

FINAL AI REPORT

IMPROVEMENT OF PREVIOUS MODEL FOR DETECTION OF POTHOLES



TEAM MEMBERS

ARIHANT JAIN - 2019213

ARJUN KHARE - 2019214

INSTRUCTOR - Dr.Kusum Kumari Bharti

INTRODUCTION

India is the second-largest road network in the world. Therefore, the road network plays an important role in Indian economic development and social functioning. According to the report in the last ten years, the Road Transport sector GDP grew at an annual average rate close to 10% compared to the overall annual GDP growth of 6%. Nowadays road construction is done very rapidly by the government of India. But road maintenance is a challenging task because of the poor drainage system and overloaded vehicles. Due to poor maintenance of roads, potholes appear on the road which causes road accidents. According to statistics submitted by the government of India from 2013 to 2016 potholes claimed 11,836 lives and 36,421 people got injured. Pothole problems cannot be resolved easily because every year almost all the places suffer from floods, disasters, heavy rainfall, etc. Though we cannot maintain the road, we can reduce the number of accidents which are increasing every year.

Detection of potholes using artificial intelligence can be a game changer as the results can be used to fill them up as early as possible and also warn people about upcoming potholes so that they can reduce their speed accordingly to be safe.

We first used the F-RCNN model to train, implement our model and saw the results, it took a long time to train and test the model. Now we will be using YOLO Object Detection Algorithm to detect Potholes and compare it to our previous model.

APPROACH

The YOLO framework (You Only Look Once) on the other hand, deals with object detection in a different way. It takes the entire image in a single instance and predicts the bounding box coordinates and class probabilities for these boxes. **The biggest advantage of using YOLO is its superb speed** — it's incredibly fast and can process 45 frames per second. YOLO also understands generalized object representation.

FRCNN is an algorithm that is based on Classification and it uses two-stage methods. Firstly, interested regions are selected. Secondly, they are classified using Convolutional Neural Networks by running predictions.

YOLO is an algorithm based on regression and is a single-step process wherein, in one run, the image is screened, and objects are located with the bounding boxes as well as their class is predicted in the same run. It is generally used for real-time detection where time, speed, and accuracy form prime concerns, it is primarily used in the application of Artificial intelligence and Deep Learning.

In order to train YOLO object detection algorithm and use it for detection, we need to perform the following steps -

Clone the darknet repository

- Compile the Source - We can directly compile the source using make. Just go to the directory where darknet is cloned and run the command

Make makes use of the Makefile, which consists of instructions to compile the C source files. After completion of the make process, you will get a file named darknet, which is a binary and executable file. You can use this binary executable to run the YOLO.

YOLO Structure

- Configuration Files

YOLO is entirely plug-n-play, that is, you can configure YOLO to detect any type of objects. In fact, you can also modify the CNN architecture itself and play around with it. YOLO does this by making use of configuration files under cfg/. The configuration files end with .cfg extension, which YOLO can parse. These configuration files consist of mainly:

- CNN Architectures (layers and activations)
- Anchor Boxes
- Number of classes
- Learning Rate
- Optimization Technique
- input size
- probability score threshold
- batch sizes

- Based on different versions, there can be many configurations from V1 to V3 to full training to tiny layers.

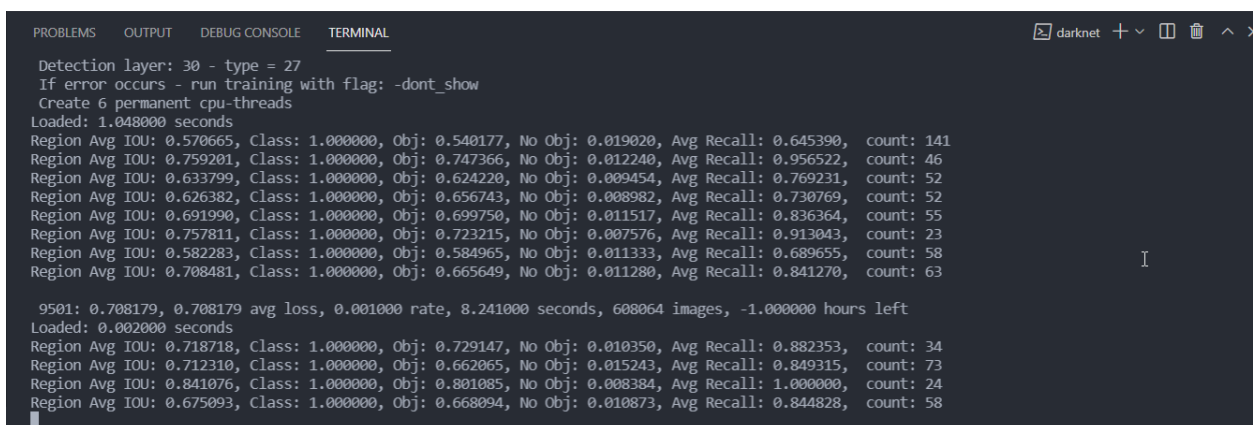
Weights

- Each configuration has corresponding pre-trained weights
- Full Weight - This is the weight trained on full 9000+ classes.
- Tiny Weight - This is the weight trained on only 80 classes.
- - first of all, we need to Train the RPN architecture with the dataset so that it can propose the expected region.
- Training of Fast R-CNN - As we know, Faster R-CNN is a combination of RPN and Fast R-CNN. So, we've to train a Fast R-CNN with the proposals obtained by RPN (after training) in order to make a Faster R-CNN.
- Fixing Convolutional layers, fine-tuning unique layers to RPN.
- Fixing Convolutional layers, fine-tuning fully connected layers of Fast R-CNN

Train YOLO

The training is a bit more complex because we have to get things and configurations right. The following command does everything:

`./darknet detector train <obj.data> <config> <weights>`



```

PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL
Detection layer: 30 - type = 27
If error occurs - run training with flag: -dont_show
Create 6 permanent cpu-threads
Loaded: 1.048000 seconds
Region Avg IOU: 0.570665, Class: 1.000000, Obj: 0.540177, No Obj: 0.019020, Avg Recall: 0.645390, count: 141
Region Avg IOU: 0.759201, Class: 1.000000, Obj: 0.747366, No Obj: 0.012240, Avg Recall: 0.956522, count: 46
Region Avg IOU: 0.633799, Class: 1.000000, Obj: 0.624220, No Obj: 0.009454, Avg Recall: 0.769231, count: 52
Region Avg IOU: 0.626382, Class: 1.000000, Obj: 0.656743, No Obj: 0.008982, Avg Recall: 0.730769, count: 52
Region Avg IOU: 0.691990, Class: 1.000000, Obj: 0.699750, No Obj: 0.011517, Avg Recall: 0.836364, count: 55
Region Avg IOU: 0.757811, Class: 1.000000, Obj: 0.723215, No Obj: 0.007576, Avg Recall: 0.913043, count: 23
Region Avg IOU: 0.582283, Class: 1.000000, Obj: 0.584965, No Obj: 0.011333, Avg Recall: 0.689655, count: 58
Region Avg IOU: 0.708481, Class: 1.000000, Obj: 0.665649, No Obj: 0.011280, Avg Recall: 0.841270, count: 63

9501: 0.708179, 0.708179 avg loss, 0.001000 rate, 8.241000 seconds, 608064 images, -1.000000 hours left
Loaded: 0.002000 seconds
Region Avg IOU: 0.718718, Class: 1.000000, Obj: 0.729147, No Obj: 0.010350, Avg Recall: 0.882353, count: 34
Region Avg IOU: 0.712310, Class: 1.000000, Obj: 0.662065, No Obj: 0.015243, Avg Recall: 0.849315, count: 73
Region Avg IOU: 0.841076, Class: 1.000000, Obj: 0.801085, No Obj: 0.008384, Avg Recall: 1.000000, count: 24
Region Avg IOU: 0.675093, Class: 1.000000, Obj: 0.668094, No Obj: 0.018873, Avg Recall: 0.844828, count: 58

```

Note=> Initially we use default darknet weights after a certain number of epochs a weight file will be generated which will be automatically updated and if we stop at any point we can

use that weight file to resume training and for testing our model.

Test YOLO

As told earlier, everything is run using the darknet executable file. If we have an image, then we can try predicting the objects as:

```
./darknet detect <config> <weights> <image>
```

Once done, there will be an image named predictions.jpeg in the same directory as the darknet file. You can view the prediction classes along with corresponding bounding boxes.

Training Command Breakdown

Here, .cfg and .weights are what they are meant to be — configurations and weight files as mentioned earlier. Everything happens using the obj.data file.

obj.data

```
classes= 1
```

```
train = custom/cfg/train.txt
```

```
valid = custom/cfg/test.txt
```

```
names = obj.names
```

```
backup = backup/
```

obj.names

This file consists of a list of class names. Example:

```
cat
```

```
dog
```

```
background
```

```
bike
```

train.txt

This file consists of a list of training images that we are going to feed into the network. The

content is similar:

custom-data/Pothole/11.jpg

custom-data/Pothole/12.jpg

custom-data/Pothole/13.jpg

...

...

Here, custom-data/Pothole/ consists of all the training images. Along with the images, this directory also consists of a text file for the bounding box corresponding to the image.

So, you will have custom-data/Pothole/11.txt whose content can be:

0 0.32502480158730157 0.3950066137566138 0.12896825396825398 0.09523809523809523

Here, the first number represents the id or class corresponding in obj.names. The remaining numbers represent the bounding box. If there were multiple boxes of multiple classes, it'd be like:

0 0.32502480158730157 0.3950066137566138 0.12896825396825398 0.09523809523809523

0 0.52502480158730157 0.3950066137566138 0.12896825396825398 0.09523809523809523

1 0.32502480158730157 0.3950066137566138 0.12896825396825398 0.09523809523809523

test.txt

This file consists of a list of test images.

custom-data/Pothole/400.jpg

custom-data/Pothole/401.jpg

custom-data/Pothole/402.jpg

...

...

DATASET INFORMATION

Dataset Link -

<https://www.kaggle.com/sachinpatel21/starter-code-to-view-dataset-images/data>

Image labelling using - <https://github.com/tzutalin/labelImg>

We had got the Pothole images dataset and I had used labelImg which is an open tool for labelling your images.

RESULTS



predictions.jpg X

predictions.jpg

Pothole: 0.93

PROBLEMS	OUTPUT	DEBUG CONSOLE	TERMINAL
24 conv	1024	3 x 3/ 1	13 x 13 x1024 -> 13 x 13 x1024 3.190 BF
25 route	16		-> 26 x 26 x 512
26			
reorg_old			
reorg_old		/ 2 26 x 26 x 512 ->	13 x 13 x2048
27 route	26 24		-> 13 x 13 x3072
28 conv	1024	3 x 3/ 1	13 x 13 x3072 -> 13 x 13 x1024 9.569 BF
29 conv	30	1 x 1/ 1	13 x 13 x1024 -> 13 x 13 x 30 0.010 BF
30 detection			

mask scale: Using default '1.000000'

Total BFLOPS 34.876

avg_outputs = 645100

Allocate additional workspace_size = 229.27 MB

Loading weights from .\custom-backup\yolo-pothole-train_last.weights...

seen 64, trained: 608 K-images (9 Kilo-batches_64)

Done! Loaded 31 layers from weights-file

Detection layer: 30 - type = 27

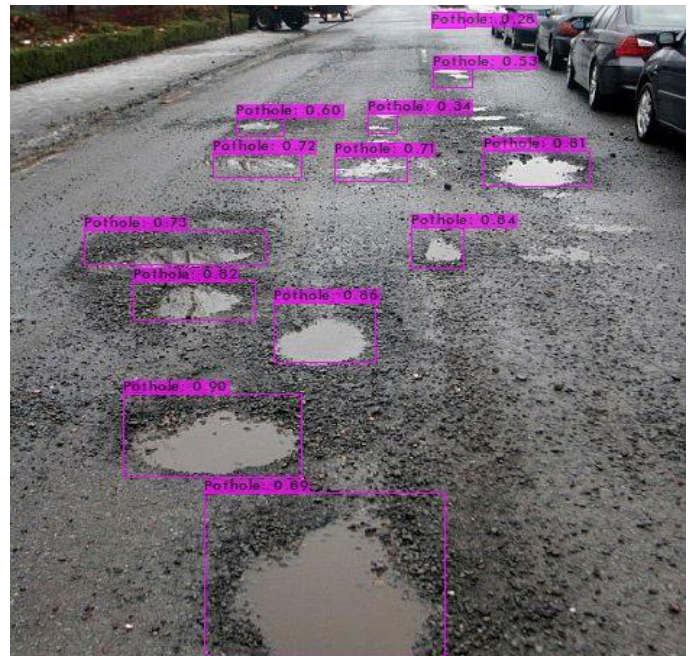
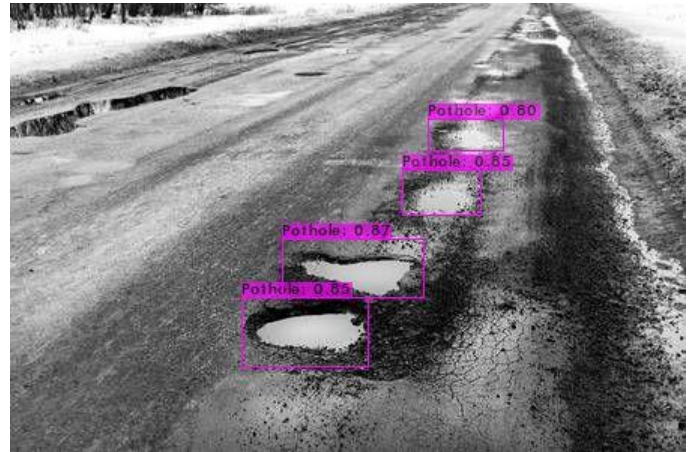
.\test_2.jpg: Predicted in 1210.494000 milli-seconds.

Pothole: 93%

Input Images

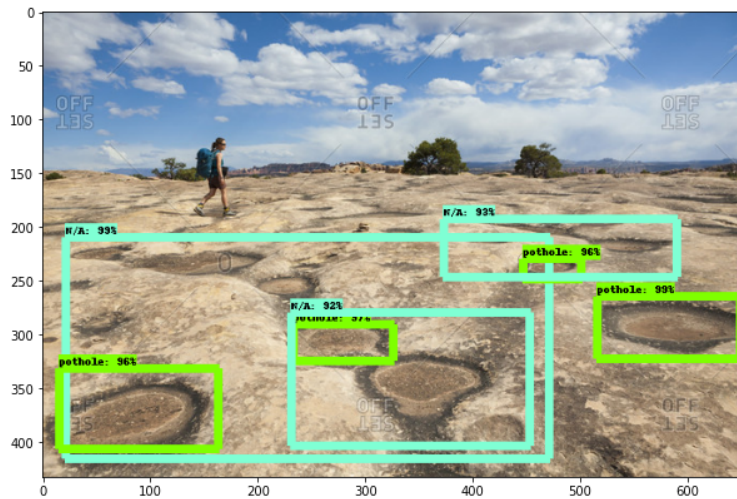


Output Images

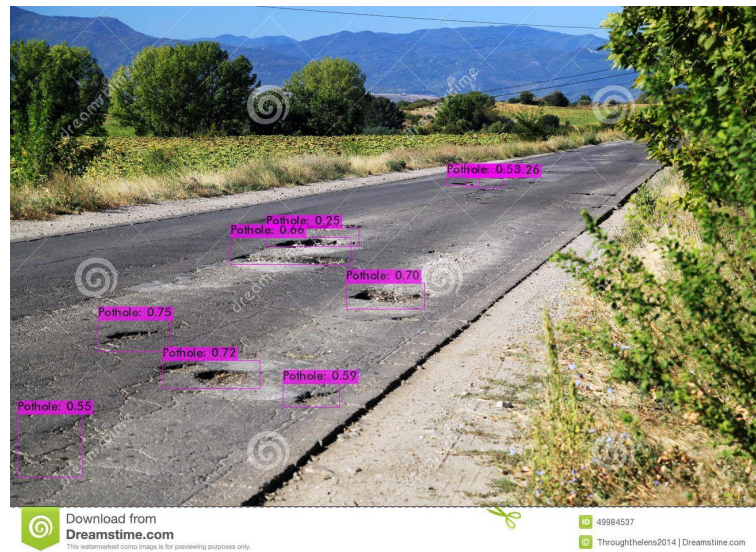
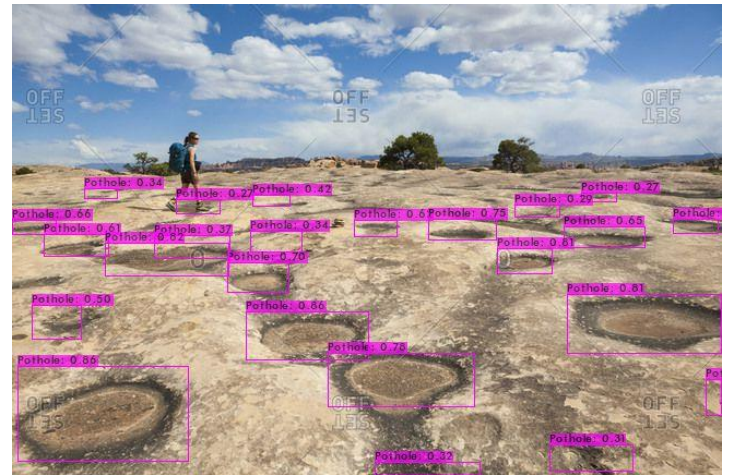


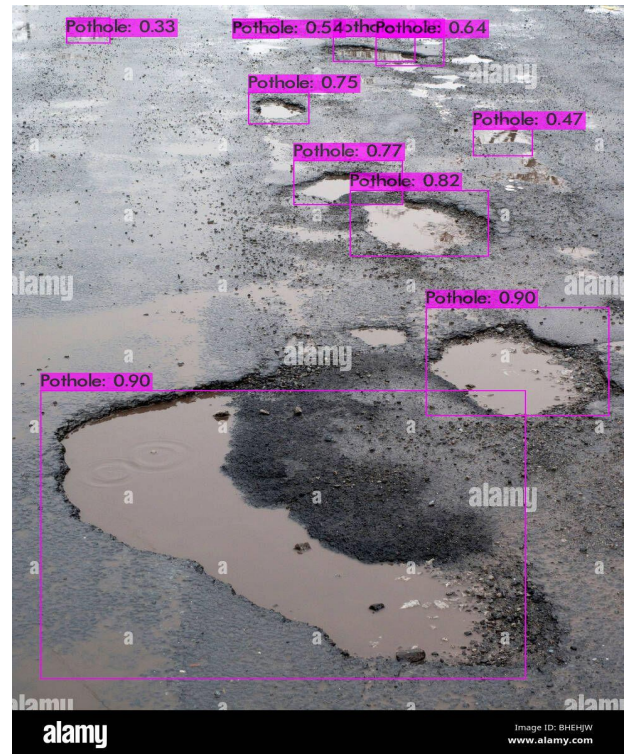
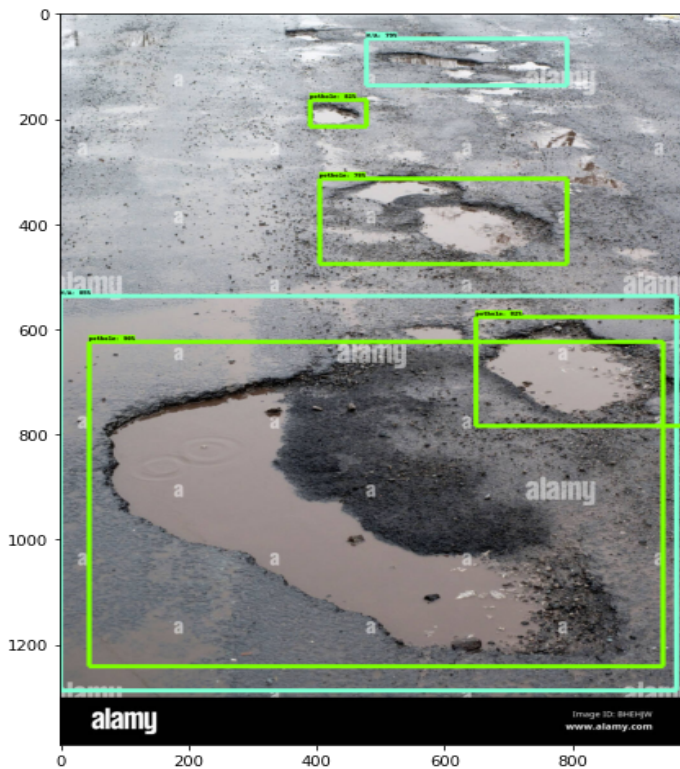
COMPARISON FRCNN VS YOLO

FRCNN



YOLO





CONCLUSION

- FRCNN does not have a wide range in detection of Potholes whereas YOLO detects almost every Pothole within a wide range.
- FRCNN's most of the detections are of large Potholes whereas YOLO tries to detect small Potholes also.
- FRCNN takes more time in detection than YOLO.

Both the models were effectively able to detect potholes from images and videos provided to it. In which YOLO performs better than FRCNN with respect to speed and accuracy.

REFERENCES

[Research Paper](#)

[Dataset](#)

[Darknet](#)

[YOLO Description](#)