DATS 6203
Junhe Zhang
Due: 12/8

Individual Final Report

Project Overview:

This project is about object detection on both image or video. The neural network structure that has been used is YOLO version 3 framework from darknet. The dataset that has been fed to the neural network was from Open Images Dataset V6 that included a total of 7 different objects of our interest which are car, bicycle wheel, bus, traffic light, jeans, laptop and person.

Project Outline:

This project could be divided into 3 sections:

1. Data Preprocessing
2. YOLO network training and model testing
3. Generate objects detected images or videos using trained model

Individual Work:

The sections to which I contributed are:

1. Data Preprocessing
   - Dataset download into EC2 using API
2. Network training and testing
   - Downloading darknet framework on EC2
   - Setting up YOLO.v3 on EC2
   - training model for 14000 iterations, 2000 for each class over 20 hours
3. Generate objects detected images or videos using trained model
   - Apply trained model and weights on images or videos taken from real life.

Individual Work Details:

Dataset download into EC2 using API:

In order to download a large dataset (over 3 GB) from an open source database into an EC2 instance, an API needs to be used. Command used to download is the following:

“python3 main.py downloader --classes Car Bicycle_wheel Bus Traffic_light Jeans Laptop Person --type_csv train -- multiclasses 1 --limit 1000”

Main.py came from the github repository that clone from OIDv4_ToolKit repository. Flag --classes tell which category of image we are interested in. Flag --type indicates we want a comma separated version (.csv) of training dataset bounding box information. Flag --limit indicates we want 1000 images for each category (some classes do not have enough images).

Network training and testing:

Darknet framework was also downloaded in our EC2 instance using git clone from https://github.com/AlexeyAB/darknet
After my group member Heng Hao preprocessed the image data and their corresponding bounding box .csv files into YOLO format data that can be used on the darknet detector. My group member Gong Ze set up the network structure in the custom_train.cfg file. I trained the network on approximately 7000 images using the following command:

‘./darknet detector train cfg/custom_data.data cfg/yolov3-custom_train.cfg weights/darknet53.com -dont_show’

Keyword detector train indicates this is a training command.
Custom_data.data is a configuration file that sets up the file directory for necessary folders like training image dataset directory, etc….
yolov3-custom_train.cfg is the network structure that we modified for your input data.
Darknet53.com is the pre-trained weights we used on our network to start with. So that the training time could be shortened.

Generate objects detected images or videos using trained model:

After the model has been trained it is time to test our model on real life object detection tasks. We used a couple pictures that were taken by our cell-phones. Then, upload them into EC2 instance and do object detection on each of them.

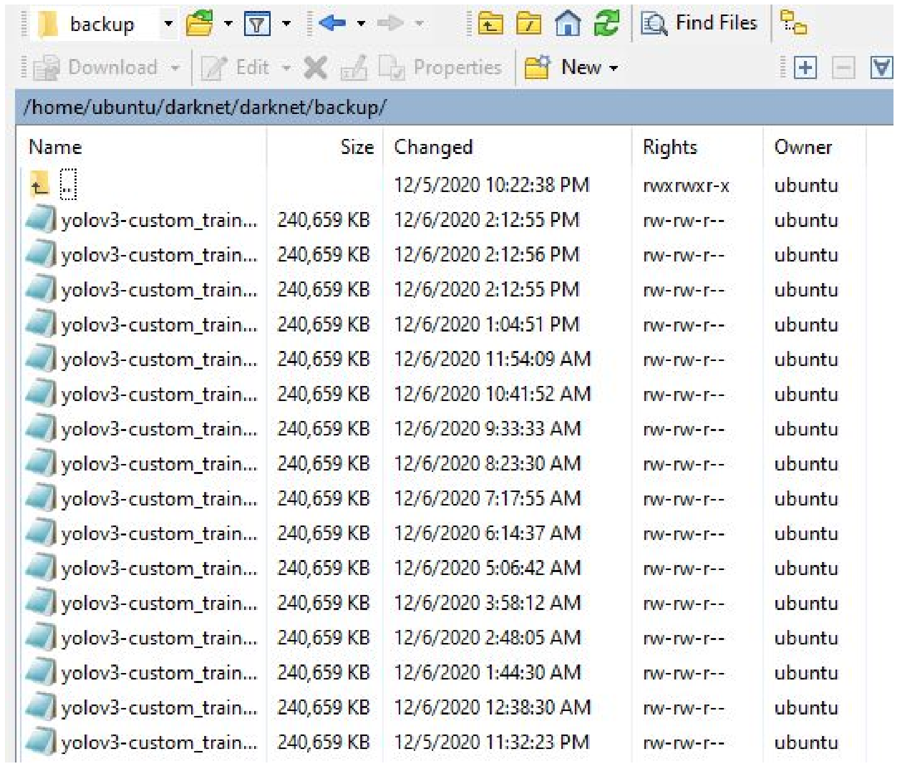
There are two ways to do object detection on the images using the trained model which are:

1. Using darknet framework’s command line
2. Run process_image_video.py file.

The first approach requires darknet framework setup properly on your machine and the command line is as following:

```
'./darknet detector test cfg/custom_data.data cfg/yolov3-custom_train.cfg
backup/yolov3-custom_train_last.weights data/laptop-jean-test.jpg -out_filename
data/result-laptop-jean-test.jpg -dont_show'
```

Keywords detector test indicates appling object detection. Yolov3-custom_train_last.weights is the weights we got from the last iteration of training. Weights were stored into backup fold in every 1000 iterations.



/home/ubuntu/darknet/darknet/backup/

Name	Size	Changed	Rights	Owner
..		12/5/2020 10:22:38 PM	rwxrwxr-x	ubuntu
yolov3-custom_train...	240,659 KB	12/6/2020 2:12:55 PM	rw-rw-r--	ubuntu
yolov3-custom_train...	240,659 KB	12/6/2020 2:12:56 PM	rw-rw-r--	ubuntu
yolov3-custom_train...	240,659 KB	12/6/2020 2:12:55 PM	rw-rw-r--	ubuntu
yolov3-custom_train...	240,659 KB	12/6/2020 1:04:51 PM	rw-rw-r--	ubuntu
yolov3-custom_train...	240,659 KB	12/6/2020 11:54:09 AM	rw-rw-r--	ubuntu
yolov3-custom_train...	240,659 KB	12/6/2020 10:41:52 AM	rw-rw-r--	ubuntu
yolov3-custom_train...	240,659 KB	12/6/2020 9:33:33 AM	rw-rw-r--	ubuntu
yolov3-custom_train...	240,659 KB	12/6/2020 8:23:30 AM	rw-rw-r--	ubuntu
yolov3-custom_train...	240,659 KB	12/6/2020 7:17:55 AM	rw-rw-r--	ubuntu
yolov3-custom_train...	240,659 KB	12/6/2020 6:14:37 AM	rw-rw-r--	ubuntu
yolov3-custom_train...	240,659 KB	12/6/2020 5:06:42 AM	rw-rw-r--	ubuntu
yolov3-custom_train...	240,659 KB	12/6/2020 3:58:12 AM	rw-rw-r--	ubuntu
yolov3-custom_train...	240,659 KB	12/6/2020 2:48:05 AM	rw-rw-r--	ubuntu
yolov3-custom_train...	240,659 KB	12/6/2020 1:44:30 AM	rw-rw-r--	ubuntu
yolov3-custom_train...	240,659 KB	12/6/2020 12:38:30 AM	rw-rw-r--	ubuntu
yolov3-custom_train...	240,659 KB	12/5/2020 11:32:23 PM	rw-rw-r--	ubuntu

The second approach is to run the process_image_video.py file. The algorithm is as following:

1. Read the input image and convert it into a blob object which is accepted by the YOLO framework.
2. Load the best trained weights and the network structure file into the YOLO framework.
3. Process blob image file into the model and generate output bounding boxes for each object inside the image.
4. Using Non-Maximun_Suppression technique on the output bounding boxes to drop duplicated boxes on the same object with lower confidence.

1. Reading images or videos using the openCV library. Need to switch the red and blue channel since openCV reads the image as BGR and converts it into a blob object with shape 416x416 to capture more details.

```
blob = cv2.dnn.blobFromImage(image_BGR, 1 / 255.0, (416, 416),
        swapRB=True, crop=False)
```

2. Loading model use openCV function:

```
cv2.dnn.readNetFromDarknet(cfg_dir, weights_dir)
```

3. Process blob image into loaded model and generate three outputs for each image since there are three output layers inside our YOLO network structures, which are layer 82, 94, 106.

4. Process all the bounding box outputs we get from the preview step and filter out the duplicated less confident bounding box on the same object using the Non-Maximun_Suppression technique with custom setup threshold. The equation behind Non-Maximun_Suppression technique is shown in figure 3

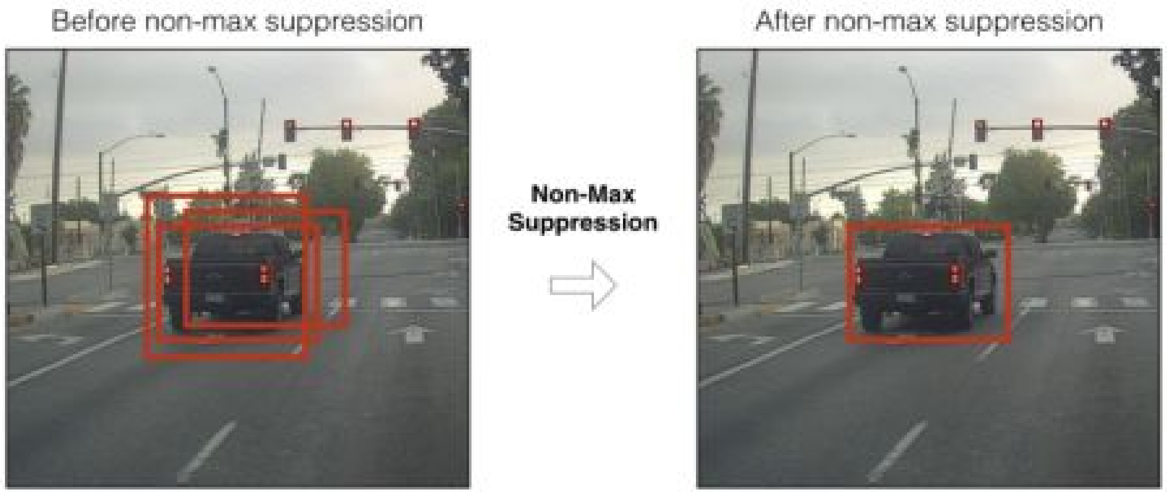


Figure 2. Before and After Non-Maximun_Suppression

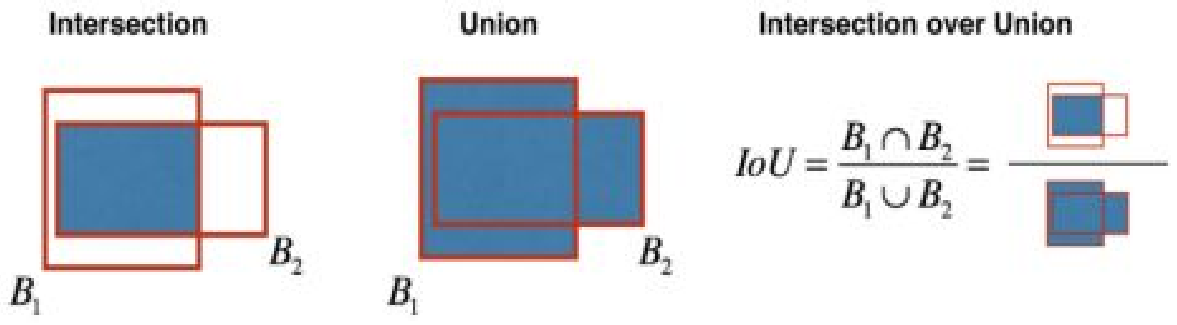
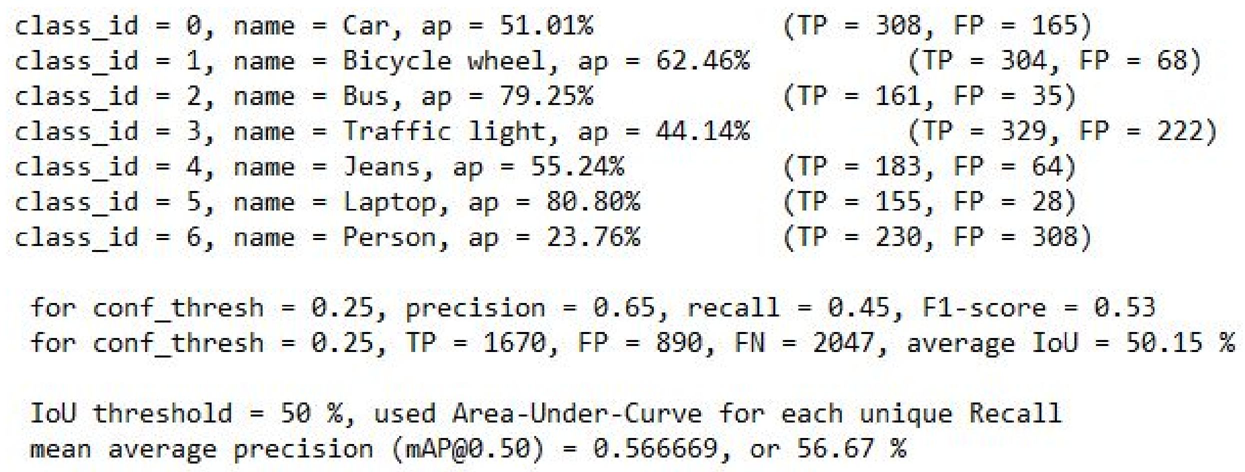


Figure 3. Non-Maximun_Suppression equation

Final Results:

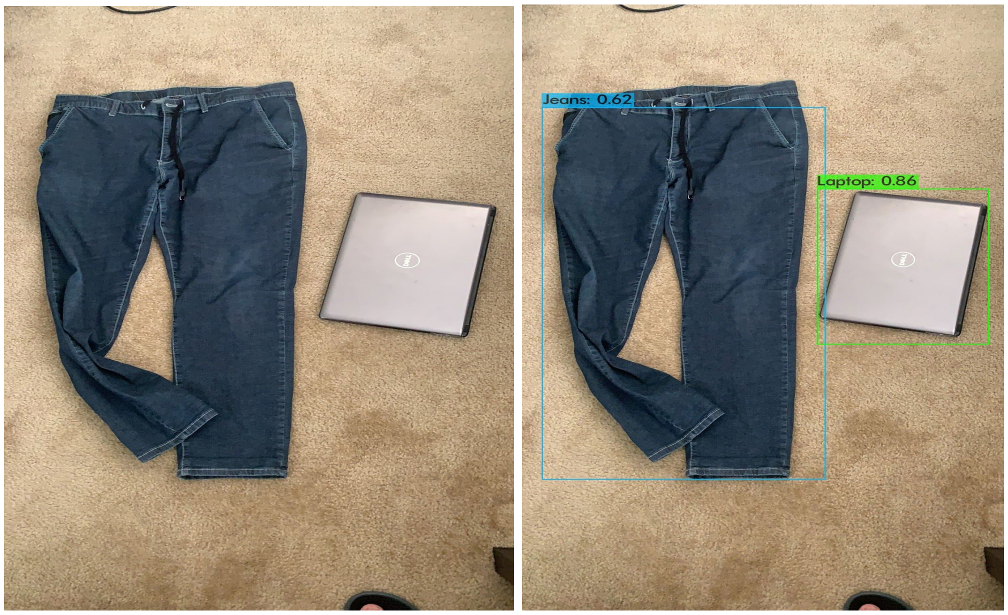
To evaluate our network, the darknet framework provides a convenient command line which generates log files using confusion matrix and Area-Under-Curve for each unique recall. The figure below shows the accuracy for each of seven classes. The worst classified class is person class which has accuracy as low as 23.76% with True Positive equals to 230 and False Positive equals to 308. The reason that might cause this problem could be modeled under training, since we did only the minimum recommended number of iterations for each class which is 2000. By increasing the training time, the result would be much better. Another possible issue is that some of our other classes' training images like Jeans and Laptop also contain people in it but did not get boxed around them. This will have a very bad effect on classifying Person class during training.



```
class_id = 0, name = Car, ap = 51.01%           (TP = 308, FP = 165)
class_id = 1, name = Bicycle wheel, ap = 62.46%         (TP = 304, FP = 68)
class_id = 2, name = Bus, ap = 79.25%           (TP = 161, FP = 35)
class_id = 3, name = Traffic light, ap = 44.14%         (TP = 329, FP = 222)
class_id = 4, name = Jeans, ap = 55.24%         (TP = 183, FP = 64)
class_id = 5, name = Laptop, ap = 80.80%        (TP = 155, FP = 28)
class_id = 6, name = Person, ap = 23.76%        (TP = 230, FP = 308)

 for conf_thresh = 0.25, precision = 0.65, recall = 0.45, F1-score = 0.53
 for conf_thresh = 0.25, TP = 1670, FP = 890, FN = 2047, average IoU = 50.15 %

 IoU threshold = 50 %, used Area-Under-Curve for each unique Recall
 mean average precision (mAP@0.50) = 0.566669, or 56.67 %
```



Conclusion:

Overall, our model can successfully detect all 7 of our interested classes both in image or video. Even though it performs badly at a couple of classes like People, there is still a great potential in this model to improve. In the future, we will plan to train on a bigger dataset and increase the number of iteration numbers.

Calculate the Percentage of the Code from Internet:

All source code getting from Valentyn Sichkar's udemy course [4]

Number of Unit in (lines)

Internet Code: 400

Discard Code from Internet: 100

Modified Code: 200

Added Code: 30

Final Result: (400-100-200)/(400-100+30) * 100 = 30.3%

References

[1]Joseph Redmon, Santosh Divvala, Ross Girshick, ALi Farhadi "You Only Look Once: Unified, Real-Time Object Detection", IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2016.

[2]Open Images Dataset V6 + Extensions. (n.d.). Retrieved December 08, 2020, from https://storage.googleapis.com/openimages/web/index.html

[3]Joseph R. (2013-2016). Darknet: Open Source Neural Networks in C. Retrieved from http://pjreddie.com/darknet/

[4]Sichkar, V. (2020, October 27). Training YOLO v3 for Objects Detection with Custom Data. Retrieved December 08, 2020, from https://www.udemy.com/course/training-yolo-v3-for-objects-detection-with-custom-data/