

DETECTION OF POTHOLES USING DEEPLARNING

Submitted in partial fulfillment of the requirements for the award of
Bachelor of Engineering Degree in Computer Science and Engineering

By

MARRI HARIPRASAD REDDY (37110439)



**DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
SCHOOL OF COMPUTING**

SATHYABAMA

INSTITUTE OF SCIENCE AND TECHNOLOGY

(DEEMED TO BE UNIVERSITY)

**Accredited with Grade "A" by NAAC | 12B Status by UGC | Approved by
AICTE JEPPIAAR NAGAR, RAJIV GANDHI SALAI, CHENNAI - 600 119**

March-2021

SATHYABAMA INSTITUTE OF SCIENCE AND TECHNOLOGY



(Established under Section 3 of UGC Act, 1956)

Jeppiaar Nagar, Rajiv Gandhi
Salai, Chennai -600 119
www.sathyabama.ac.in



DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

BONAFIDE CERTIFICATE

This is to certify that this Project Report is the bonafide work of **MARRI HARIPRASAD REDDY (17SCS0160)** who carried out the project entitled **“DETECTION OF POTHoles USING DEEPLearning”** under our supervision from August 2020 to March 2021.

Internal Guide

Dr. USHA NANDHINI , M.E., Ph.D.,

Head of the Department

Dr.S.VIGNESHWARI , ME.,Ph.D.,

Submitted for Viva Voce Examination held on _____

Internal Examiner

External Examiner

DECLARATION

I, **MARRI HARIPRASAD REDDY(17SCS0160)** hereby declare that the Project Report entitled “**DETECTION OF POTHOLE USING DEEPLARNING**” done by me under the guidance of **Dr.USHA NANDHINI ,M.E.,Ph.D.**, at Sathyabama Institute of Science and Technology is submitted in partial fulfillment of the requirements for the award of Bachelor of Engineering degree in Computer Science and Engineering.

DATE:

PLACE:

SIGNATURE OF THE CANDIDATE

ACKNOWLEDGEMENT

I am pleased to acknowledge my sincere thanks to Board of management of **SATHYABAMA INSTITUTE OF SCIENCE AND TECHNOLOGY** for their kind encouragement in doing this project and for completing it successfully. We are grateful to them.

I convey my thanks to **Dr. T. Sasikala, M.E., Ph.D., Dean, School of Computing** and **Dr. S.Vigneshwari, M.E., Ph.D., and Dr.L.Lakshmanan, M.E.,Ph.D., Heads of the Department, Department of Computer Science and Engineering** for providing us the necessary support and details at the right time during the progressive reviews.

I would like to express my sincere and deep sense of gratitude to our Project Guide **Dr. USHA NANDHINI ,M.E.,Ph.D.,**for his valuable guidance, suggestions and constant paved way for the successful completion of my project work.

I wish to express my thanks to all Teaching and Non-teaching staff members of the **Department of Computer Science and Engineering** who were helpful in many ways for the completion of project.

ABSTRACT

A pothole is one of the major types of defects frequently found on the road whose assessment is necessary to process. It is one of the critical reasons for accidents on the road and the wear and tear of vehicles. Road defects assessment is to be done through defects data collection and processing of this collected data. Currently, using various types of imaging systems, data collection is near becoming automated, but an assessment of defects from collected data is still manual. Manual classification and evaluation of potholes are expensive, labor-intensive, time-consuming, and thus slow down the overall road maintenance process.

This project describes a method for classification and detection of the potholes on road images using convolutional neural networks, which are deep learning algorithms. We used a convolutional neural network-based approach with pre-trained models to classify given input images into pothole and non- pothole categories in the proposed system. The method was implemented in python using the OpenCV library under windows and the collab environment. The results are evaluated and compared for convolutional neural networks and various seven pre-trained models through accuracy, precision, and recall metrics.

TABLE OF CONTENTS

CHAPTER	TITLE	PAGE
NO.		NO.
	ABSTRACT	i
	LIST OF FIGURES	iv
1	INTRODUCTION	5
2	LITERATURE SURVEY	7
3	AIM AND SCOPE OF THE PROJECT	10
	3.1 AIM	10
	3.2 PROJECT SCOPE	10
	3.3 OVERALL OBJECTIVE	10
	3.4 PROPOSED SYSTEM	11
4	ALGORITHM AND TECHNIQUES USED	12
	4.1 ALGORITHM EXPLANATION	12
	4.2 YOLO v3	13
	4.3 ANCHOR BOXES AND PREDICTION	17
	4.3.1 COORDINATE CENTERS	18
	4.3.2 NON FUNCTIONAL REQUIREMENTS	19
	4.3.3 ENVIRONMENTAL REQUIREMENTS	19
	4.4 DIMENSIONS OF THE BOUNDARY BOXES	20

	4.5 OBJECT SCORES AND CLASS CONFIDENCE	21
	4.6 PREDICTION ACROSS DIFFERENT SCALES	21
	4.7 OUTPUT PROCESS	22
5	DESIGN	24
6	IMPLEMENTATION	27
	6.1 DATA COLLECTION	27
	6.1.1 ABOUT THE DATA SET	27
	6.2 LABELING THE DATA	28
	6.2.1 INSTALLING LABELING	29
	6.3 TRAINING YOLO MODEL	31
	6.4 TESTING AND VALIDATION RESULT	33
	6.5 CODE TO RUN THE PROJECT	37
7	CONCLUSION	40
	REFERENCES	41

LIST OF FIGURES

FIGURE NO	FIGURE NAME	PAGE
		NO
4.1	THE MODEL	13
4.2	ARCHITETURE	14
4.3	IMAGE GRID	16
4.4	FORMULA	18
4.5	DIMENSIONS OF BOUNDARY BOXES	20
4.6	PREDICTION FREATER AT DIFFERENT SCALES	22
4.7	INTERSECTION OVER UNION	23
5.1	FLOW CHART(BASE PAPER)	24
5.2	FLOW CHART DESIGN	25
6.1	POTHOLES PRIDITION BEFORE PREDICTION	28
6.2	GPU	29
6.3	ANNOTATING USING GPU	30
6.4	PARAMETERS OF THE ANNOTATED POTHOLES IMAGE	30
6.5	OUTPUT ARCHITETURE	32
6.6	EXAMPLES OF PRIDITION.JPG	33
6.7	EXAMPLES OF PRIDITION.JPG	34
6.8	INPUT IMAGE	35

6.9	ARCHITETURE	36
6.10	PREDICTION.JPG	37

CHAPTER 1

INTRODUCTI

ON

One of the growing fields in Science and Technology is the Advanced Driver Assistance Systems (ADAS) [2], [3]. Over the year's transportation has gained paramount importance in daily life. Currently, the world is experiencing an exponential advancement in intelligent transportation systems. The ADAS has automated a plethora of features in vehicles leading to a smart and sophisticated driving experience.

When the past present future of ADAS is examined, the word "safety" is always given prime importance. In essence, ADAM's main job is to reduce the fatalities caused by roads due to human errors. While speaking about road safety, potholes play a crucial role. In India, for three years (2015-2017), there have been over 9300 deaths and 25000 injuries exclusively only due to potholes not taking into consideration any other object that caused accidents [4]. When driving for prolonged hours or under stress or tension, the driver tends to miss out on concentration.

One of the primary reasons behind such accidents is the driver's inability to pay attention to every single detail on the road, and that's when the ADAS comes into the picture. The data can either be directly reported to the driver by giving an alert symbol in the cabin of the vehicle or the data can be used by the autonomous driverless system where the system decides on what action it needs to mete out in order to prevent a collision and ensure a safe and comfortable riding experience for the passengers. Potholes detection can be broadly classified into three categories namely vibration- based methods, 3D reconstruction-based methods, and vision-based methods (2D image) [5]. Out of the three vision-based methods prove to be the most cost-effective ways of detection.

At the same time, accurate detection of potholes using a 2D image is very tough. So, there is a need for a system which can detect Potholes from a 2D image

with better

accuracy and at greater speed. Especially countries like India need more focus as the potholes count is huge. Also, the system will have technical differences when implemented in different places depending on the degree of road maintenance.

Detection of potholes is much more difficult when compared to other objects such as a pedestrian, vehicles, traffic signs, etc. because the former has a wide range of geometrics. When comparing the recognition algorithms, Convolutional Neural Networks (CNN) has proven to be one of the best [6].

In this paper, pothole detection is implemented using one of the CNN family's unique representatives You Only Look Once (YOLO) to a newly created dataset for Indian roads. The pothole detection system aims to provide a solution for both municipal authority and the citizens to reduce the number of potholes by developing an application for mobiles.

The entire design of the system is based on the integration of various libraries and frameworks. A mobile application, APIs to provide insights from data gathered, an object detection model to precisely detect the potholes and a cloud storage solution to store and handle various forms of data. The application has two major functionalities: First, for detection and gathering of data mainly for the local municipal body's usage and the other is the view of the gathered data on a map which is helpful for both the authorities responsible for maintenance and also for citizens of the locality to be aware of their surrounding roads.

CHAPTER 2

LITERATURE SURVEY

There are so many architectures and methodologies that can be used for potholes detection. Evaluating a pothole in most cases requires 3D equipment which is very expensive and not feasible on a large scale [7]. Instead, detecting a defect in the road using an image and then further processing it into defective and not-defective regions and analyzing the patterns in both the regions can help differentiate a pothole from the non-defective region which is essentially the road. Another method of analyzing these problems is by using three features of an image which are, the texture, shape, and dimensions of the defective area in the frame [8], [9]. But the method doesn't have machine learning at the core of it.

Therefore, one of the most basic ways to carry out potholes detection is by using Convolutional Neural Networks (CNNs). The Object Detection field started off with Region-Based Convolutional Neural Networks (R-CNN) and moved to faster and more advanced algorithms like YOLO (You Only Look Once) and SSD (Single-Shot Detection). The R-CNN's use selective search algorithms to extract 2000 bounding boxes from the input image in the first step itself to stick with processing only the most important features in an input based on color, pattern, shape, and size [10]. The R- CNNs use a three-stage mechanism: feature extraction via Selective Search Algorithm, SVM classification, and Regression modeling for tight bounding boxes. The Fast R- CNN model uses a single-stage mechanism where it directly passes the input to a CNN and the output from this CNN is the Regions of Interest (RoI) [11].

Then an RoI pooling layer is applied to the output from the CNN to warp the images to the size the Network is accustomed to working with. These ROIs are given to a fully connected Neural Network (NN) that segregates them and returns bounding boxes on the ROIs using linear regression and softmax networks working in a parallel manner. This provides drastic speed gains as well as savings in terms of the size of the model. Although Fast R-CNN uses a one-

stage mechanism, it relies on the selective search method initially and this consumes time.

The Faster R-CNN passes the input image to a CNN which returns the feature map of the input and these feature maps are passed to Region Proposal Networks which gives object proposals along with a score of how much of an object that particular prediction resembles [12] [13]. This is passed to an RoI pooling layer which warps all the proposed Regions to the same size to make them compatible with further hidden networks that will be used in the process. The final layer is similar to that of the R- CNN which is a fusion of both the softmax and Linear Regression Layer which classifies and draws bounding boxes for the various objects available in the image.

Even though the Faster R-CNN has improved speed, it has a fundamental drawback as the other two R-CNNs, it doesn't look at the complete image at once, instead, they focus on parts of the image sequentially and this requires many complex stages. To work around this issue, the SSD and YOLO methods were created. The framework of YOLO yields drastic speed gains. It processes up to 45 fps, which is a massive improvement compared to Region-based CNN. To view the real-world events in real-time smoothly, 30 fps is a sufficient speed, but with 45 fps YOLO favors real- time Object Detection. SSD [14] is similar to YOLO but is more lenient in terms of aspect ratio compatibility. It works with 6 more additional aspect ratios than YOLO.

This enables tighter object wrapping than YOLO. It also has higher accuracy than YOLO as it has a greater number of hidden layers. This added advantage of better accuracy comes with a trade-off, that is speed. Since cars move at very high speeds, pothole detection at such high speeds calls for high- speed detection and one of the highest speeds can be obtained by using YOLO as per the explanations available and illustrated above.

In the pothole detection method, various studies have been carried out with different approaches, for example using a laser [8], vibration sensor [9], [10] and image-based detection, both two-dimensional [11] and three-dimensional using a stereo camera [12][13]. The development of advanced

image processing technique, easy image interpretation, and the availability of low-cost camera acquisition devices have inspired the development of image-based pothole detection. Likewise, with the presence of new methods in the computer vision study, especially related to objects detection, give new opportunities to implement it in the road damage detection. In this study, YOLO (You Only Look Once), as one of the state-of-the-art methods in the field of object detection, is chosen as a pothole detection method. This research, hopefully, provides an alternative pothole solution on the road surface for road distress identification and classification as a complement to existing method

CHAPTER 3

AIM AND SCOPE OF THE PROJECT

3.1 AIM :

The main aim of the project is to identify the potholes on the roads and mark them on the road using deep learning.

3.2: PROJECT SCOPE :

The primary goal is to extract pattern using a convolutional neural network-based approach with pre-trained models to classify given input images into pothole and non-pothole categories in the proposed system. The method was implemented in python using the OpenCV library under windows and the collab environment. The results are evaluated and compared for convolutional neural networks and various seven pre-trained models through accuracy, precision, and recall metrics.

3.3 OVERALL OBJECTIVE :

This analysis aims to observe which features are most helpful in predicting the crime rate and to see the general trends that may help us in model selection. Currently, using various types of imaging systems, data collection is near becoming automated, but an assessment of defects from collected data is still manual. Manual classification and evaluation of potholes are expensive, labor-intensive, time-consuming, and thus slow down the overall road maintenance process.

3.4 PROBLEM ANALYSIS

There is an increasing need for assessment of national road condition. Currently, some automatic devices have been extensively applied to collect up-date data about road condition, such as the use of survey vehicle for collecting data— which make it faster and more accessible, and semi automatically data processing that is useful for policy decision making. Yet, demand for more detail road data is continuously growing; thus, data improvement needs to perform, upgrading the existing solution. To date, stages on identification and classification of road damages are being conducted semi manually based on images collected by survey vehicle; it is hindered due to the facts that this method is the cost-consuming process and may result in inconsistency. Therefore, this present work used YOLO with three different architecture configuration, i.e., Yolo v3, Yolo v3 Tiny, and Yolo v3 SPP, enabling us to create a more accurate assessment for detecting potholes on the road surface. The results showed the average mAP values for Yolo v3, Yolo v3 Tiny, and Yolo v3 SPP at 83.43%, 79.33%, and 88.93%. While the area measurement shows the accuracy of 64.45%, 53.26%, and 72.10% respectively. And it needs 0,04 second to detect each image. Conclusively, it shows a satisfying result in pothole detection; thus, this technique has a high opportunity to be developed and implemented as a tool for road assessment.

Road construction is one of the government's priorities. Therefore, every year targeted to increase national road length. These raises also improve road maintenance and road rehabilitation programs. Consequently, it is crucial to monitor the road condition to determine the proper programs.

CHAPTER 4

ALGORITHMS AND TECHNIQUES USED

4.1 ALGORITHM EXPLANATION:

The algorithm used to achieve this project is YOLO v3, a new approach to object detection. Prior work on object detection repurposes classifiers to perform detection. Instead, they framed object detection as a regression problem to spatially separated bounding boxes and associated class probabilities. A single neural network predicts bounding boxes and class probabilities directly from full images in one evaluation. Since the whole detection pipeline is a single network, it can be optimized end-to-end directly on detection performance. Our unified architecture is extremely fast. Their base YOLO model processes images in real-time at 45 frames per second.

A smaller version of the network, Fast YOLO, processes an astounding 155 frames per second while still achieving double the mAP of other real-time detectors. Compared to state-of-the-art detection systems, YOLO makes more localization errors but is far less likely to predict false detections where nothing exists. Finally, YOLO learns very general representations of objects. It outperforms all other detection methods, including DPM and RCNN, by a wide margin when generalizing from natural images to artwork on both the Picasso Dataset and the PeopleArt Dataset.

YOLO reasons globally about the image when making predictions. Unlike sliding window and region proposal-based techniques, YOLO sees the entire image during training and test time so it encodes contextual information about classes as well as their appearance. Fast R-CNN, a top detection method, mistakes background patches in an image for objects because it can't see the larger context. YOLO makes less than half the number of background errors compared to Fast R-CNN. YOLO learns generalizable representations of objects.

When trained on natural images and tested on the artwork, YOLO outperforms top detection methods like DPM and R-CNN by a wide margin. Since YOLO is highly generalizable it is less likely to break down when applied to new domains or unexpected input.

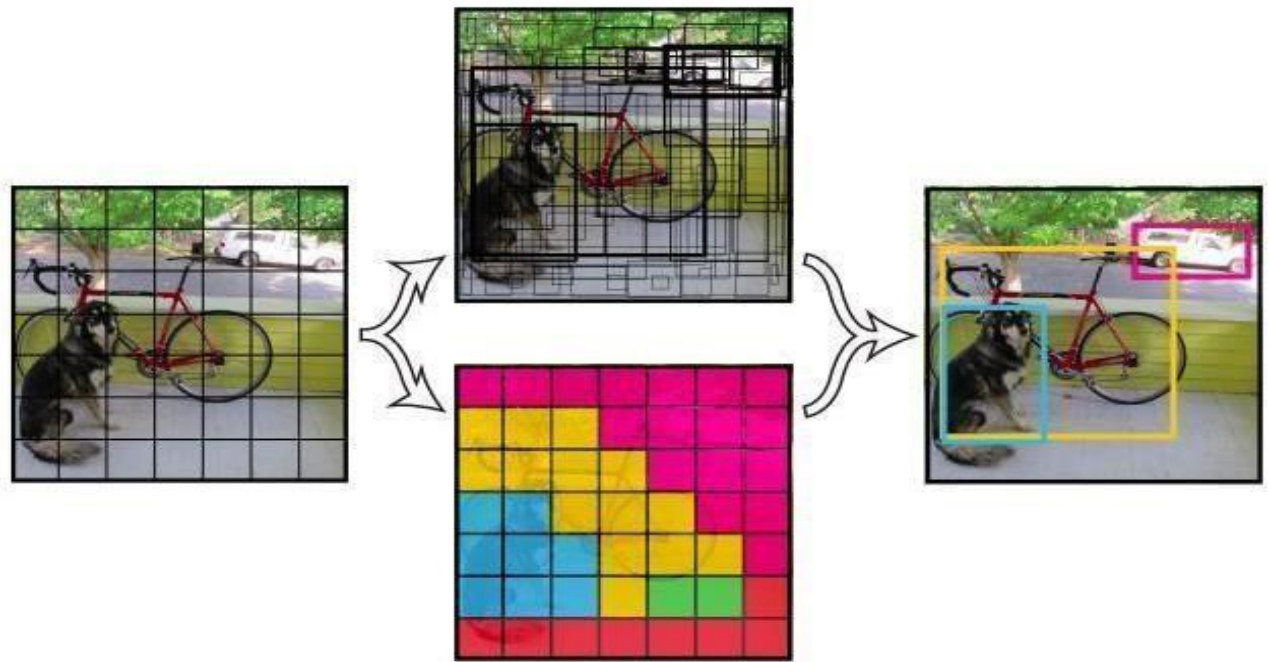


Figure 4.1 The Model

The Initial Yolo model is inspired by GoogLeNet which has 24 Convolutional layers and followed by 2 fully connected or Dense layers. Training a YOLO model is very fast and efficient.

4.2 YOLO v3:

YOLO makes use of only convolutional layers, making it a fully convolutional network (FCN). In the YOLO v3 paper, the authors present a new, deeper architecture of a feature extractor called Darknet-53. As its name suggests, it contains 53 convolutional layers, each followed by a batch normalization layer and Leaky ReLU activation. No form of pooling is used, and a convolutional layer with stride 2 is used to downsample the feature maps.

This helps in preventing the loss of low-level features often attributed to pooling. YOLO is invariant to the size of the input image. However, in practice, we might want to stick to a constant input size due to various problems that only show their heads when we are implementing the algorithm.

	Type	Filters	Size	Output
	Convolutional	32	3×3	256×256
	Convolutional	64	$3 \times 3 / 2$	128×128
1x	Convolutional	32	1×1	
	Convolutional	64	3×3	
	Residual			128×128
	Convolutional	128	$3 \times 3 / 2$	64×64
2x	Convolutional	64	1×1	
	Convolutional	128	3×3	
	Residual			64×64
	Convolutional	256	$3 \times 3 / 2$	32×32
8x	Convolutional	128	1×1	
	Convolutional	256	3×3	
	Residual			32×32
	Convolutional	512	$3 \times 3 / 2$	16×16
8x	Convolutional	256	1×1	
	Convolutional	512	3×3	
	Residual			16×16
	Convolutional	1024	$3 \times 3 / 2$	8×8
4x	Convolutional	512	1×1	
	Convolutional	1024	3×3	
	Residual			8×8
	Avgpool		Global	
	Connected		1000	
	Softmax			

Figure 4.2 Architecture

A big one among these problems is that if we want to process our images in batches (images in batches can be processed in parallel by the GPU, leading to speed boosts), we need to have all images of fixed height and width. This is needed to concatenate multiple images into a large batch. The network down samples the image by a factor called the stride of the network. For example, if the stride of the network is 32, then an input image of size 416 x 416 will yield an output of size 13 x 13. Generally, the stride of any layer in the network is equal to the factor by which the output of the layer is smaller than the input image to the network.

The input is a batch of images of shape $(m, 416, 416, 3)$. The output is a list of bounding boxes along with the recognized classes. Each bounding box is represented by 6 numbers $(\square\square, \square\square, \square\square, \square h, \square\square, \square)$ (pc, bx, by, bh, bw, c) . If we expand $\square c$ into an 80- dimensional vector, each bounding box is then represented by 85 numbers.

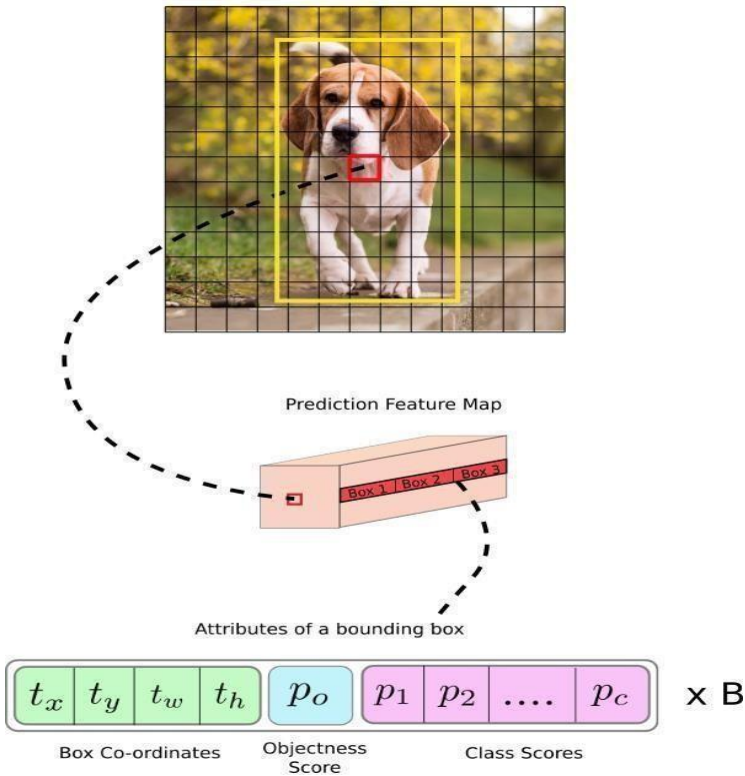


Figure 4.3 Image Grid

In the same way, as in all object detectors, the features learned by the convolutional layers are passed onto a classifier/regressor which makes the detection prediction.

In YOLO, the prediction is done by using a convolutional layer that uses 1×1 convolutions. So, the first thing to notice is our output is a feature map. Since we have used 1×1 convolution, the size of the prediction map is exactly the size of the feature map before it. In YOLO v3, the way you interpret this prediction map is that each cell can predict a fixed number of bounding boxes.

For example, if we have $(B \times (5 + C))$ entries in the feature map. B represents the number of bounding boxes each cell can predict.

According to the paper, each of these B bounding boxes may specialize in detecting a certain kind of object. Each of the bounding boxes has $5 + C$ attributes, which describe the center coordinates, the dimensions, the objectness score, and C class confidences for each bounding box. YOLO v3 predicts 3 bounding boxes for every cell.

We expect each cell of the feature map to predict an object through one of its bounding boxes if the center of the object falls in the receptive field of that cell.

This has to do with how YOLO is trained, where only one bounding box is responsible for detecting any given object. First, we must ascertain which of the cells this bounding box belongs to.

To do that, we divide the input image into a grid of dimensions equal to that of the final feature map. Take a look at the example below, where the input image is 416×416 , and the stride of the network is 32. As pointed earlier, the dimensions of the feature map will be 13×13 . We then divide the input image into 13×13 cells.

Then, the cell (on the input image) containing the center of the ground truth box of an object is chosen to be the one responsible for predicting the object. In the image, it is the cell that is marked red, which contains the center of the ground

box (marked yellow). Now, the red cell is the 7th cell in the 7th row on the grid.

We now assign the 7th cell in the 7th row on the feature map (corresponding cell on the feature map) as the one responsible for detecting the dog.

Now, this cell can predict three bounding boxes. Which one will be assigned to the dog's ground truth label? In order to understand that, we must wrap our heads around the concept of anchors. (Note that the cell we're talking about here is a cell on the prediction feature map. We divide the input image into a grid just to determine which cell of the prediction feature map is responsible for prediction).

4.3 Anchor Boxes and Prediction :

It might make sense to predict the width and the height of the bounding box, but in practice, that leads to unstable gradients during training. Instead, most of the modern object detectors predict log-space transforms, or simply offsets to predefined default bounding boxes called anchors.

Then, these transforms are applied to the anchor boxes to obtain the prediction. YOLO v3 has three anchors, which result in the prediction of three bounding boxes per cell.

Anchors are sort of bounding box priors, that were calculated on the COCO dataset using k-means clustering. We are going to predict the width and height of the box as offsets from cluster centroids. The center coordinates of the box relative to the location of the filter applications are predicted using a sigmoid function.

The following formula describes how the network output is transformed to obtain bounding box predictions:

$$\begin{aligned}b_x &= \sigma(t_x) + c_x \\b_y &= \sigma(t_y) + c_y \\b_w &= p_w e^{t_w} \\b_h &= p_h e^{t_h}\end{aligned}$$

Figure 4. 4 Formula

Here x_c, y_c, w, h are the x,y center coordinates, width, and height of our prediction. $\hat{x}_c, \hat{y}_c, \hat{w}, \hat{h}$ is what the network outputs. x_{min} and y_{min} are the top-left coordinates of the grid. w_a and h_a are anchor dimensions for the box.

4.3.1 Center Coordinates :

We are running our center coordinates prediction through a sigmoid function. This forces the value of the output to be between 0 and 1. Normally, YOLO doesn't predict the absolute coordinates of the bounding box's center. It predicts offsets which are:

- Relative to the top left corner of the grid cell which is predicting the object
- Normalized by the dimensions of the cell from the feature map, which is, 1.

Example, consider the case of our above dog image. If the prediction coordinates for the center is (0.4, 0.7), then this means that the center lies at (6.4, 6.7) on the 13 x 13 feature map. (Since the top-left coordinates of the red cell are (6,6)).

But wait, what happens if the predicted x and y coordinates are greater than one, for example (1.2, 0.7). This means that center lies at (7.2, 6.7). The center now lies in a cell just right to our red cell, or the 8th cell in the 7th row.

This breaks the theory behind YOLO because if we postulate that the red box is responsible for predicting the dog, the center of the dog must lie in the red cell, and not in the one beside it. So to solve this problem the output is passed through a sigmoid function, which squashes the output in a range from 0 to 1, effectively keeping the center in the grid which is predicting.

4.3.2 NON-FUNCTIONAL REQUIREMENTS :

Process of functional steps,

1. Problem define
2. Preparing data
3. Evaluating algorithms
4. Improving results
5. Prediction the result

4.3.3 ENVIRONMENTAL REQUIREMENTS :

1. Software Requirements:

Operating System : Windows

Tool : Anaconda with Jupyter Notebook

2. Hardware requirements:

Processor : Pentium IV/III

Hard disk : minimum 80 GB

RAM : minimum 2 GB

Notebook document:

Notebook documents (or “notebooks”, all lower case) are documents produced by the Jupyter Notebook App, which contain both computer code (e.g. python) and rich text elements (paragraph, equations, figures, links, etc...). Notebook documents are both human-readable documents containing the analysis description and the results (figures, tables, etc.) as well as executable documents which can be run to perform data analysis.

Jupyter Notebook App:

The Jupyter Notebook App is a server-client application that allows editing and running notebook documents via a web browser. The Jupyter Notebook App can be executed on a local desktop requiring no internet access (as described in this document) or can be installed on a remote server and accessed through the internet. In addition to displaying/editing/running notebook documents, the Jupyter Notebook App has a “Dashboard” (Notebook Dashboard), a “control panel” showing local files and allowing to open notebook documents or shutting down their kernels.

4.4 Dimensions of the Bounding Boxes :

The dimensions of the bounding box are predicted by applying a log-space transformation to the output and then multiplying with an anchor

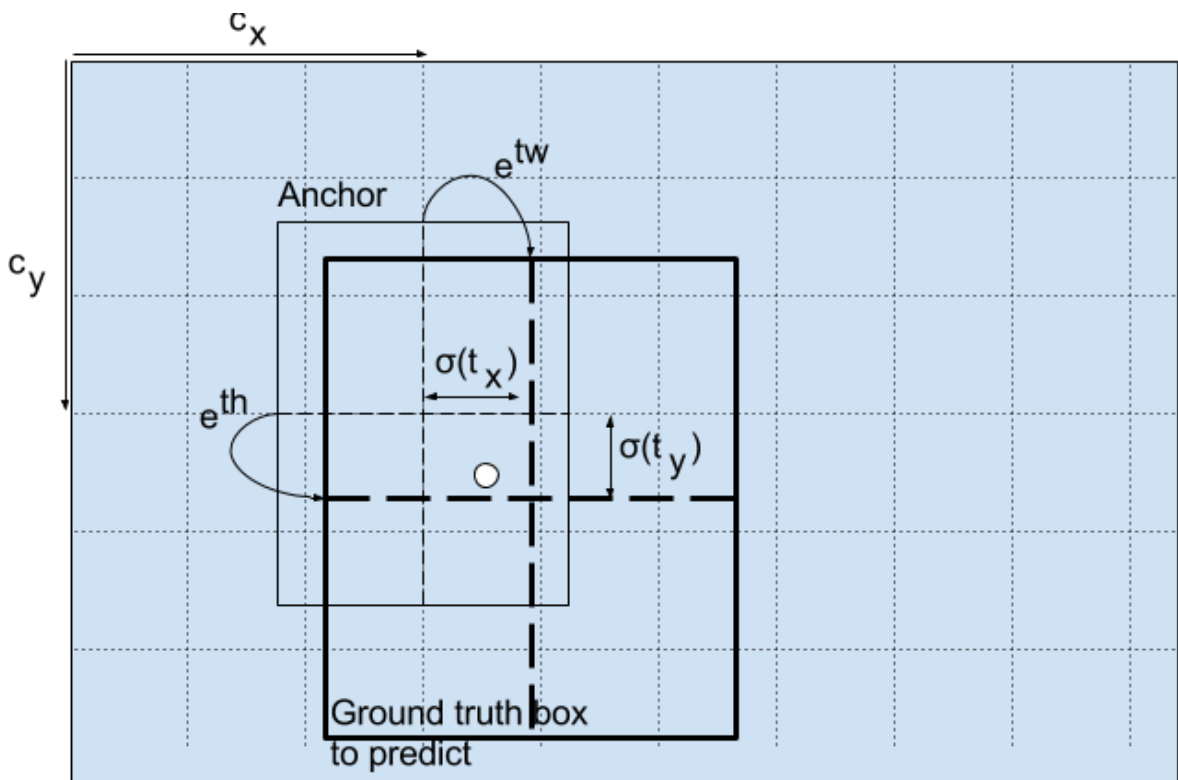


Figure 4.5 Dimensions of Bounding Boxes

Here predictions, $\sigma(t_x)$, and $\sigma(t_y)$, are normalized by the height and width of the image. (Training labels are chosen this way). So, if the predictions $\sigma(t_x)$ and $\sigma(t_y)$ for the box containing the dog are (0.3, 0.8), then the actual width and height on the 13 x 13 feature map are (13*0.3, 13*0.8).

4.5 Objectness Score and Class Confidences :

The First of all object score represents the probability that an object is contained inside a bounding box. It should be nearly 1 for the red and the neighboring grids, whereas almost 0 for, say, the grid at the corners.

The objectness score is also passed through a sigmoid, as it is to be interpreted as a probability. Talking about Class confidences, they represent the probabilities of the detected object belonging to a particular class (Dog, Cat, person, car, bicycle, etc). In older YOLO versions it was used to softmax the class scores.

In YOLO authors have decided to use sigmoid instead. The reason is that Softmaxing class scores assume that the classes are mutually exclusive. In simple words, if an object belongs to one class, then it cannot belong to another class. This is true for the COCO database, which we will implement first.

4.6 Prediction across different scales:

YOLO v3 makes predictions across 3 different scales. The detection layer is used to make detection at feature maps of three different sizes, having strides 32, 16, 8 respectively. This means, with an input of 416 x 416, we make detections on scales 13 x 13, 26 x 26, and 52 x 52.

The network down samples the input image until the first detection layer, where detection is made using feature maps of a layer with stride 32. Further, layers are upsampled by a factor of 2 and concatenated with feature maps of previous layers having identical feature map sizes. Another detection is now made at layer with stride 16. The same upsampling procedure is repeated, and a final detection is made at the layer of stride 8.

At each scale, each cell predicts 3 bounding boxes using 3 anchors, making the total number of anchors used 9. (The anchors are different for different scales)

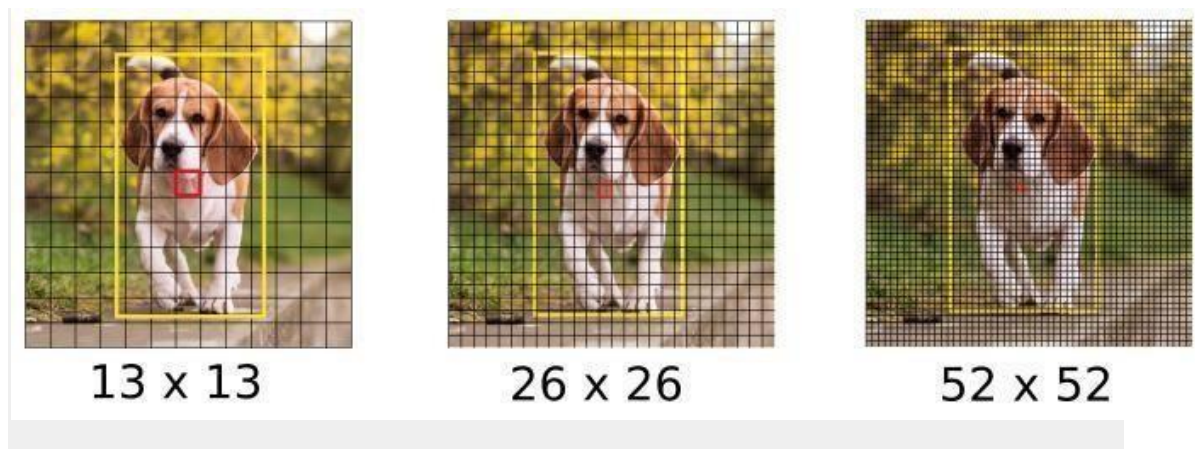


Figure 4. 6 Prediction Feature at Different Scales

YOLO authors report that this helps get better at detecting small objects, a frequent complaint with the earlier versions of YOLO. Upsampling can help the network learn fine-grained features that are instrumental in detecting a small object.

4.7 Output Processing (Filtering with a threshold on a less Scores):

For an image of size 416×416 , YOLO predicts $((52 \times 52) + (26 \times 26) + 13 \times 13) \times 3 = 10647$ bounding boxes. However, in the case of our image, there's only one object, a dog. So, you may ask how do we reduce the detections from 10647 to 1?

First, we filter boxes based on their objectness score. Generally, boxes having scores below a threshold (for example below 0.5) are ignored. Next, Non-maximum Suppression (NMS) intends to cure the problem of multiple detections of the same image. For example, all the 3 bounding boxes of the red grid cell may detect a box or the adjacent cells may detect the same object, so NMS is used to remove multiple detections.

Specifically, we'll carry out these steps:

- ❑ Get rid of boxes with a low score (meaning, the box is not very confident about detecting a class).

- ❓ Select only one box when several boxes overlap with each other and detect the same object (Non-max suppression)

For better understanding, I will use the same example with the car I mentioned above. At first, we are going to apply the first filter by thresholding. We would like to get rid of any box for which the class “score” is less than a chosen threshold.

The model gives us a total of $19 \times 19 \times 5 \times 85$ numbers, with each box described by 85 numbers.

It'll be convenient to rearrange the $(19, 19, 5, 85)$ (or $(19, 19, 425)$) dimensional tensor into the following variables:

- ❓ `box_confidence`: tensor of shape $(19 \times 19, 5, 1)$ containing $\square\square$ (confidence probability that there's some object) for each of the 5 boxes predicted in each of 19×19 cells.
- `boxes`: tensor of shape $(19 \times 19, 5, 4)$ containing $(\square\square, \square\square, \square h, \square\square)$ for each of the 5 boxes per cell.
- `box_class_probs`: tensor of shape $(19 \times 19, 5, 80)$ containing the detection probabilities $(\square 1, \square 2, \dots, \square 80)$ for each of the 80 classes for each of the 5 boxes per cell.

Even after filtering by thresholding over the class scores, we still end up with a lot of overlapping boxes. A second filter for selecting the right boxes is called NMS. NMS uses the very important function called “Intersection over Union”, or IoU.

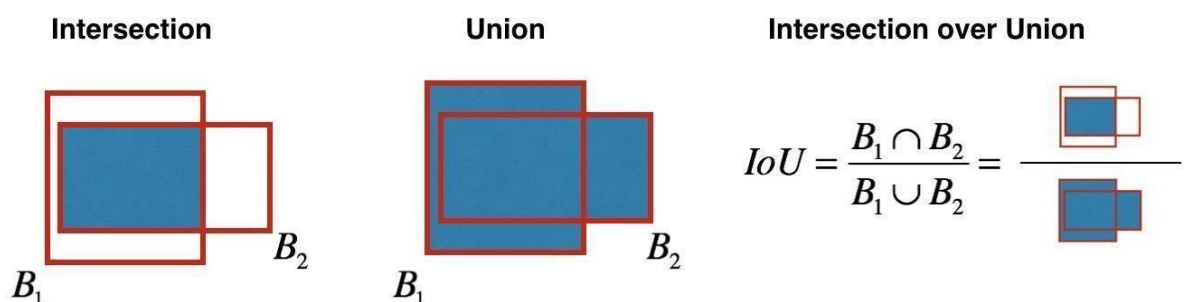


Figure 4. 7 Intersection over union

CHAPTER 5: DESIGN

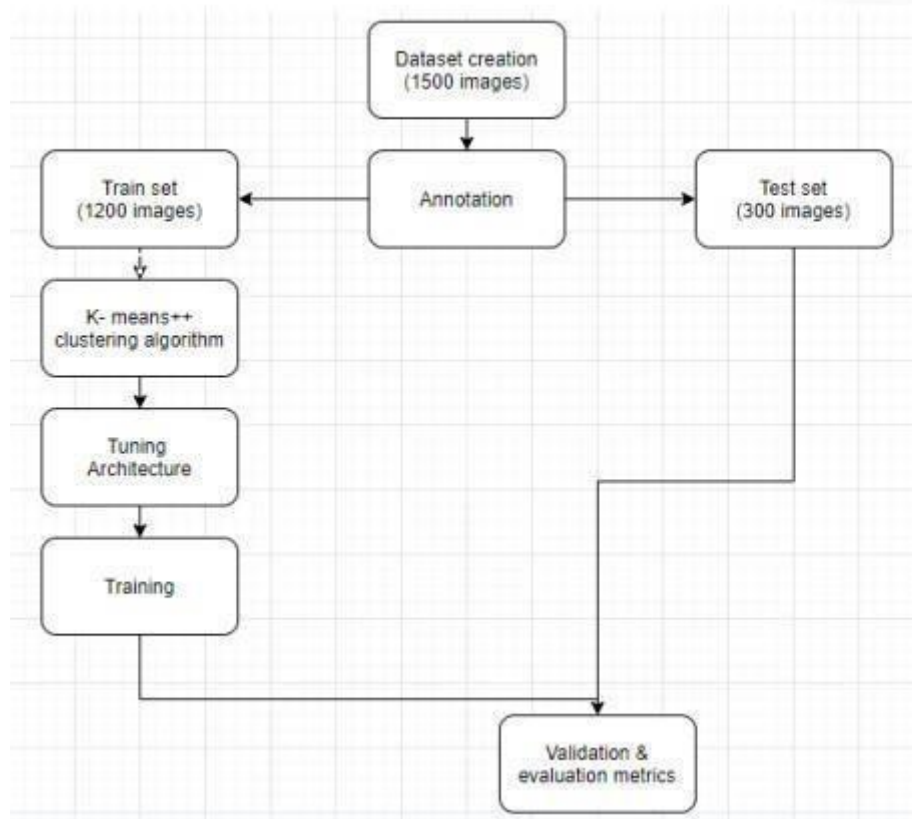


Figure 5.1 Flowchart (Base paper)

The design or flow of the project as shown in the above flow charts flows in the following way:

- Dataset is created by collecting various pothole images.
- Annotating the potholes in image with the help of GPU (graphical processing unit).
- Splitting the annotated pothole images into training set and testing test.
- The training set will go through the k-means clustering.
- YOLO makes use of bounding box “priors” or “anchors” The authors of YOLO used K-means clustering to cluster the bounding boxes of objects in the training data to determine suitable anchor box sizes.
- Gives through the turning architecture of YOLO.
- And then performing training to the set.
- And then predicting the accuracy score of the model.

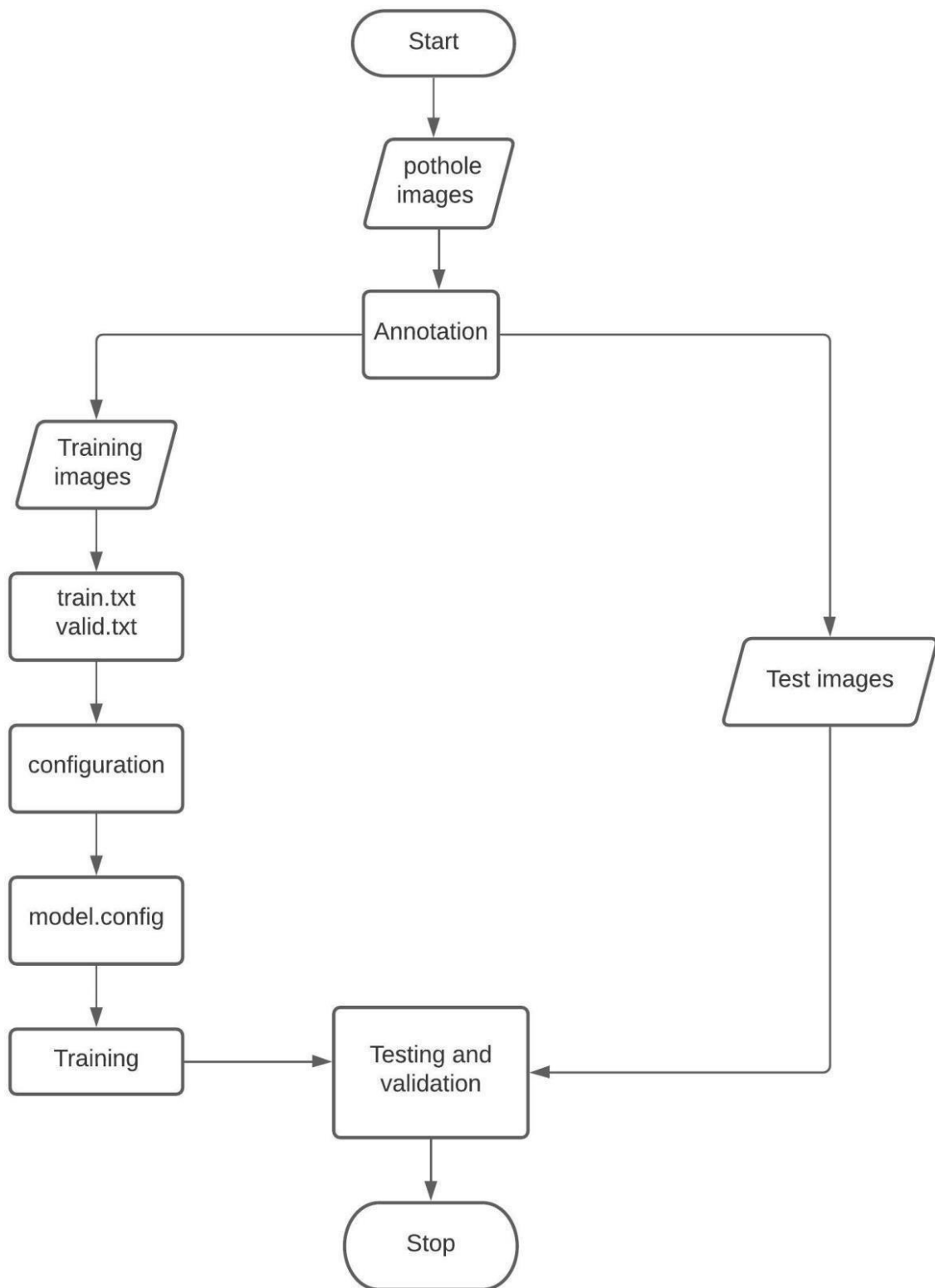


Figure 5.2 Flowchart design

The design or flow of the project as shown in the above flow charts flows in the following way:

1. Initially we need to collect the data i.e images of potholes in this project we collected 7500 pothole images from kaggle
2. Then we need to annotate the data set i.e draw the bounding boxes where the potholes exist.
3. After drawing the bounding boxes we need to divide the annotated dataset into training and test sets in order to train the model and validate the model over test set
4. Now we need to create two files namely train.txt and test.txt where all file paths will be there for each and every image, these files will act as input to our train file to train and validate.
5. Before training we need to configure all files, more about these files is discussed in IMPLEMENTATION section.
6. Now we need to configure the yolo configuration file i.e. the Convolutional neural network used to train the Yolo object detection model.
7. Now we need to start the training and ideally we need to stop the training when loss is below 0.06.
8. Validate the trained model with the test set.
9. Finish.

6. IMPLEMENTATION

In the above sections, it is all discussed about the background or the motivation behind this project: the project is to identify Potholes with various kinds of images. In this section all the steps will be given discussed:

Steps:

1. Collecting the data
2. Label the data
3. Prepare the config file
4. Prepare the text files for training and testing
5. Train the model
6. Test the model over custom images

6.1 Data Collection:

This phase is common to any machine learning or deep learning project, Data is collected from a Kaggle repo.

6.1.1 About the Dataset:

The dataset for training constitutes 7,500 images for training and 1350 images for testing the model developed. This dataset consists of repetitive images which need to be filtered out such that images in the final dataset are unique and of different features. But, certain factors in this dataset are constant such as colors of roads, illumination, and pothole vicinity do not help in making the model well versed with various real-life scenarios. Using YOLO and OpenCV, we also reduced the resolution of images to remove the constant part of images in the dataset like the dashboard and the sky.



Figure 6.1 Pothole image before predicting

6.2 Labeling the data:

In order to train the object detection model firstly after collecting the data, we need to label it using a tool i.e. we need to draw the bounding boxes and save these bounding boxes in the format of YOLO i.e. it saves the files in the `.txt` format. The tool used to label these images in 'Labeling'.

6.2.1 Installing labeling:

Goto this Github repo <https://github.com/tzutalin/labelImg> and clone this GitHub repo to the local computer using

```
'git clone https://github.com/tzutalin/labelImg.git '
```

Open this folder in the in-command prompt and install the tool using the following command

```
'python setup.py install'
```

After installation, we can launch the tool using the command 'python labeling.py'



Figure 6.2 GPU

After launching the tool we need to change it to the YOLO format of labeling as mentioned above.

After which we need to load the image directory consisting of all the images of potholes.



Figure 6.3 Annotating using GPU

The output file after labeling will be in the following format

```
1 0 0.54972826087 0.511956521739 0.0179347826087 0.00507246376812
2 0 0.410326086957 0.525362318841 0.025 0.00579710144928
3 0 0.308423913043 0.528260869565 0.0146739130435 0.00579710144928
4 0 0.165489130435 0.522463768116 0.0277173913043 0.00579710144928
5 0 0.101630434783 0.528623188406 0.0195652173913 0.00652173913043
6 0 0.0790760869565 0.529347826087 0.0157608695652 0.00797101449275
7
```

Figure 6.4 Parameters of the Annotated Pothole Image

The categories in the text file are the class of bounding boxes and the coordinates of the bounding boxes.

6.3 Training Yolo Model:

To train the Yolo model we need to collect official pre-made code by the developer Pjreddie and to collect it and install necessary packages we need to clone the GitHub repo.

```
'git clone https://github.com/pjreddie/darknet'
```

After cloning the repo navigate to the cloned repo and type 'make'. 'make' is a command which is used to install or to execute multiple bash shell commands in this makefile all the necessary installations like OpenCV build and Cuda build will be done.

After the build is done we need to download the pertained convolutional weights which are trained over imagenet dataset. To download the pre-trained weight file use the following code.

```
wget https://pjreddie.com/media/files/darknet53.conv.74
```

Before starting the training there are some files and configuration that is to be done. Create a folder named training, inside it create a file named 'trainer.data' put the following code inside it

```
classes= 1
```

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

```
names = custom/objects.names backup = backup/
```

In the train.txt file, there will be all image file paths that will be used for training, and similarly in test.txt, there will be all image paths used for validation. In objects.names file the name of the class 'Pothole'.

And the backup is a parameter where all the weights will be saved for every 1000 steps while we train the model.

After all these steps we need to modify the config file of yolo-tiny.cfg It contains the training parameters as batch size, learning rate, etc., and also the architecture of the network as a number of layers, filters, type of activation function, etc.

There are some changes that are to be done in the filters of Conv layers just before yolo layers. Change the number of filters for convolutional layer "[convolution]" just before every yolo output "[yolo]" such that the number of filters= $3 \times (5 + \#ofclasses) = 3 \times (5+1) = 18$.

Now we need to start the training.

To start the training type the following command.

```
./darknet detector train training/trainer.data training/ yolo3-tiny.cfg
darknet53.conv.74
```

```
(base) ➔ darknet git:(master) X ./darknet detector train training/obj.data training/yolo3-tiny.cfg darknet53.conv.74
yolo3-tiny
layer   filters  size      input           output
0 conv   16  3 x 3 / 1  416 x 416 x 3  -> 416 x 416 x 16  0.150 BFLOPs
1 max     2  2 x 2 / 2  416 x 416 x 16  -> 208 x 208 x 16
2 conv   32  3 x 3 / 1  208 x 208 x 16  -> 208 x 208 x 32  0.399 BFLOPs
3 max     2  2 x 2 / 2  208 x 208 x 32  -> 104 x 104 x 32
4 conv   64  3 x 3 / 1  104 x 104 x 32  -> 104 x 104 x 64  0.399 BFLOPs
5 max     2  2 x 2 / 2  104 x 104 x 64  -> 52 x 52 x 64
6 conv  128  3 x 3 / 1   52 x 52 x 64  -> 52 x 52 x 128  0.399 BFLOPs
7 max     2  2 x 2 / 2   52 x 52 x 128  -> 26 x 26 x 128
8 conv  256  3 x 3 / 1   26 x 26 x 128  -> 26 x 26 x 256  0.399 BFLOPs
9 max     2  2 x 2 / 2   26 x 26 x 256  -> 13 x 13 x 256
10 conv  512  3 x 3 / 1   13 x 13 x 256  -> 13 x 13 x 512  0.399 BFLOPs
11 max     2  2 x 2 / 1   13 x 13 x 512  -> 13 x 13 x 512
12 conv 1024  3 x 3 / 1   13 x 13 x 512  -> 13 x 13 x1024  1.595 BFLOPs
13 conv  256  1 x 1 / 1   13 x 13 x1024  -> 13 x 13 x 256  0.089 BFLOPs
14 conv  512  3 x 3 / 1   13 x 13 x 256  -> 13 x 13 x 512  0.399 BFLOPs
15 conv  18   1 x 1 / 1   13 x 13 x 512  -> 13 x 13 x 18   0.003 BFLOPs
16 yolo
17 route 13
18 conv  128  1 x 1 / 1   13 x 13 x 256  -> 13 x 13 x 128  0.011 BFLOPs
19 upsample 2x 13 x 13 x 128 -> 26 x 26 x 128
20 route 19 8
21 conv  256  3 x 3 / 1   26 x 26 x 384  -> 26 x 26 x 256  1.196 BFLOPs
22 conv  18   1 x 1 / 1   26 x 26 x 256  -> 26 x 26 x 18   0.006 BFLOPs
23 yolo
Loading weights from darknet53.conv.74...Done!
Learning Rate: 0.001, Momentum: 0.9, Decay: 0.0005
Resizing
384
Loaded: 0.069065 seconds
Region 16 Avg IOU: -nan, Class: -nan, Obj: -nan, No Obj: 0.499833, .5R: -nan, .75R: -nan, count: 0
Region 23 Avg IOU: 0.074545, Class: 0.499238, Obj: 0.499804, No Obj: 0.498700, .5R: 0.000000, .75R: 0.000000, count: 1
1: 276.251129, 276.251129 avg, 0.000000 rate, 2.543151 seconds, 1 images
Loaded: 0.000050 seconds
Region 16 Avg IOU: -nan, Class: -nan, Obj: -nan, No Obj: 0.499832, .5R: -nan, .75R: -nan, count: 0
Region 23 Avg IOU: 0.089182, Class: 0.499239, Obj: 0.499764, No Obj: 0.498698, .5R: 0.000000, .75R: 0.000000, count: 1
2: 272.783295, 275.904358 avg, 0.000000 rate, 2.466708 seconds, 2 images
Loaded: 0.000047 seconds
Region 16 Avg IOU: -nan, Class: -nan, Obj: -nan, No Obj: 0.499832, .5R: -nan, .75R: -nan, count: 0
Region 23 Avg IOU: 0.096990, Class: 0.499153, Obj: 0.499796, No Obj: 0.498699, .5R: 0.000000, .75R: 0.000000, count: 1
3: 278.743774, 276.188293 avg, 0.000000 rate, 2.469016 seconds, 3 images
Loaded: 0.000066 seconds
Region 16 Avg IOU: -nan, Class: -nan, Obj: -nan, No Obj: 0.499832, .5R: -nan, .75R: -nan, count: 0
Region 23 Avg IOU: 0.105919, Class: 0.499192, Obj: 0.499834, No Obj: 0.498697, .5R: 0.000000, .75R: 0.000000, count: 2
4: 278.832764, 276.452728 avg, 0.000000 rate, 2.699542 seconds, 4 images
Loaded: 0.000063 seconds
Region 16 Avg IOU: -nan, Class: -nan, Obj: -nan, No Obj: 0.499833, .5R: -nan, .75R: -nan, count: 0
Region 23 Avg IOU: 0.060707, Class: 0.499143, Obj: 0.499816, No Obj: 0.498698, .5R: 0.000000, .75R: 0.000000, count: 1
5: 276.213562, 276.428802 avg, 0.000000 rate, 2.461815 seconds, 5 images
Loaded: 0.000064 seconds
Region 16 Avg IOU: -nan, Class: -nan, Obj: -nan, No Obj: 0.499833, .5R: -nan, .75R: -nan, count: 0
Region 23 Avg IOU: 0.074637, Class: 0.499169, Obj: 0.499822, No Obj: 0.498698, .5R: 0.000000, .75R: 0.000000, count: 1
6: 277.242920, 276.510223 avg, 0.000000 rate, 2.523788 seconds, 6 images
```

Figure 6.5 Output Architecture

6.4 Testing and Validating Results:

To test the trained model over custom images there is a format / code that is to be executed to test the model.

Using the following code we can test the trained model: `./darknet detector test`

`training/obj.data` `training/yolov3-tiny.cfg` `yolov3-tiny_final.weights`

`CCPLX_G0089833.jpg`



Figure 6.6 Examples of prediction.jpg



Figure 6.7 Example of predictions.jpg

```
%ls  
drive/ sample_data/
```

- Checking list of directories.

```
%cd drive/My Drive/potholes/darknet  
/content/drive/My Drive/potholes/darknet
```

- Changing the directory to the potholes folder and then to the darknet file.

```
!make
```

```
make: Nothing to be done for 'all'.
```

- Make is a command line tool which allows you to download the latest version of popular developer tools on your installation, installing it alongside all of the required dependencies

```
!chmod +x darknet
```

- Chmod is used to change the access permission of file systems.

```
[ ] from IPython.display import display, Image  
display(Image(filename='pot.jpg'))
```



Figure 6.8 input image

- To display the input image before applying YOLO.

```

!./darknet detector test training/obj.data training/yolov3-tiny.cfg yolov3-tiny_final.weights pot.jpg

```

layer	filters	size	input	output	
0 conv	16	3 x 3 / 1	416 x 416 x 3	416 x 416 x 16	0.150 BFLOPs
1 max		2 x 2 / 2	416 x 416 x 16	208 x 208 x 16	
2 conv	32	3 x 3 / 1	208 x 208 x 16	208 x 208 x 32	0.399 BFLOPs
3 max		2 x 2 / 2	208 x 208 x 32	104 x 104 x 32	
4 conv	64	3 x 3 / 1	104 x 104 x 32	104 x 104 x 64	0.399 BFLOPs
5 max		2 x 2 / 2	104 x 104 x 64	52 x 52 x 64	
6 conv	128	3 x 3 / 1	52 x 52 x 64	52 x 52 x 128	0.399 BFLOPs
7 max		2 x 2 / 2	52 x 52 x 128	26 x 26 x 128	
8 conv	256	3 x 3 / 1	26 x 26 x 128	26 x 26 x 256	0.399 BFLOPs
9 max		2 x 2 / 2	26 x 26 x 256	13 x 13 x 256	
10 conv	512	3 x 3 / 1	13 x 13 x 256	13 x 13 x 512	0.399 BFLOPs
11 max		2 x 2 / 1	13 x 13 x 512	13 x 13 x 512	
12 conv	1024	3 x 3 / 1	13 x 13 x 512	13 x 13 x 1024	1.595 BFLOPs
13 conv	256	1 x 1 / 1	13 x 13 x 1024	13 x 13 x 256	0.089 BFLOPs
14 conv	512	3 x 3 / 1	13 x 13 x 256	13 x 13 x 512	0.399 BFLOPs
15 conv	18	1 x 1 / 1	13 x 13 x 512	13 x 13 x 18	0.003 BFLOPs
16 yolo					
17 route	13				
18 conv	128	1 x 1 / 1	13 x 13 x 256	13 x 13 x 128	0.011 BFLOPs
19 upsample		2x	13 x 13 x 128	26 x 26 x 128	
20 route	19 8				
21 conv	256	3 x 3 / 1	26 x 26 x 384	26 x 26 x 256	1.196 BFLOPs
22 conv	18	1 x 1 / 1	26 x 26 x 256	26 x 26 x 18	0.006 BFLOPs
23 yolo					

```

Loading weights from yolov3-tiny_final.weights...Done!
pot.jpg: Predicted in 0.006208 seconds.
Pothole: 95%
Unable to init server: Could not connect: Connection refused

(predictions:258): Gtk-WARNING **: 08:06:39.492: cannot open display:

```

Figure 6.9 Architecture

- Detector command is used on image to detect the potholes in the image. The input images goes through the various layer like convolutional layer, max pooling layer, up sampling layer etc.

```
[ ] from IPython.display import display, Image
    display(Image(filename='predictions.jpg'))
```



Figure 6.10 predictions.jpg

- Output of the predicted pothole image.

6.5 Code to Run the project on a live camera

```
import cv2 import time
```

```
import numpy as np import core.utils as utils import tensorflow as tf from PIL import Image
```

```
return_elements = ["input/input_data:0", "pred_sbbox/concat_2:0",  
"pred_mbbox/concat_2:0", "pred_lbbox/concat_2:0"]
```

```
pb_file          = "./yolov3_coco.pb"
```

```

video_path      = "./docs/images/road.mp4" # video_path      = 0
num_classes     = 80
input_size      = 416 graph = tf.Graph()
return_tensors = utils.read_pb_return_tensors(graph, pb_file, return_elements)

```

```

with tf.Session(graph=graph) as sess: vid = cv2.VideoCapture(video_path) while True:
    return_value, frame = vid.read() if return_value:
        frame      =      cv2.cvtColor(frame,      cv2.COLOR_BGR2RGB)      image      =
        Image.fromarray(frame)
    else:
        raise ValueError("No image!") frame_size = frame.shape[:2]
    image_data =  utils.image_preporcess(np.copy(frame),  [input_size,  input_size])
    image_data = image_data[np.newaxis, ...]
    prev_time = time.time()

```

```

pred_sbbox, pred_mbbox, pred_lbbox = sess.run( [return_tensors[1], return_tensors[2],
    return_tensors[3]],
    feed_dict={ return_tensors[0]: image_data})

```

```

pred_bbox = np.concatenate([np.reshape(pred_sbbox, (-1, 5 + num_classes)),
    np.reshape(pred_mbbox, (-1, 5 + num_classes)),
        np.reshape(pred_lbbox, (-1, 5 + num_classes))], axis=0) bboxes =
    utils.postprocess_boxes(pred_bbox,  frame_size,  input_size,  0.3) bboxes =
    utils.nms(bboxes, 0.45, method='nms')
    image = utils.draw_bbox(frame, bboxes)

```

```

curr_time = time.time()
exec_time = curr_time - prev_time

```

```
result = np.asarray(image)
info    = "time:  %.2f  ms"  %(1000*exec_time)  cv2.namedWindow("result",
cv2.WINDOW_AUTOSIZE) result = cv2.cvtColor(image, cv2.COLOR_RGB2BGR)
cv2.imshow("result", result)
if cv2.waitKey(1) & 0xFF == ord('q'): break.
```

CONCLUSION

Detection of potholes is special when compared to other artifacts. Detections such as a person, a vehicle, an aircraft, etc. Unlike other organizations, Things like potholes have no fixed form. It makes it this way Struggling for detection. Raising the scoring average Due to the aforementioned limitations, accuracy for the identification of potholes is challenging.

In addition, by making appropriate architectural improvements, the mean average accuracy is increased. In the future, using a raspberry pi with a camera, the device can be applied in real-time in a car dashboard. The device can also be incorporated with a GPS[18] to track the position of the observed potholes, which can be a perfect use for the road maintenance squad.

- [1] Geronimo, David, et al. "Survey of pedestrian detection for advanced driver assistance systems." IEEE transactions on pattern analysis and machine intelligence 32.7 (2009): 1239-1258
- [2] Shaout, Adnan, Dominic Colella, and S. S. Awad. "Advanced driver assistance systems-past, present and future." 2011 Seventh International Computer Engineering Conference (ICENCO'2011). IEEE, 2011.
- [3] Bengler, Klaus, et al. "Three decades of driver assistance systems: Review and future perspectives." IEEE Intelligent transportation systems magazine 6.4 (2014): 6- 22.
- [4] <https://www.indiatoday.in/india/story/over-9300-deaths-25000-injured-in-3-years-due-to-potholes-1294147-2018-07-24>
- [5] Kim, Taehyeong, and Seung-Ki Ryu. "Review and analysis of pothole detection methods." Journal of Emerging Trends in Computing and Information Sciences 5.8 (2014): 603-608.
- [6] Du, Juan. "Understanding of Object Detection Based on CNN Family and YOLO." Journal of Physics: Conference Series. Vol. 1004. No. 1. IOP Publishing, 2018.
- [7] Koch, Christian, and Ioannis Brilakis. "Pothole detection in asphalt pavement images." Advanced Engineering Informatics 25.3 (2011): 507- 515.
- [8] Huidrom, Lokeshwor, Lalit Kumar Das, and S. K. Sud. "Method for automated assessment of potholes, cracks and patches from road surface video clips." Procedia-

Social and Behavioral Sciences 104.2013 (2013): 312-321.

[9] Karthika, R., and Latha Parameswaran. "An automated vision-based algorithm for out of context detection in images." *International Journal of Signal and Imaging Systems Engineering* 11.1 (2018): 1-8.

[10] Girshick, Ross, et al. "Region-based convolutional networks for accurate object detection and segmentation." *IEEE transactions on pattern analysis and machine intelligence* 38.1 (2015): 142-158.

[11] Girshick, Ross. "Fast r-cnn." *Proceedings of the IEEE international conference on computer vision*. 2015.

[12] Ren, Shaoqing, et al. "Faster r-cnn: Towards real-time object detection with region proposal networks." *Advances in neural information processing systems*. 2015.

[13] Chebrolu, Koti Naga Renu, and P. N. Kumar. "Deep Learning based Pedestrian Detection at all Light Conditions." *2019 International Conference on Communication and Signal Processing (ICCSP)*. IEEE, 2019.

[14] Liu, Wei, et al. "Ssd: Single shot multibox detector." *European conference on computer vision*. Springer, Cham, 2016.

[15] Redmon, Joseph, et al. "You only look once: Unified, real-time object detection." *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016.

[16] Redmon, Joseph, and Ali Farhadi. "YOLO9000: better, faster, stronger." *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2017.

[17] Redmon, Joseph, and Ali Farhadi. "Yolov3: An incremental improvement."

arXiv preprint arXiv:1804.02767 (2018).

[18] Madli, Rajeshwari, et al. "Automatic detection and notification of potholes and humps on roads to aid drivers." IEEE sensors journal 15.8 (2015): 4313-4318.

[19] <https://pjreddie.com/darknet/yolo/>.

[https://towardsdatascience.com/yolo-you-only-look-once-real-time-object-detecti
on- explained-492dc9230006](https://towardsdatascience.com/yolo-you-only-look-once-real-time-object-detection-explained-492dc9230006)

