Final Report

Ze Gong

Introduction

In this project:

- We download the dataset from Google Open Image dataset with certain categories we choose to train the model and generate all needed files for training yolov3 model on darknet framework.
- We use the Darknet frame and pre-trained weight of the yolov3 model to train on our own dataset and evaluate the model.
- At last we will use our model on a video we recorded and an image we took to see the actual performance of the model.
- We shared most work by doing it altogether and my main part is training.

Background

• YOLO

YOLO stands for "You Only Look Once" and uses convolutional neural networks (CNN) for object detection. When YOLO works it predicts classes' labels and detects locations of objects at the same time. That is why, YOLO can detect multiple objects in one image. The name of the algorithm means that a single network just once is applied to whole image. YOLO divides image into regions, predicts bounding boxes and probabilities for every such region. YOLO also predicts confidence for every bounding box showing information that this particular bounding box actually includes object, and probability of included object in bounding box being a particular class. Then, bounding boxes are filtered with technique called non-maximum suppression that excludes some of them if confidence is low or there is another bounding box for this region with higher confidence.

• Software Implementation Frameworks

- Numpy
- Pandas
- OpenCV
- Darknet
- OS

Training YOLO in Darknet framework

1. Setting up the configuration

- To train YOLO, the first step is setting up the configuration. We have 7 classes to identify and the filters for coco dataset is 36. The max_batches is the total number of iterations and when it comes to steps, the learning rate becomes 10 percent of before respectively. Batch is the number of samples that will be processed in one batch. Subdivisions is the number of mini batches in one batch.

```
Classes = 7
filters = ( 7 + 5 ) x 3 = 36
max_batches = 7 x 2000 = 14000
steps = 0.8 x max_batches / 0.9 x max_batches
      = 11200 / 12600
batch = 32
subdivisions = 8
minibatch = batch/subdivisions = 4
```

- The minibatches here are determined by batch divided by subdivisions. GPU processes minibatch samples at once. The weights will be updated for batch samples, that is 1 iteration processes batch images.

2. Installing Darknet

- In most of the cases the general approach is as following or quite similar:

a. Install prerequisites (OpenCV, CUDA, cuDNN, etc.)
b. Clone repository
c. Adjust options in Makefile (for Linux & Mac)
d. Compile it (for Linux & Mac) or Build it (for Windows)

3. Preparing files for training

- In order to train on datasets, we need to copy data files (custom_data.data and ts_data.data) into the existing cfg folder inside the root Darknet directory.

4. Running training process

- To start training process in Darknet framework, navigate to the directory with executable file and type in Terminal or command line specific command as described below.

```
/darknet detector train cfg/custom_data.data cfg/yolov3_custom_train.cfg
weights/darknet53.conv.74
```

- Continue training with saved weights after 1000 iterations

It is possible to stop training, for example, after 1000 iteration and continue later by using already saved weights. In order to continue training just specify at the end of command location of needed weights to continue training from.

5. Find best weights to stop training

- Recommended number of iterations for training is defined by using following equation.

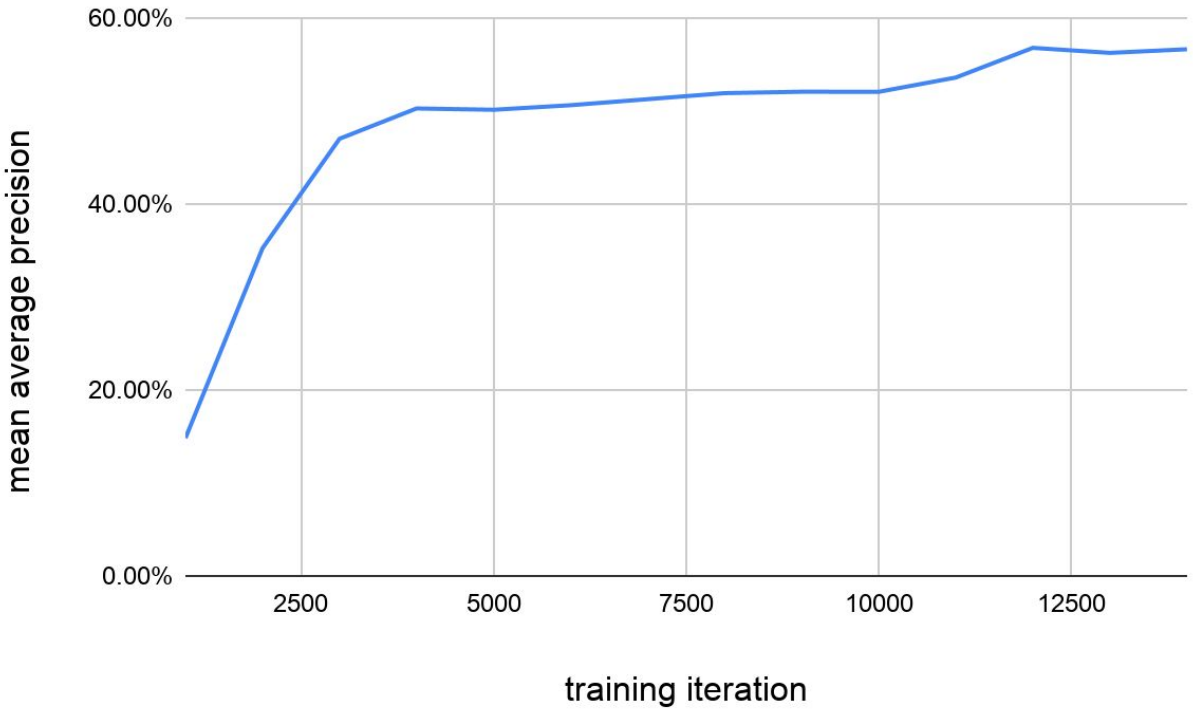
  max_batches = classes * 2000
  (but not less than 4000 in total)

  It means that every class should have around 2000 iterations. For example, in our custom dataset of 7 classes, the recommended total number of iterations for training on this particular dataset will be 14000.

- To define the best weights after training. Our task now is to define the best one to use for detection making sure that there is no overfitting. Weights will be saved every 1000 iterations.

- Algorithm is as following: start checking saved and trained weights from the end one by one calculating mean average precision (mAP). The goal is to find weights that have the biggest mAP. We can use these codes in the terminal to calculate the mAP.

  ```
  ./darknet detector map cfg/custom_data.data cfg/yolov3-custom_train.cfg
  backup/yolov3-custom_train_8000.weights
  ```

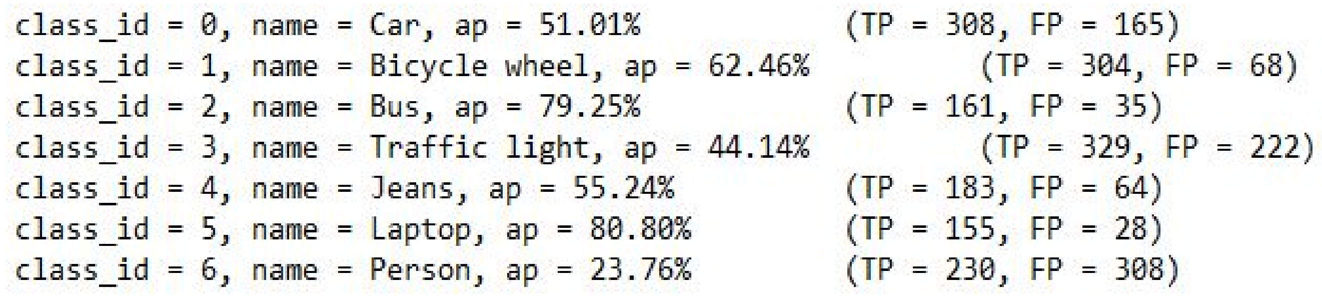


The best mAP is 56.82%, which appears at iteration 12000.
In this graph, the mean average precision lifts before 4000 iterations, then goes flat and the best result appears at 12000 iteration. it’s 2000 iterations for one class, but seems it’s still not enough. Another reason may be we need to use more dataset to train.

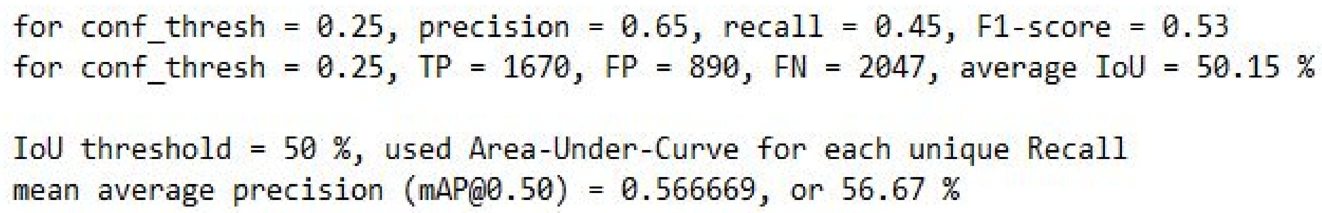
Final Result

1. The best model

The best model we got after the 14,000 iteration was acceptable, which could successfully detect all seven classes that we are interested in. However, it is not comparable to the official COCO dataset model provided by YOLO website, which can classify over 80 different classes with MAP above 90%. There are a couple classe that did not class well during the training process, for example: Person, Jeans, etc…. The reasons that cause this problem might be because the training dataset for Person and Jeans are overlapping. For instance, people wearing a pair of jeans only get labeled as either people or jeans. Another improvement could be increasing the size of the dataset and increasing the number of iterations.



```
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)
```



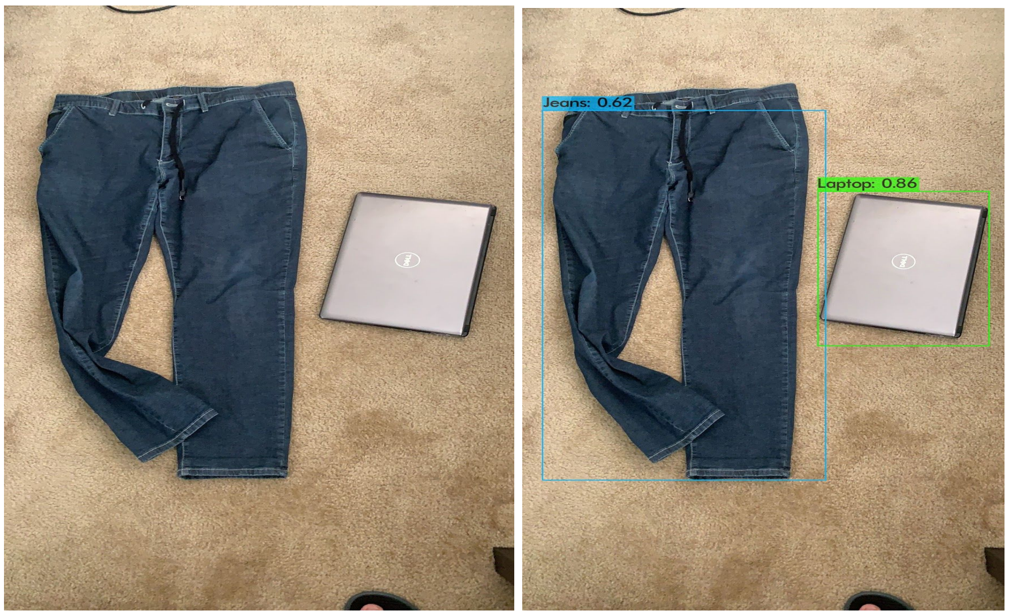
```
 for conf_thresh = 0.25, precision = 0.65, recall = 0.45, F1-score = 0.53
 for conf_thresh = 0.25, TP = 1670, FP = 890, FN = 2047, average IoU = 50.15 %

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

2. Display the result

The images below demonstrate what the original image and its corresponding result image look like. The Jeans have a confidence rate of 62% and laptops have a confidence rate of 86%. To do object detection on videos, it is as simple as splitting input video into continuous frames and treating each frame as image. Then process each image with the steps that demonstrated in 1 - 4. The only difference is to create an openCV writer to save each modified frame (with bounding boxes and confidence rate drawed on the frame) back into video format (.avi).

Example Processed Image:



Links of Classified Videos:

- https://drive.google.com/file/d/1xc2RdESogHPag3VwwnbuNQs-RX--mZzF/view?usp=sharing
- https://drive.google.com/file/d/1IuAp5ggEG8hvPBlzG8yjvvBXh1PIXClW/view?usp=sharing

Conclusion

Overall, our model still detects certain classes in images or videos, even though it performs bad at a couple of classes. I learned how to train a model in darknet which is very useful in a lot of applications like ImageNet. But there is still a lot of space left to improve the model. In the future, we will plan to train on a bigger dataset and increase the number of iteration numbers.

Calculate the percentage

My main part is training and for darknet I modified configuration and command line. For the other part, I work with the group members. The percentage is about 70%.

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

[4]Valentyn S. Training YOLO v3 for Objects Detection with Custom Data. Retrieved from https://storage.googleapis.com/openimages/web/index.html