Object Detection System

END TERM REPORT

by

NAME OF THE CANDIDATE(S)

Yash Patel	**<11802515>**
Ashwani Kumar	**<11802585>**
Navdeep Upadhyay	**<11802595>**
Reshma	**<11802565>**

(Section: K18HV)
(Roll Number(s): 31,39,29,38)

Department of Intelligent Systems
School of Computer Science Engineering
Lovely Professional University, Jalandhar

Student Declaration

This is to declare that this report has been written by me. No part of the report is copied from other sources. All information included from other sources have been duly acknowledged. I/We aver that if any part of the report is found to be copied, I/we are shall take full responsibility for it.

Yash Patel
Roll number: 31

Ashwani Kumar
Roll number: 39

Navdeep Upadhyay
Roll number: 29

Reshma
Roll number: 38

BONAFIDE CERTIFICATE

Certified that this project report “Object Detection System” is the bonafide work of “Yash Patel, Ashwani Kumar, Navdeep Upadhyay, Reshma” who carried out the project work under my supervision.

Dipen Saini

Faculty

23681

Division of Admissions

TABLE OF CONTENTS

TITLE- Object Detection System **PAGE NO.**

Introduction

Object Detection System

Object identification and recognition is an area within the field of artificial intelligence(AI) that focuses on robots recognizing different objects. As AI machines become more and more part of our everyday lives, machine learning is making improvements on image tagging and object identification skills. Google and Microsoft are examples of just two tech giants investing in object identification tech.

Computer Vision Basics

Computer vision is the science of computer systems recognizing and analyzing different images and scenes. A key component of computer vision is object detection. Object detection is used to perform a number of AI tasks such as facial recognition, vehicle detection, security scanning, and self-driving. A number of algorithms depend on object detection to work successfully in AI implementations. The most well-documented algorithms tied to object detection processes include R-CNN, Fast RCNN, and Faster RCNN. The CNN in the algorithm stands for convolutional neural networks. CNN is designed to focus on pixels within images located next to each other. Each image is passed through the network as an input and then sent back as an output with each object classified.

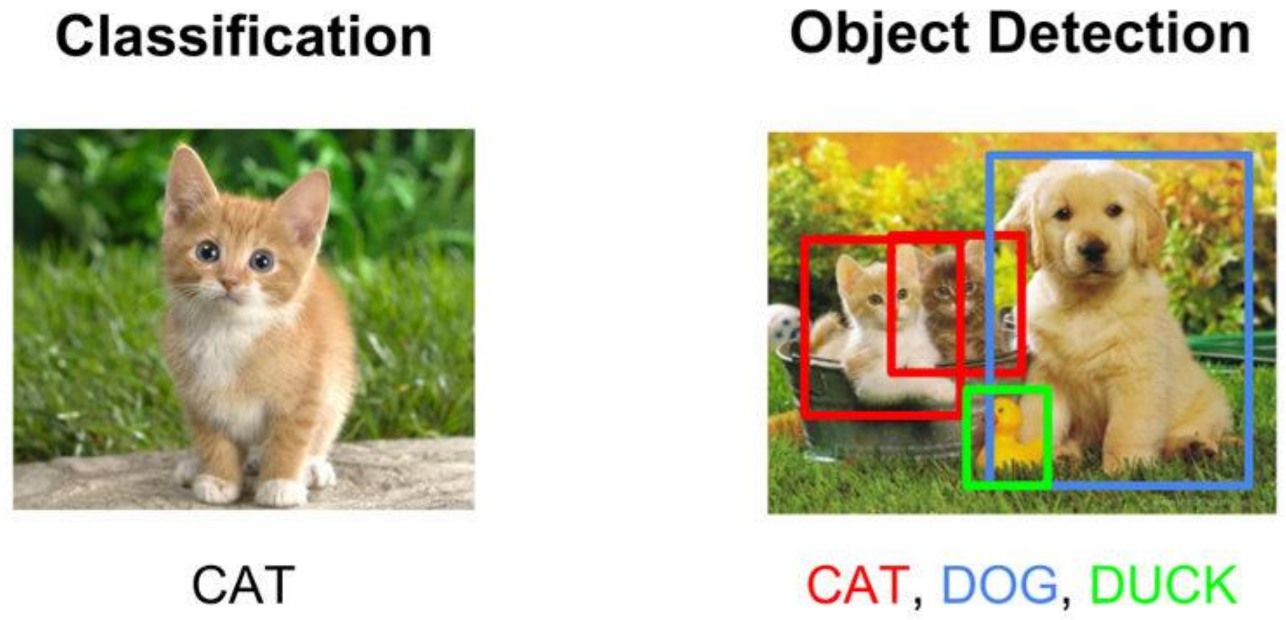
Recently, a number of advancements in deep learning have paved the way for solving object detection problems. MIT has funded a number of object detection projects including the development of neural networks based on how the human brain functions. Deep neural networks have shown a high-level of performance for successfully completing object classification tasks. MIT also created a deep learning system that allowed for object identification to occur in real-time through speech recognition.

Computer Vision Tasks

We focus on two main computer vision tasks — image classification and object detection.

1. **Image Classification** focuses on grouping an image into a predefined category. To achieve this, we need to have multiple images with the class that is of interest to us and train a computer to essentially convert pixel numbers to symbols. This is just saying that the computer sees a photo of a cat and says that there is a cat in it.

2. **Object detection** utilizes an image classifier to figure out what is present in an image and where. These tasks have been made easier through the use of Convolutional Neural Networks (CNNs) which have made it possible to detect multiple classes in a single pass of the image.



Object Detection Challenges

A challenge within AI systems is to employ accurate object detection when in contact with fast-moving objects like vehicles. In July of 2018, researchers from the University of California created a 3-D printed neural network with the capacity to use light photons to rapidly process images and recognize certain patterns to enable object identification. The AI network had over 91 percent accuracy for object detection with the research team continuously working to improve performance.

Another challenge for AI is that studies are demonstrating fallacies within current systems. For instance, Auburn University released a November 2018 research paper that Inception, Google's image-recognition system could be tricked by object rotations. As an example, a school bus was placed on its side by the research team with Inception incorrectly identifying it as a snowplow. The point of the study was to show Inception could be easily confused with out of the norm placement of objects.

Improvements in object detection are critical to the future of AI. If robots can understand their surroundings better, they are able to perform complex tasks.

Software Requirements :-

Installing dependencies

Following things are needed to execute the code we will be writing.

- Python 3

- Numpy

  ```
  pip install numpy
  ```

- OpenCV Python bindings

  ```
  pip install opencv-python
  ```

YOLOv3

YOLOv3 is the latest variant of a popular object detection algorithm **YOLO – You Only Look Once**. The published model recognizes 80 different objects in images and videos, but most importantly it is super fast and nearly as accurate as Single Shot MultiBox (SSD).

Starting with OpenCV 3.4.2, we can easily use YOLOv3 models in our own OpenCV application..

How does YOLO work ?

We can think of an object detector as a combination of a **object locator** and an **object recognizer**.

In traditional computer vision approaches, a sliding window was used to look for objects at different locations and scales. Because this was such an expensive operation, the aspect ratio of the object was usually assumed to be fixed.

Early Deep Learning based object detection algorithms like the R-CNN and Fast R-CNN used a method called Selective Search to narrow down the number of bounding boxes that the algorithm had to test.

Another approach called Overfeat involved scanning the image at multiple scales using sliding windows-like mechanisms done convolutionally.

This was followed by Faster R-CNN that used a Region Proposal Network (RPN) for identifying bounding boxes that needed to be tested. By clever design the features extracted for recognizing objects, were also used by the RPN for proposing potential bounding boxes thus saving a lot of computation.

YOLO on the other hand approaches the object detection problem in a completely different way. It forwards the whole image only once through the network. SSD is another object detection algorithm that forwards the image once though a deep learning network, but YOLOv3 is much faster than SSD while achieving very comparable accuracy. YOLOv3 gives faster than realtime results on a M40, TitanX or 1080 Ti GPUs.

Lets see how YOLO detects the objects in a given image.

First, it divides the image into a 13×13 grid of cells. The size of these 169 cells vary depending on the size of the input. For a 416×416 input size that we used in our experiments, the cell size was 32×32. Each cell is then responsible for predicting a number of boxes in the image.

For each bounding box, the network also predicts the confidence that the bounding box actually encloses an object, and the probability of the enclosed object being a particular class.

Most of these bounding boxes are eliminated because their confidence is low or because they are enclosing the same object as another bounding box with very high confidence score. This technique is called **non-maximum suppression**.

The authors of YOLOv3, Joseph Redmon and Ali Farhadi, have made YOLOv3 faster and more accurate than their previous work YOLOv2. YOLOv3 handles multiple scales better. They have also improved the network by making it bigger and taking it towards residual networks by adding shortcut connections.

Why use OpenCV for YOLO ?

Here are a few reasons we want to use OpenCV for YOLO

1. **Easy integration with an OpenCV application**: If our application already uses OpenCV and we simply want to use YOLOv3, we don’t have to worry about compiling and building the extra Darknet code.

2. **OpenCV CPU version is 9x faster**: OpenCV’s CPU implementation of the DNN module is astonishingly fast. For example, Darknet when used with OpenMP takes about 2 seconds on a CPU for inference on a single image. In contrast, OpenCV’s implementation runs in a mere 0.22 seconds! Check out table below.
3. **Python support**: Darknet is written in C, and it does not officially support Python. In contrast, OpenCV does. There are python ports available for Darknet though.

Steps Taken for the Project:-

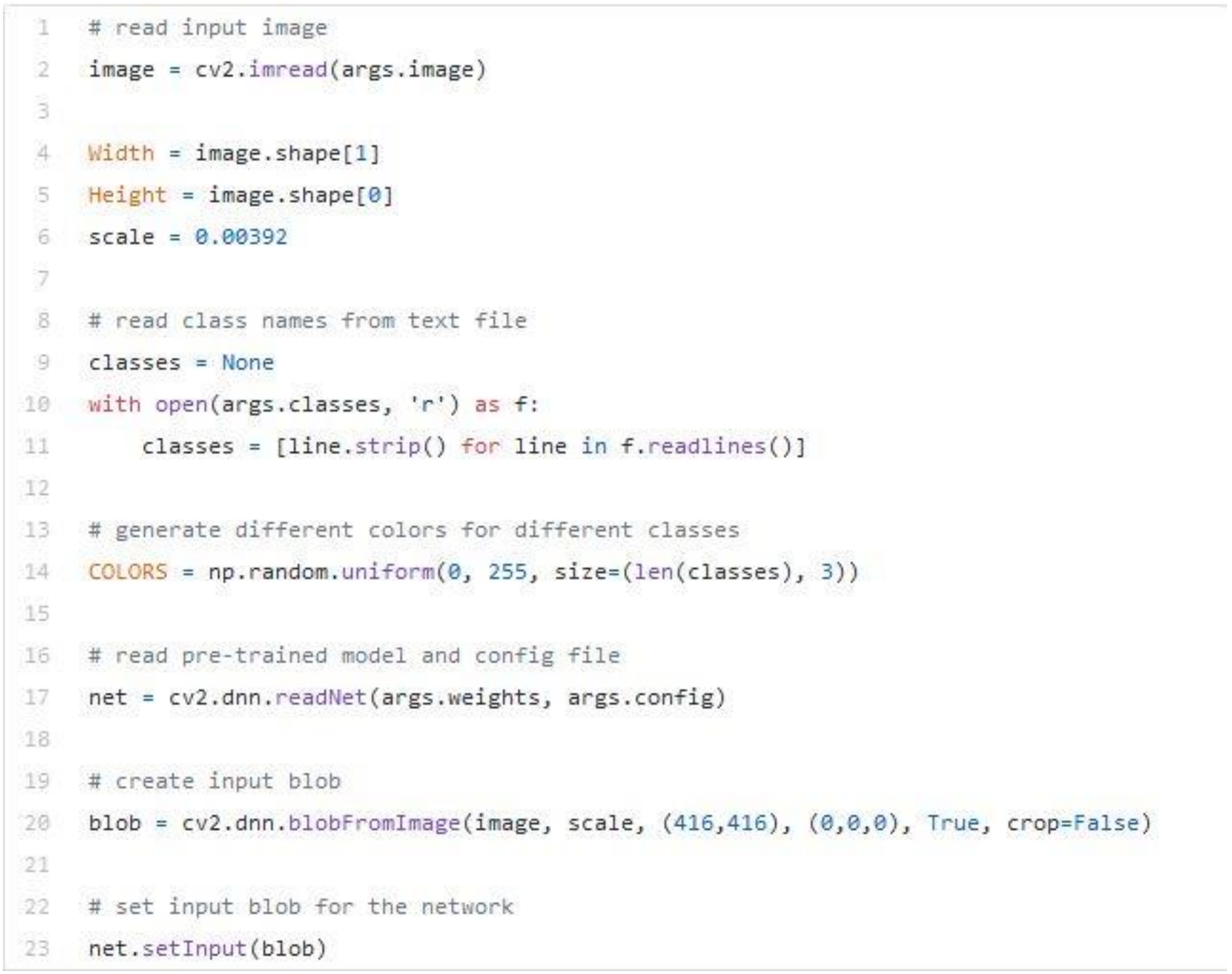
The script requires four input arguments.

- input image
- YOLO config file
- pre-trained YOLO weights
- text file containing class names

  The YOLO config file and text file can be found on our github while the pre-trained model can be downloaded from the following link:-

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

Preparing Inputs-



```python
# read input image
image = cv2.imread(args.image)

Width = image.shape[1]
Height = image.shape[0]
scale = 0.00392

# read class names from text file
classes = None
with open(args.classes, 'r') as f:
    classes = [line.strip() for line in f.readlines()]

# generate different colors for different classes
COLORS = np.random.uniform(0, 255, size=(len(classes), 3))

# read pre-trained model and config file
net = cv2.dnn.readNet(args.weights, args.config)

# create input blob
blob = cv2.dnn.blobFromImage(image, scale, (416,416), (0,0,0), True, crop=False)

# set input blob for the network
net.setInput(blob)
```

Read the input image and get its width and height.

Read the text file containing class names in human readable form and extract the class names to a list.

Generate different colors for different classes to draw bounding boxes.

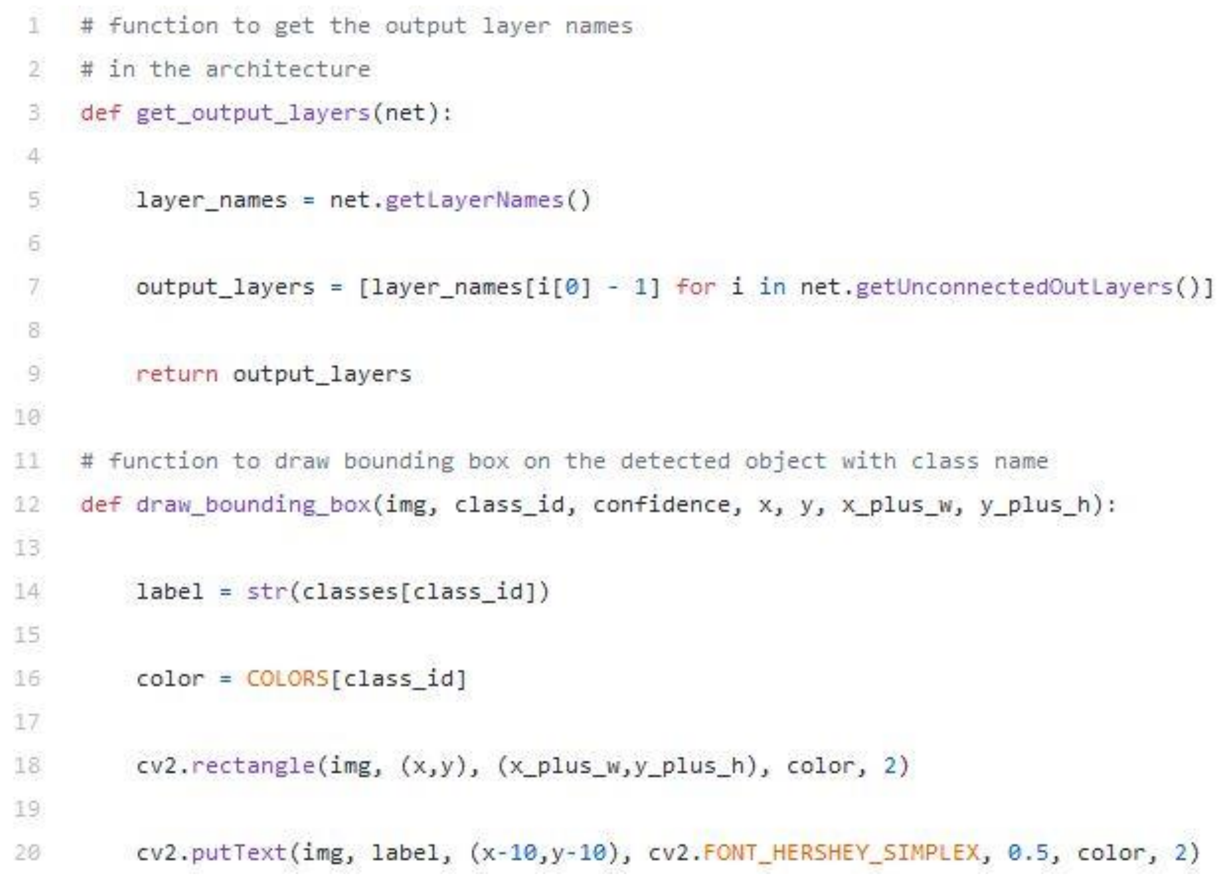
```
net = cv2.dnn.readNet(args.weights, args.config)
```

Above line reads the weights and config file and creates the network.

```
blob = cv2.dnn.blobFromImage(image, scale, (Width,Height), (0,0,0), True, crop=False)

net.setInput(blob)
```

Above lines prepares the input image to run through the deep neural network.

Output layer And Bounding Box:

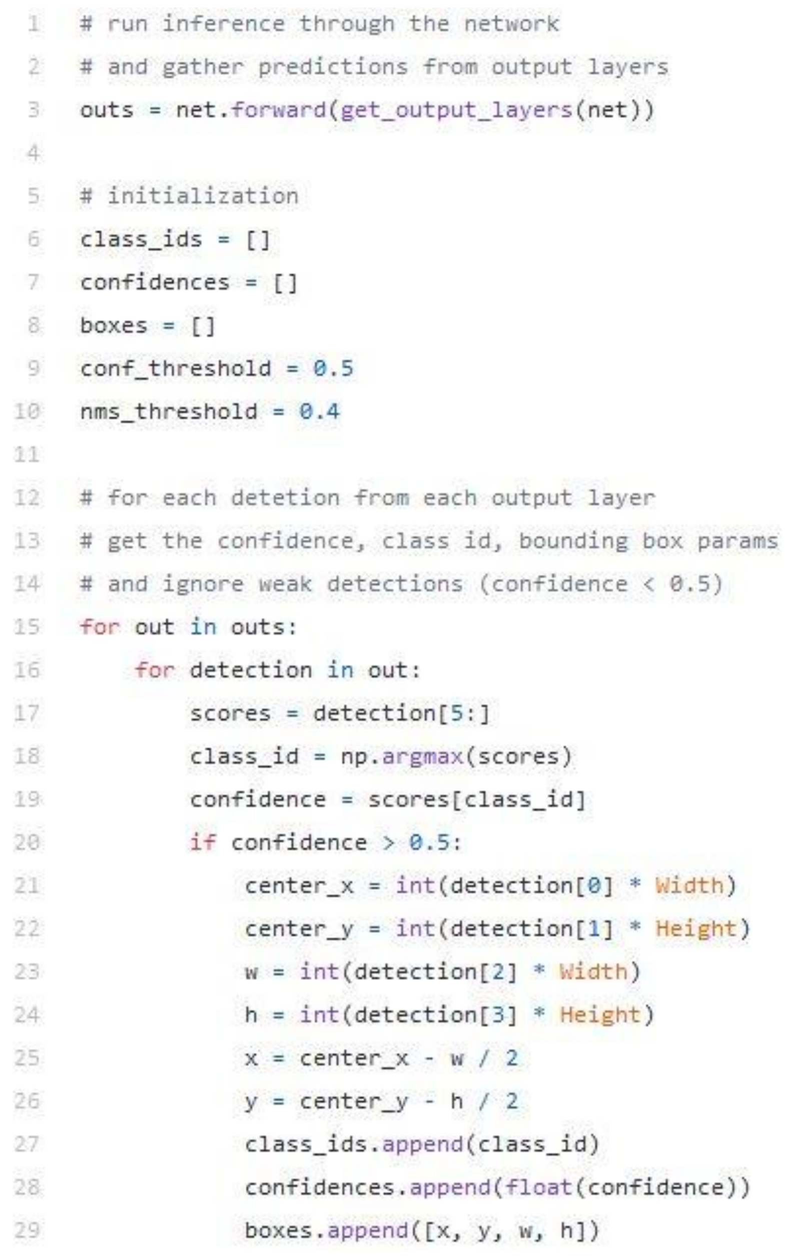


```
# function to get the output layer names
# in the architecture
def get_output_layers(net):

    layer_names = net.getLayerNames()

    output_layers = [layer_names[i[0] - 1] for i in net.getUnconnectedOutLayers()]

    return output_layers

# function to draw bounding box on the detected object with class name
def draw_bounding_box(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)
```

Generally in a sequential CNN network there will be only one output layer at the end. In the YOLO v3 architecture we are using there are multiple output layers giving out predictions. `get_output_layers()` function gives the names of the output layers. An output layer is not connected to any next layer.

`draw_bounding_box()` function draws rectangle over the given predicted region and writes class name over the box. If needed, we can write the confidence value too.

Running Interface:



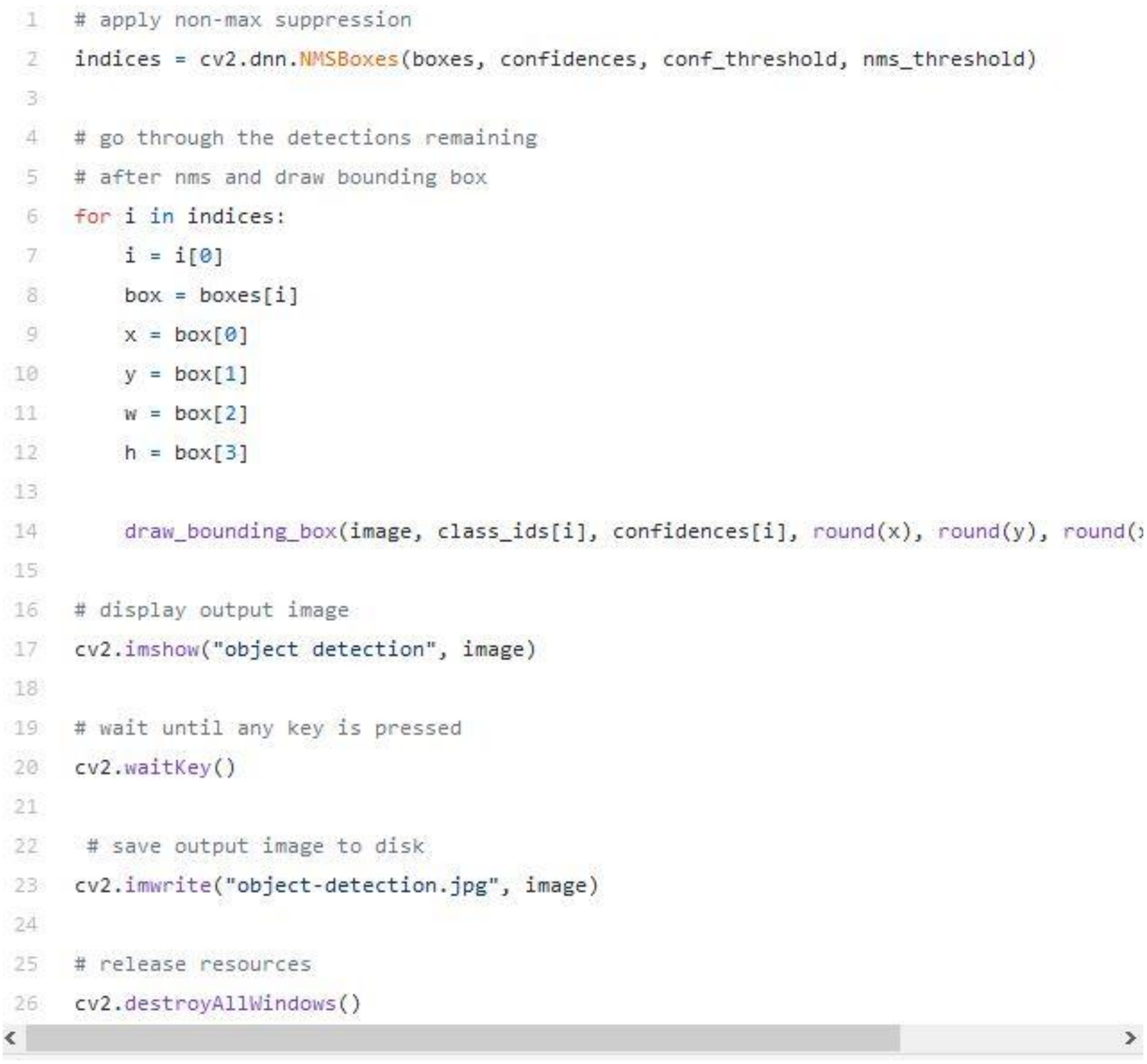
```python
# run inference through the network
# and gather predictions from output layers
outs = net.forward(get_output_layers(net))

# initialization
class_ids = []
confidences = []
boxes = []
conf_threshold = 0.5
nms_threshold = 0.4

# for each detetion from each output layer
# get the confidence, class id, bounding box params
# and ignore weak detections (confidence < 0.5)
for out in outs:
    for detection in out:
        scores = detection[5:]
        class_id = np.argmax(scores)
        confidence = scores[class_id]
        if confidence > 0.5:
            center_x = int(detection[0] * Width)
            center_y = int(detection[1] * Height)
            w = int(detection[2] * Width)
            h = int(detection[3] * Height)
            x = center_x - w / 2
            y = center_y - h / 2
            class_ids.append(class_id)
            confidences.append(float(confidence))
            boxes.append([x, y, w, h])
```

```
outs = net.forward(get_output_layers(net))
```

Above line is where the exact feed forward through the network happens. Moment of truth. If we don’t specify the output layer names, by default, it will return the predictions only from final output layer. Any intermediate output layer will be ignored.

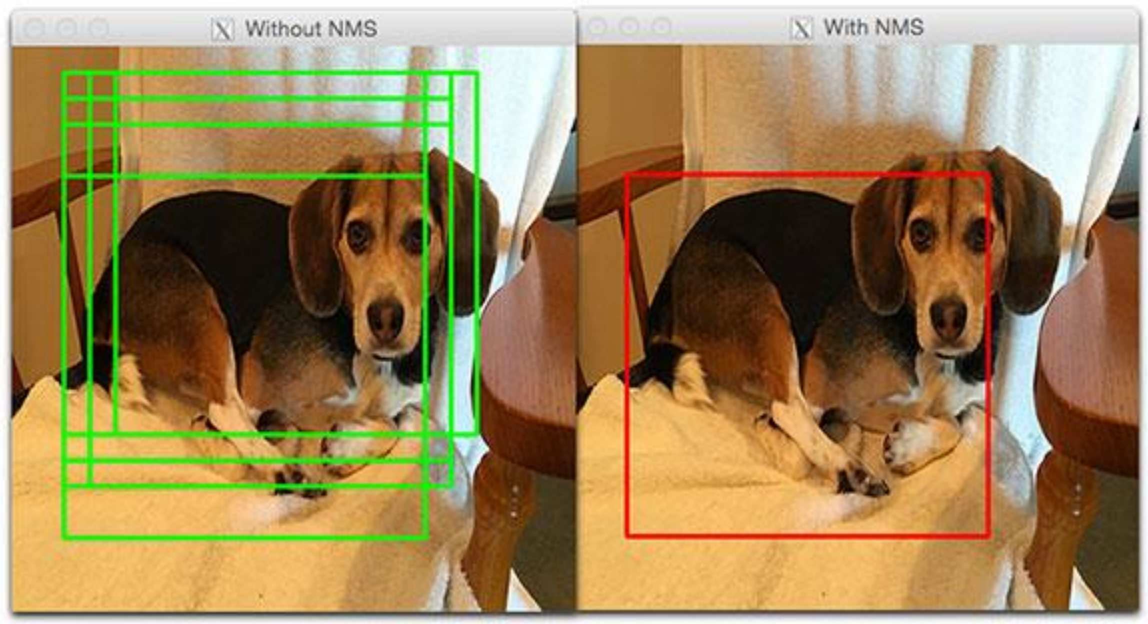
We need go through each detection from each output layer to get the class id, confidence and bounding box corners and more importantly ignore the weak detections (detections with low confidence value).

Non-Max Suppression:



```
# apply non-max suppression
indices = cv2.dnn.NMSBoxes(boxes, confidences, conf_threshold, nms_threshold)

# go through the detections remaining
# after nms and draw bounding box
for i in indices:
    i = i[0]
    box = boxes[i]
    x = box[0]
    y = box[1]
    w = box[2]
    h = box[3]

    draw_bounding_box(image, class_ids[i], confidences[i], round(x), round(y), round(

# display output image
cv2.imshow("object detection", image)

# wait until any key is pressed
cv2.waitKey()

 # save output image to disk
cv2.imwrite("object-detection.jpg", image)

# release resources
cv2.destroyAllWindows()
```

Even though we ignored weak detections, there will be lot of duplicate detections with overlapping bounding boxes. Non-max suppression removes boxes with high overlapping.



Finally we look at the detections that are left and draw bounding boxes around them and display the output image.

Steps to Run the Script:

Place all the files including YOLO config file, YOLO text file and the pre-trained model file in the same directory where the python script is saved. Now, open command line interface (Command Prompt) and type in the following command to run the script-

```
python yolo_opencv.py --image dog.jpg --config yolov3.cfg --
weights yolov3.weights --classes yolov3.txt
```

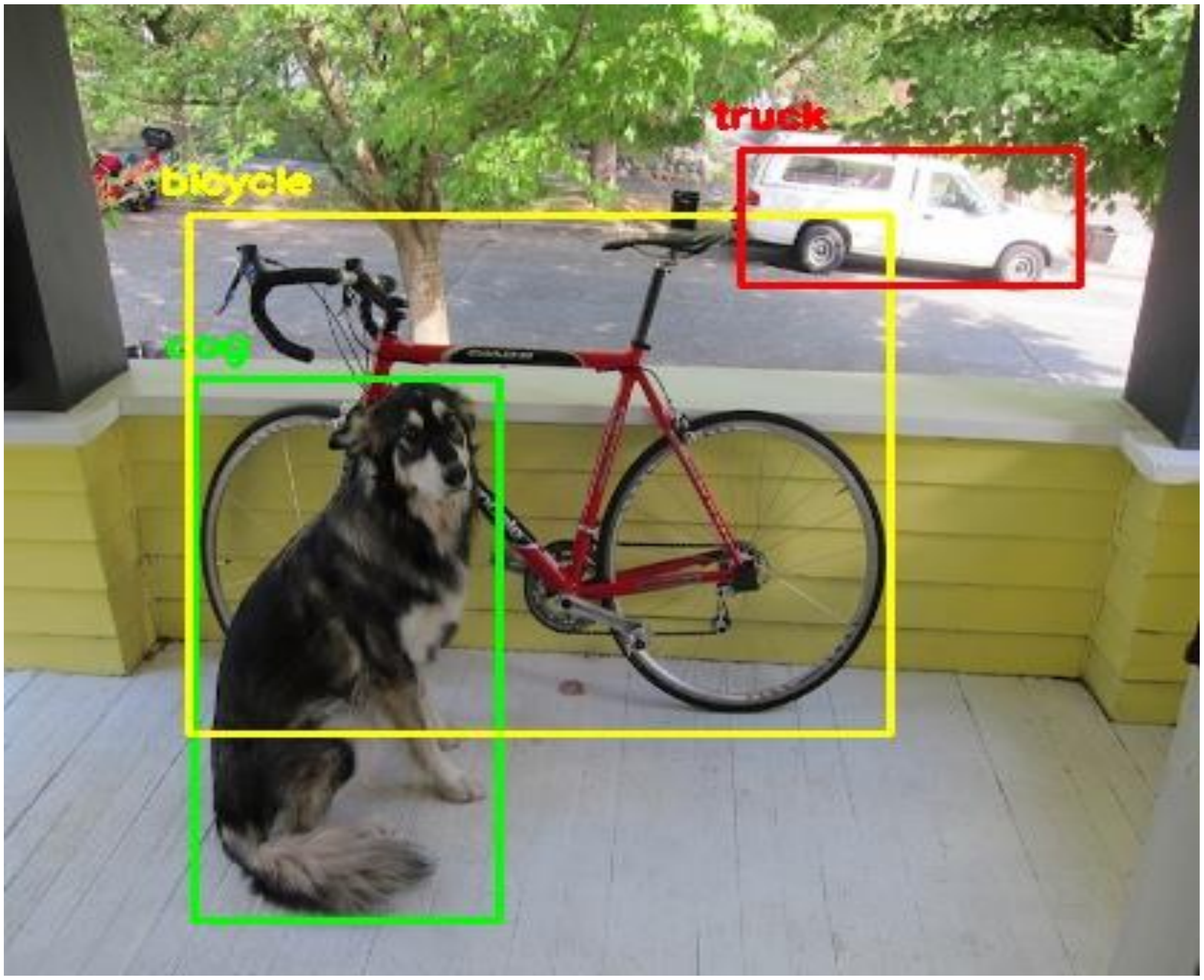
In the above command, dog.jpg is the name of the image file to be tested and this can be replaced with the name of any file required to be tested (But the file should not be very large).

Conclusion

Object detection is different from other computer vision tasks. We can use a pre-trained model and edit as needed to meet our needs. We'll prob need GCP or another platform that allows for higher computing power. Math is hard.

In the beginning, we found that the model was not able to predict many of the classes because many of them only had a few training images, which caused an imbalance training dataset.

Object detection is a very challenging topic, but from the help of various open sources online, like Coursera, YouTube instructional videos, GitHub, and Medium. All these as references helped to succeed in our amazing project.



Future Work — Improvements

Train the model on more classes to detect a greater variety of objects. To reach this goal, we need to first solve the problem of imbalanced data. A potential solution is that we can collect more images with these rarer classes.

a. Data Augmentation — Change existing images slightly to create new images

b. Image duplication — We can use the same image multiple times to train the algorithm on the specific rare class

c. Ensemble — Train one model on the popular classes and train another for the rare classes and use predictions from both.

2. In addition, we can try an ensemble of different models, such as MobileNet, VGG, etc. which are convolutional neural networks algorithms also used for object detection.