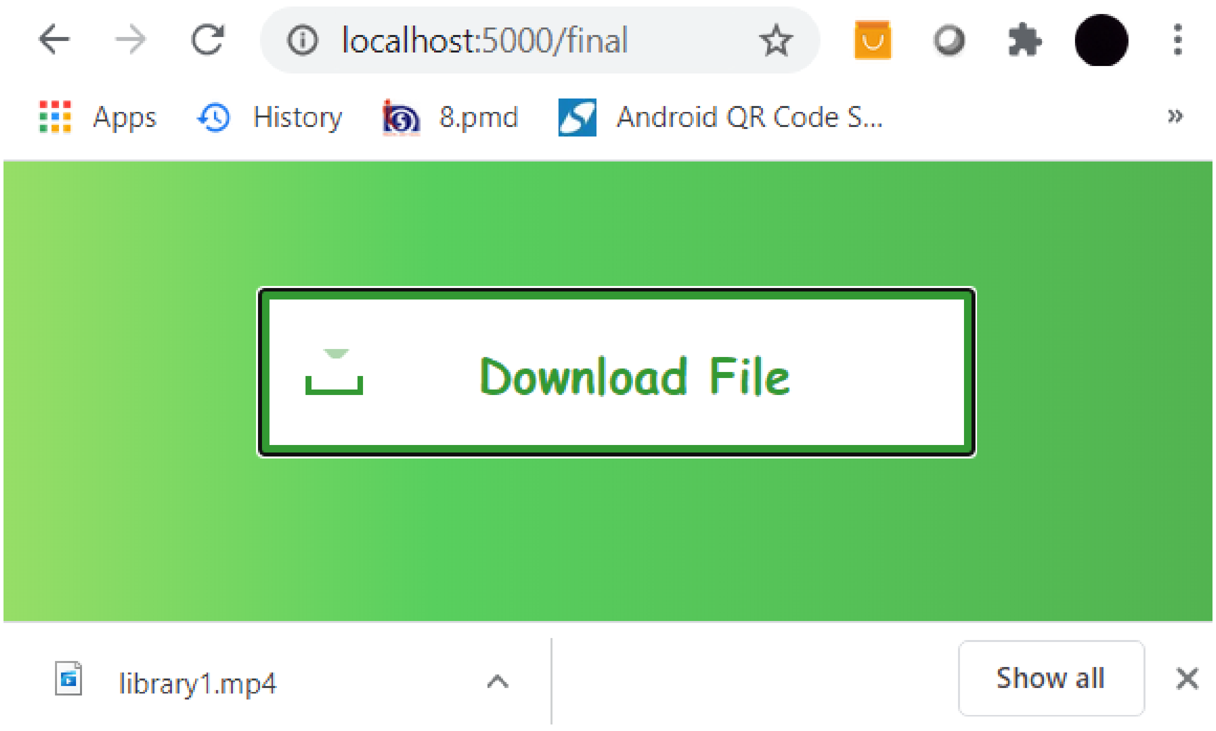
(HTML page to upload a video to detect objects)

The other of an giving an input is in the form of an image that can be place in the images/test folder.

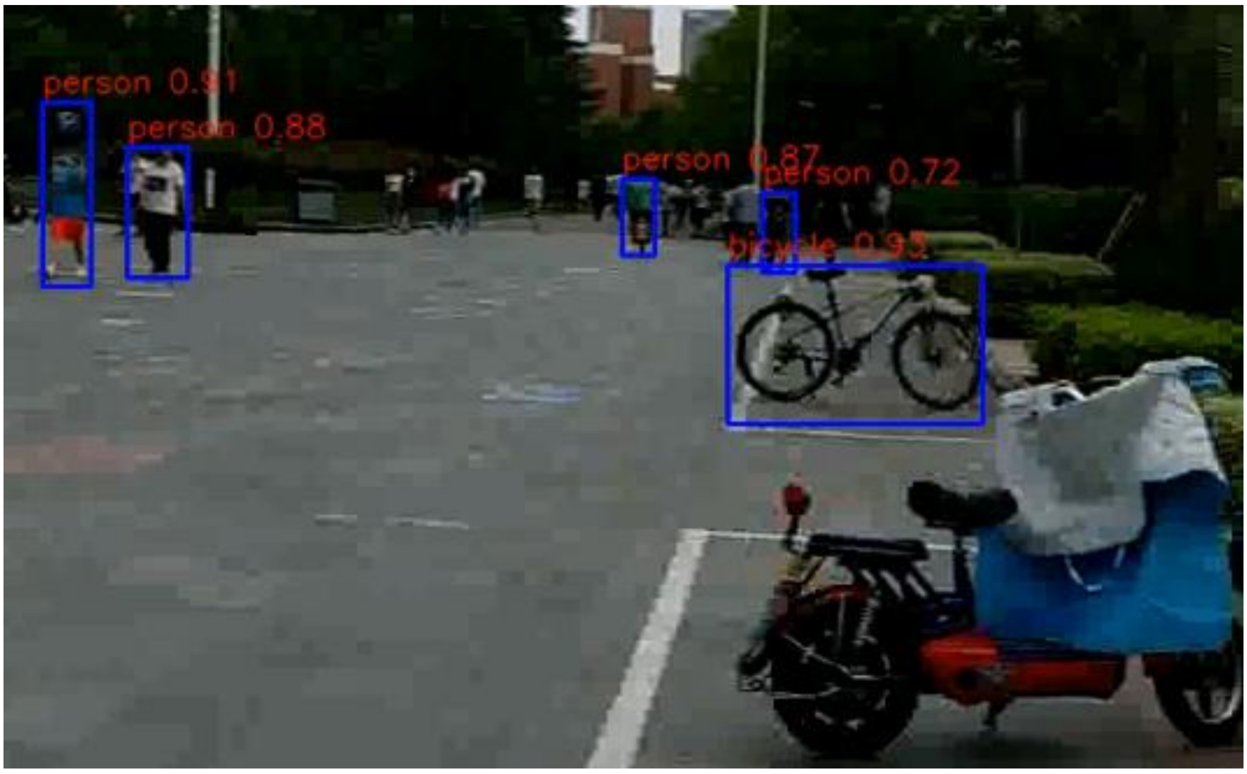
(Sample Image stored in images/test folder)

Expected Output:

The Output will be given to the user in the form of the download link, from where he can download or save the video after the object detection done on the video.



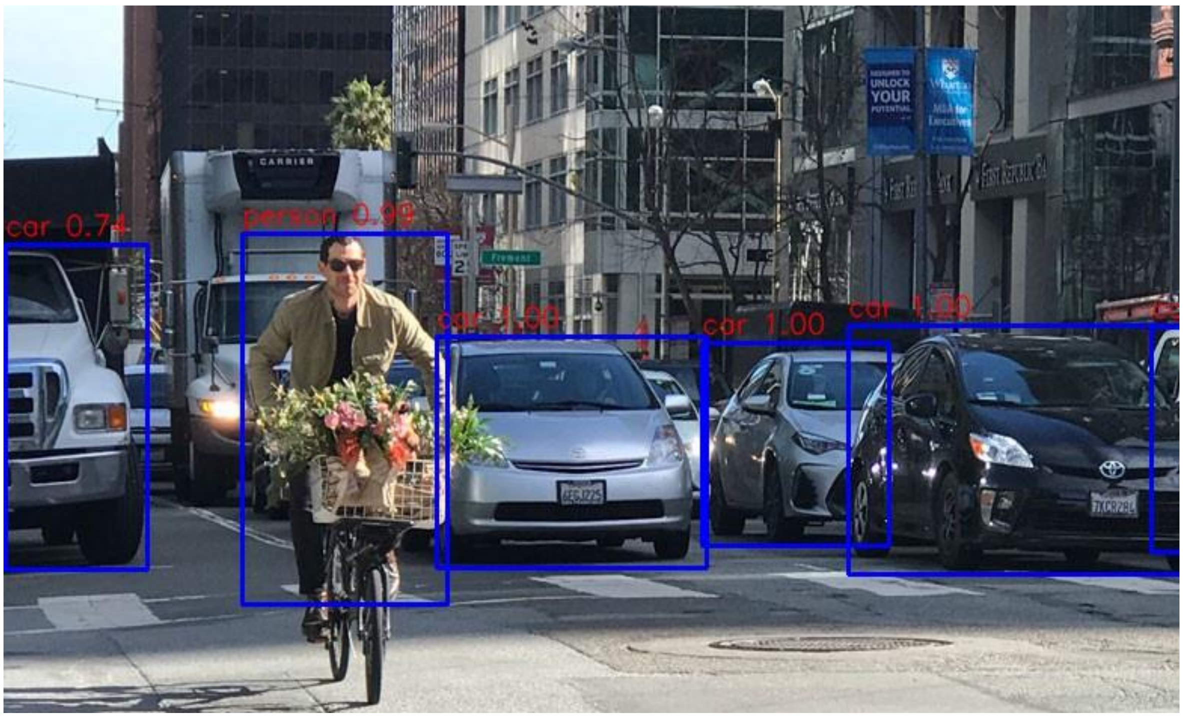
(HTML page to download a file and a file downloaded is shown i.e. 'library1.mp4')



(Snapshot of video file after object detection)

For an Image the output will be given by storing an image in the image/res folder with the detected objects marked in a box.

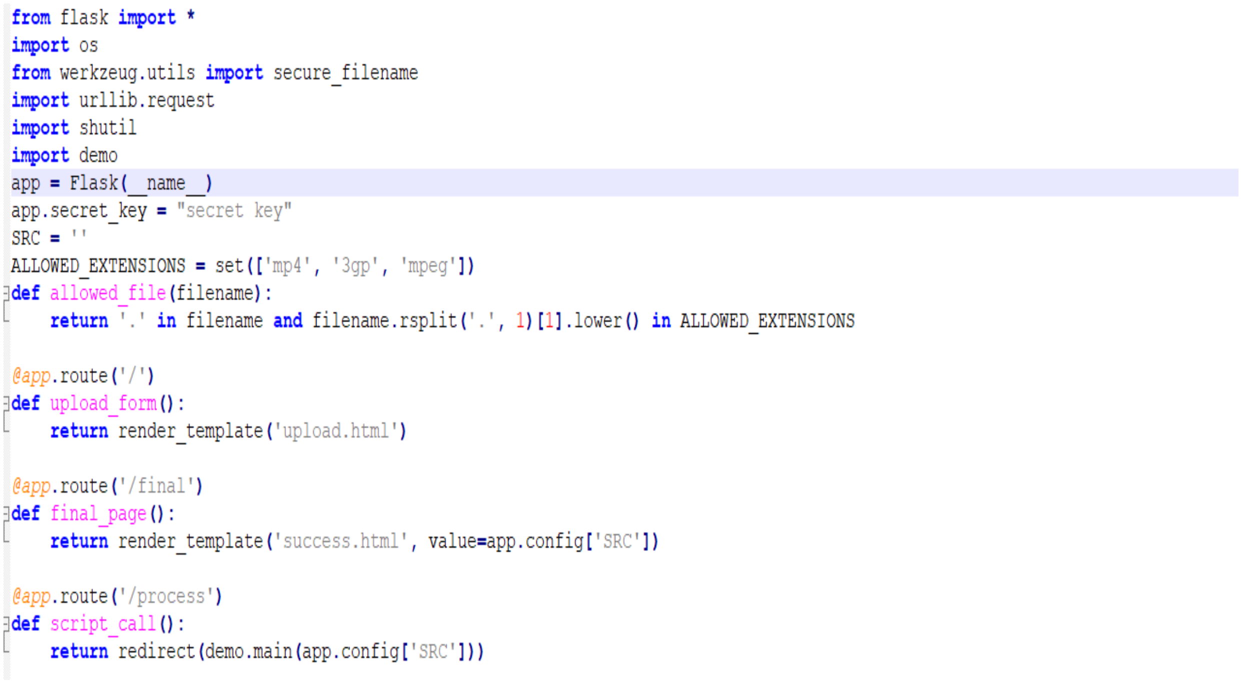
Example:



(Test image after Object Detection)

Flask File:

Flask File is made as an intermediate between the web page and the python file.

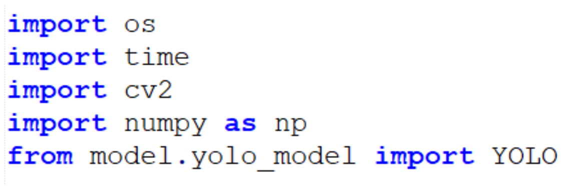


```
from flask import *
import os
from werkzeug.utils import secure_filename
import urllib.request
import shutil
import demo
app = Flask(__name__)
app.secret_key = "secret key"
SRC = ''
ALLOWED_EXTENSIONS = set(['mp4', '3gp', 'mpeg'])
def allowed_file(filename):
    return '.' in filename and filename.rsplit('.', 1)[1].lower() in ALLOWED_EXTENSIONS

@app.route('/')
def upload_form():
    return render_template('upload.html')

@app.route('/final')
def final_page():
    return render_template('success.html', value=app.config['SRC'])

@app.route('/process')
def script_call():
    return redirect(demo.main(app.config['SRC']))
```

(Flask File Snapshot)

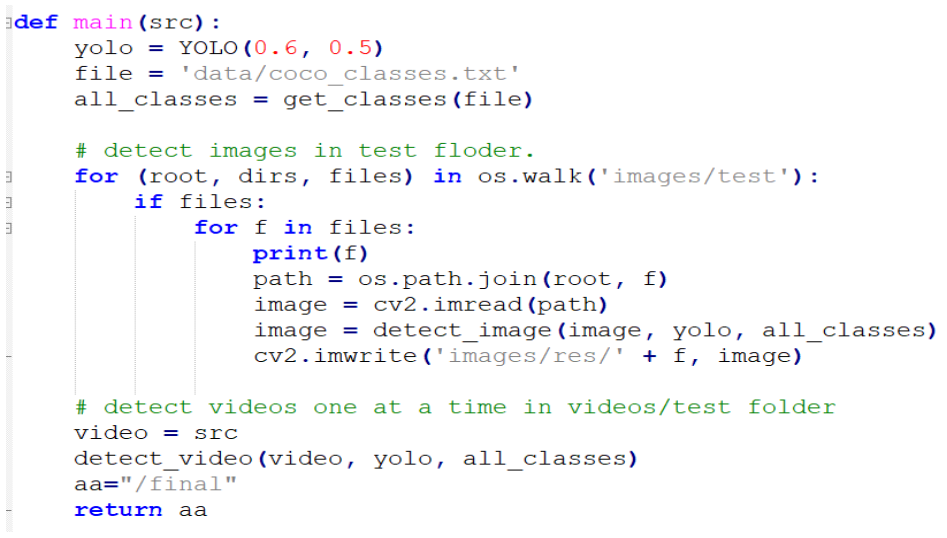
Testing Model (using OpenCV):

Import libraries:



```python
import os
import time
import cv2
import numpy as np
from model.yolo_model import YOLO
```

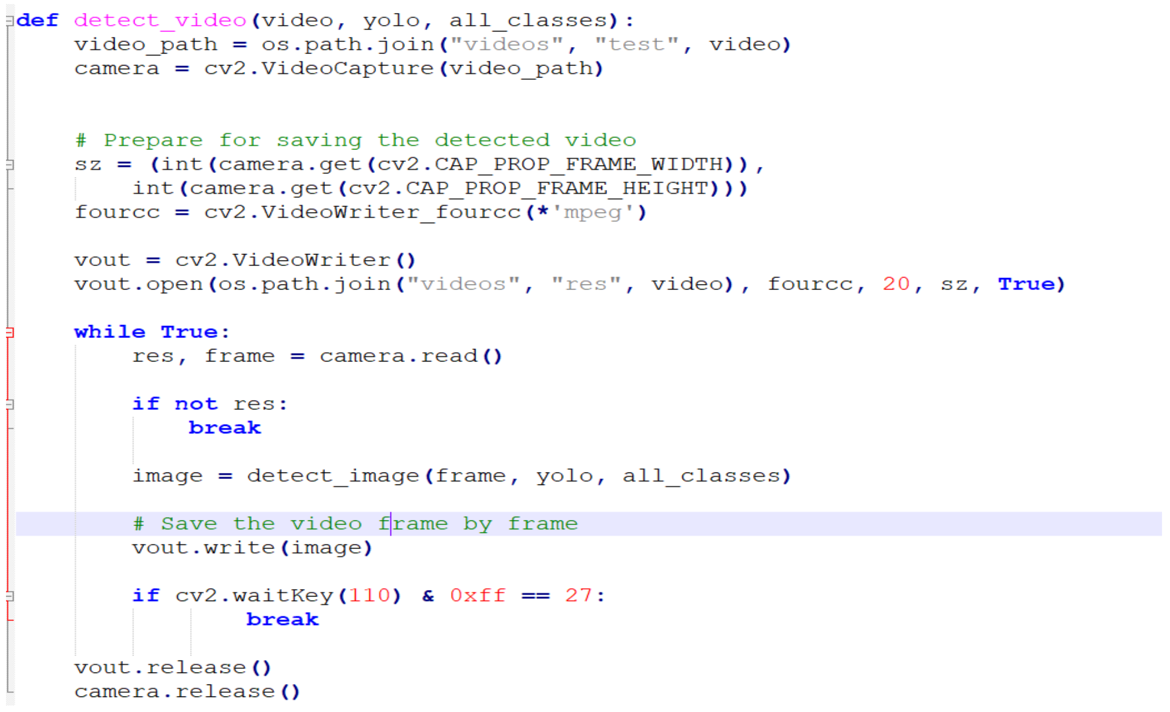
Getting file uploaded by user and the images in the images/test Folder:



```python
def main(src):
    yolo = YOLO(0.6, 0.5)
    file = 'data/coco_classes.txt'
    all_classes = get_classes(file)

    # detect images in test floder.
    for (root, dirs, files) in os.walk('images/test'):
        if files:
            for f in files:
                print(f)
                path = os.path.join(root, f)
                image = cv2.imread(path)
                image = detect_image(image, yolo, all_classes)
                cv2.imwrite('images/res/' + f, image)

    # detect videos one at a time in videos/test folder
    video = src
    detect_video(video, yolo, all_classes)
    aa="/final"
    return aa
```

Detecting the video:

In this the video is divided into the frames and those frames are used to detect the object and after that the video is saved frame by frame one after another.



```python
def detect_video(video, yolo, all_classes):
    video_path = os.path.join("videos", "test", video)
    camera = cv2.VideoCapture(video_path)


    # Prepare for saving the detected video
    sz = (int(camera.get(cv2.CAP_PROP_FRAME_WIDTH)),
        int(camera.get(cv2.CAP_PROP_FRAME_HEIGHT)))
    fourcc = cv2.VideoWriter_fourcc(*'mpeg')

    vout = cv2.VideoWriter()
    vout.open(os.path.join("videos", "res", video), fourcc, 20, sz, True)

    while True:
        res, frame = camera.read()

        if not res:
            break

        image = detect_image(frame, yolo, all_classes)

        # Save the video frame by frame
        vout.write(image)

        if cv2.waitKey(110) & 0xff == 27:
                break

    vout.release()
    camera.release()
```

Processing the Image detected:

This function will resize the image into the standard image size used by us in our configuration file, i.e. *yolov3.cfg*

```python
def process_image(img):

    image = cv2.resize(img, (416, 416),
                       interpolation=cv2.INTER_CUBIC)
    image = np.array(image, dtype='float32')
    image /= 255.
    image = np.expand_dims(image, axis=0)

    return image
```

Detecting the Image and using the yolo.predict() to detect objects in the picture.

yolo.predict() returns the 3 parameters of type array:

boxes: contain the dimension of the box to be drawn

classes: to which class the object belongs to.

Score: score shows that how much the detected object belongs to the particular class. It is between 0 to 1 only.

```python
def detect_image(image, yolo, all_classes):

    pimage = process_image(image)

    start = time.time()
    boxes, classes, scores = yolo.predict(pimage, image.shape)
    end = time.time()

    print('time: {0:.2f}s'.format(end - start))

    if boxes is not None:
        draw(image, boxes, scores, classes, all_classes)

    return image
```

Drawing Boxes:

Rectangles are drawn around the object that is detected and the class and score is defined around the rectangle.

```python
def draw(image, boxes, scores, classes, all_classes):

    for box, score, cl in zip(boxes, scores, classes):
        x, y, w, h = box

        top = max(0, np.floor(x + 0.5).astype(int))
        left = max(0, np.floor(y + 0.5).astype(int))
        right = min(image.shape[1], np.floor(x + w + 0.5).astype(int))
        bottom = min(image.shape[0], np.floor(y + h + 0.5).astype(int))

        cv2.rectangle(image, (top, left), (right, bottom), (255, 0, 0), 2)
        cv2.putText(image, '{0} {1:.2f}'.format(all_classes[cl], score),
                    (top, left - 6),
                    cv2.FONT_HERSHEY_SIMPLEX,
                    0.6, (0, 0, 255), 1,
                    cv2.LINE_AA)

        print('class: {0}, score: {1:.2f}'.format(all_classes[cl], score))
        print('box coordinate x,y,w,h: {0}'.format(box))

    print()
```

6. CONCLUSION

This project is made using the YOLOv3 Object detection. The Object detection is done in both the images and the video as well. The videos are divided frame by frame and then each frame is tested as an image and then again it is saved as frame by frame to make a video. This project is useful in many real-life problems such as; for the self-driving cars to detect the objects in front of it. The result can be seen in the form of image for the image output and an '*mpeg*' formatted video for a video input. The result will include the boxes around the objects and the name of the class they belong to i.e. person, bicycle, car etc. and the confidence that show confident is our model that the object belongs to the particular class. The confidence lies between 0 to 1.

7. REFRENCES

- https://medium.com/@whatdhack/reflections-on-non-maximum-suppression-nms-d2fce148ef0a#:~:text=Non%20Maximum%20Suppression%20(NMS)%20is,out%20of%20many%20overlapping%20entities.&text=Most%20object%20detection%20algorithms%20use,detected%20rectangles%20to%20a%20few.
- https://github.com/pjreddie/darknet
- https://devopedia.org/imagenet
- https://in.mathworks.com/discovery/object-detection.html
- https://github.com/allanzelener/YAD2K
- https://medium.com/analytics-vidhya/everything-you-need-to-know-to-train-your-custom-object-detector-model-using-yolov3-1bf0640b0905
- https://pypi.org/project/tensorflow-gpu/
- https://opencv.org/
- https://pjreddie.com/media/files/papers/YOLOv3.pdf
- https://pjreddie.com/darknet/yolo/
- https://pypi.org/project/Flask/
- https://flask.palletsprojects.com/en/1.1.x/