IBM PHASE 4 PROJECT

- Object Detection with YOLO

TEAM MEMBERS

- DHARANI G
- GOPIKA SHREE G
- ILAKIYA V
- KEERTHIGA DEVI P
- KARTHIKA
- KOWSALYA R

Object Detection with YOLO

What is YOLO exactly?
YOLO (You Only Look Once) is a method / way to do object detection. It is the algorithm /strategy behind how the code is going to detect objects in the image

Earlier detection frameworks, looked at different parts of the image multiple times at different scales and repurposed image classification technique to detect objects. This approach is slow and inefficient.

OpenCV dnn module

DNN (Deep Neural Network) module was initially part of opencv_contrib repo. It has been moved to the master branch of opencv repo last year, giving users the ability to run inference on pre-trained deep learning models within OpenCV itself.

```
# import required packages
import cv2
import argparse
import numpy as np

# handle command line arguments
ap = argparse.ArgumentParser()
ap.add_argument('-i', '—image', required=True,
        help = 'path to input image')
ap.add_argument('-c', '—config', required=True,
        help = 'path to yolo config file')
ap.add_argument('-w', '—weights', required=True,
        help = 'path to yolo pre-trained weights')
ap.add_argument('-cl', '—classes', required=True,
        help = 'path to text file containing class names')
args = ap.parse_args()
```

- **Installing dependencies**
- **Following things are needed to execute the code we will be writing.**
-
- **Python 3**
- **Numpy**
- **OpenCV Python bindings**

1.Python 3

If you are on Ubuntu, it's most likely that Python 3 is already installed. Run python3 in terminal to check whether its installed. If its not installed use

sudo apt-get install python3

2.Numpy

Pip install numpy
This should install numpy. Make sure pip is linked to Python 3.x (pip –V will show this info)
If needed use pip3. Use sudo apt-get install python3-pip to get pip3 if not already installed.

3.Opencv python binding

Adrian Rosebrock has written a good blog post on PyImageSearch on this. (Download the source from master branch instead of from archive)

If you are overwhelmed by the instructions to get OpenCV Python bindings from source, you can get the unofficial Python package using

pip install opencv-python

This is not maintained officially by OpenCV.org. It's a community maintained one.

Command line arguments

The script requires four input arguments.

Input image
YOLO config file
pre-trained YOLO weights
text file containing class names
All of these files are available on the github repository I have put together. (link to download pre-trained weights is available in readme.)

Input image can be of your choice. Sample input is available in the repo.

Run the script by typing

$ python yolo_opencv.py –image dog.jpg –config yolov3.cfg –weights yolov3.weights –classes yolov3.txt

Preparing input

```
# 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

Outer 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

```
# 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))

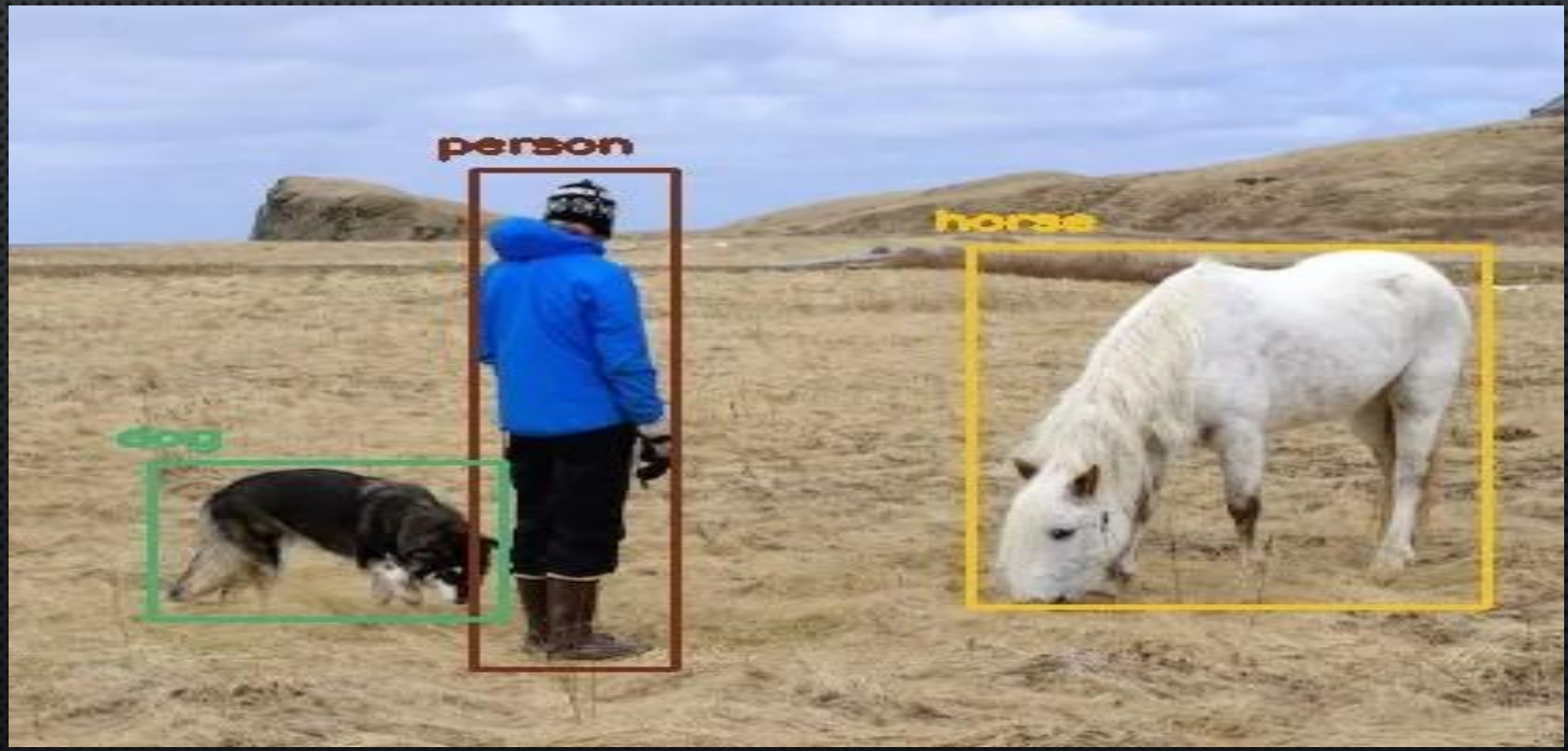
Above line is where the exact feed forward through the network happens. Moment of truth. If we don't specify the output layer names, by default, it will return the predictions only from final output layer. Any intermediate output layer will be ignored.

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(x+w), round(y+h))

# 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()
```

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



THANK YOU