PERFORMING POTHOLE DETECTION USING CONVOLUTIONAL NEURAL NETWORKS

Richard Hanxu
Student# 1008025726
richard.hanxu@mail.utoronto.ca

Satvick Acharya
Student# 1007984512
satvick.acharya@mail.utoronto.ca

Jacob Johannsson
Student# 1008299468
jacob.johannsson@mail.utoronto.ca

Erfan Yeganehfar
Student# 1008288586
erfan.yeganehfar@mail.utoronto.ca

—-Total Pages: 9

1 INTRODUCTION

Potholes can damage vehicles, cause accidents, and increase maintenance costs. Traditional methods of detecting potholes, such as manual inspection, can be time-consuming, labor-intensive, and may miss smaller potholes or those in hard-to-reach locations. Potholes and other road deformities also pose a threat to new autonomous vehicles that may not have vision systems that are robust enough to detect and avoid these obstacles. Therefore, there is a need for more efficient and effective methods of detecting potholes. Deep learning models such as Convolutional Neural Networks (CNNs) have shown success in a wide range of computer vision tasks, including object detection, image classification, and segmentation. Thus, there exists the potential to create a deep learning based solution for pothole detection.

2 BACKGROUND AND RELATED WORKS

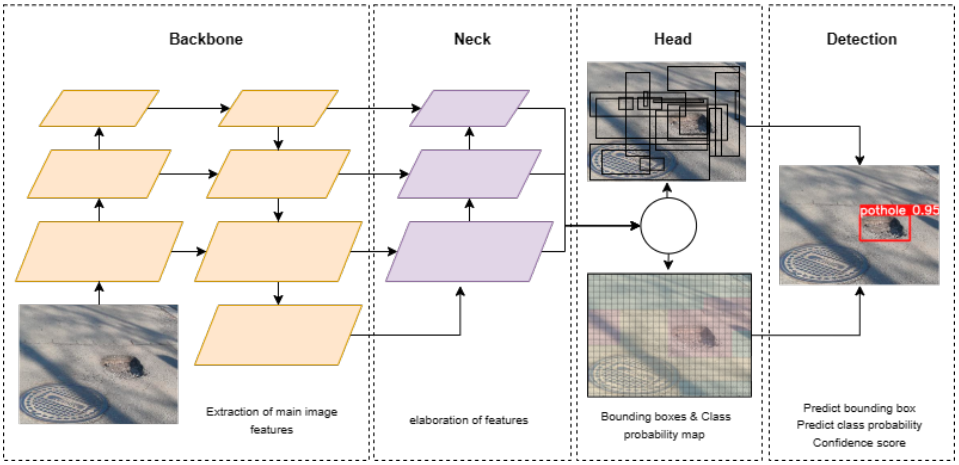


Figure 1: A high-level diagram of the YOLOv5 architecture

A wide array of techniques have already been applied to the general problem of object detection and classification. For example, Scale Invariant Feature Transform (SIFT) is a computer-vision technique that is invariant to translations, rotations, and scaling transformations, and uses key points to help isolate and identify images (Lowe, 2004). Researchers at Case Western Reserve have also applied stereo vision algorithms to detect potholes (analyzing multiple vantages of a scene for 3D information extraction) with respectable success (Li et al., 2018). RCNNS (Recurrent Convolutional Neural Networks with Spatial Attention) is an object detection algorithm developed using a recurrent

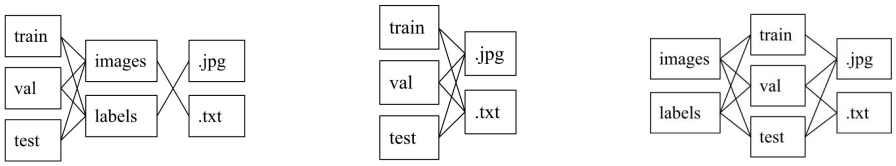
neural network and spatial attention to selectively focus on different regions of an image and improve the accuracy of object detection Girshick et al. (2014). Single Shot Detectors (SSD) is another object detection algorithm developed in 2016. It uses a single convolutional neural network to detect objects in images by predicting the probability of object presence and the corresponding bounding box coordinates in a single pass of the network Liu et al. (2016).

Our model will incorporate a specific model architecture called YOLO (You Only Look Once), to identify and locate potholes from various vantage points (as seen in Figure 1). Released in 2014, YOLO is a deep learning model which uses elements of CNNs to detect objects in images (Redmon et al., 2016).

3 Data Processing

Three separate online datasets were chosen for this project, totaling 2492 total images. The first dataset contained 1362 road images in surburban locations, taken taken by a drone at a bird's eye view Silva et al. (2020). The second dataset with 665 images contained images of urban locations taken from street-level Chitholian (2020). The last dataset with 465 images was also taken from human-height in urban and suburban locations Roboflow (2023).

While downloading the datasets, the 2 smaller datasets were organized in YOLOv3 format (Figure 2a), while the larger dataset was organized in its own unique format (Figure 2b). Thus, a program in Python was written (Johannsson (2023b)) to convert the file structures and make them compatible with the YOLOv5 format (Figure 2c) (the code was slightly different between the different datasets to account for the difference in formatting). To speed up the network and make all the images consistent, all images were compressed to a size of (400, 300) using the Pillow python module. Due to the normative nature of the .txt files, there was no need to edit those files to compensate for the image transformation. Each image was also renamed using a renaming convention of "yolo_#.jpg" or "yolo_#.txt" for the corresponding .jpg and .txt file, where # ranged from 0000 to 2491 for all 2492 samples. Each image received their own bounding boxes on each pothole (Figure 3a, with the corresponding .txt file containing the information for the bounding boxes (Figure 3b).



(a) YOLOv3 format (b) Unknown format (c) YOLOv5 format

Figure 2: The hierarchical organization of the YOLOv3, YOLOv5, and the unknown formats.

```
0 0.46796875 0.384375 0.2 0.14140625
0 0.5046875 0.5453125 0.18203125 0.1234375
0 0.70703125 0.42109375 0.21640625 0.128125
```

(a) Example of multiple potholes (b) .txt file for the corresponding bounding boxes

Figure 3: Image/label combination Source: Roboflow (2023)

To simulate a variety of environments to make our model more robust, we then applied data augmentation using the Pillow module in Python (Johannsson (2023b)). First, the RGB values were modified such that blue and green were heated up (multiplied by a factor of 1.10 and 1.05, respectively) and red was cooled down (multiplied by a factor of 0.85). Then, Pillow ImageEnhancer was

used to randomly set image contrast, sharpness, colour, and brightness to a value between 0.5 and 1.5, where 1 represents the original image. The image was also flipped over the horizontal axis. One example of a resulting transformation is shown in (Figure 4). The new data was saved following the same naming convention as before. Overall, this method was used to double the dataset size, which was then finally split into train/val/test sets using an 80-10-10 split. Including the augmented data, there were 3984 train samples, 500 validation samples, and 500 test samples. The number of potholes per image was also distributed with a mean of 2.565, with a reasonable portion containing even more. Thus, it was qualitatively determined that that there was sufficient variety in the number of potholes per image.

Figure 4: Comparison between the sourced, and the augmented image, to increase data. This is image 0164, referred to with the nomenclature previously mentioned

Some challenges faced were the differing formats (YOLOv3, and unknown format with the large dataset), which were incompatible with YOLOv5. This required sorting each file individually into the correct location. Additionally, finding a suitable dimension for the input photos required compromises, as the non-fitting examples had to be squished/stretched to the final dimension. The largest dataset all contained images with 4x3 aspect ratios, and many of the other images were either perfect square or slightly wider than tall, so it made sense to convert to 4x3.

4 BASELINE MODEL

The baseline model will be based on classical computer vision techniques. This section describes the basic structure of such a program along with the relevant functions from the OpenCV Python library. We will then discuss some preliminary results and findings regarding the implementation process.

After reading an image, the program will convert the image to gray scale using `cv2.cvtColor()`, since color information is not as relevant to the task of pothole detection. Next, a Gaussian blur (`cv2.GaussianBlur()`) is applied in order to reduce the detail of the image and avoid artefacts in our detection. Then, Canny edge detection (`cv2.Canny()`) and a dilation kernel (`cv2.dilate()`) are used to extract the outer edges of the potholes. Finally, the function `cv2.findContours()` is used to separate groups of edges and `cv2.boundingRect()` determines the coordinates of the bounding box for each contour. A visualization of the approach can be found in Figure 5.

When tested on the test data set, the baseline model achieved an average F1 score of 0.41 and an average IOU score of 0.45, indicating a fair amount of consistency with the ground truth labels but a low accuracy, as indicated by the large number of very low F1 scores (see Figure 6). The code for the baseline model can be found here. The file `main.py` is the model itself, `pred_baseline.py` generates the predictions for a directory of images using the baseline model and `eval_baseline.py` compares the baseline model output to the ground truth labels to generate IOU and F1 scores and evaluate performance.

Qualitatively, the model is able to identify potholes in most images with a high accuracy, however, since it relies on contour detection, it often produces multiple bounding boxes for a single pothole. This is because it is very rare for a pothole to be recognized as a single contour. As a result, there tends to be a great disparity in the number of bounding boxes generated by the baseline and the

number of boxes in the ground truth. The model also tends to recognize the boundaries of the potholes because that is where the contours are the most defined.

One challenge faced by the team with regards to the baseline model was that for some images, the number of bounding boxes produced by the baseline and the number of boxes in the ground truth were very different. This made calculating IOU and F1 scores more difficult since cases like this or cases where one label has no bounding boxes needed to be treated differently. Another challenge faced by the team was spurious detections from the baseline model. Since the code performs contour detection in order to segment individual potholes, even smaller edges in the road could get picked up by the program and classified as a contour. In order to combat this, the team filtered out contours which were smaller than 50 pixels squared in area.

Because the baseline model uses a classical computer vision approach, there are a number of tuneable thresholds and coefficients which have an impact on the model performance. Since the ideal values for these thresholds is highly dependent on the data set, in the future, the group plans to experiment with different coefficient and threshold values and hyperparameters.

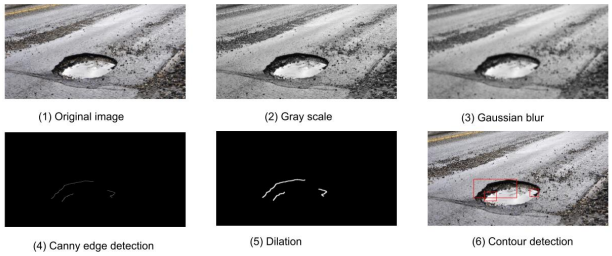


Figure 5: Baseline model performance on sample image with intermediate steps.

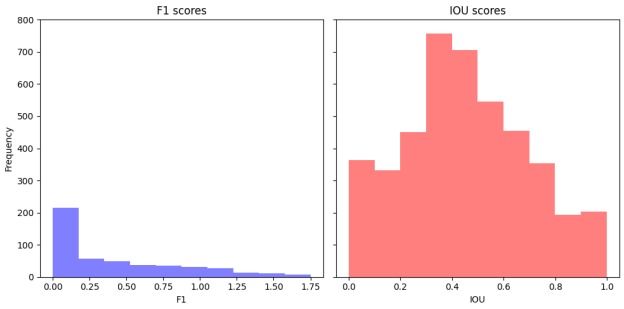


Figure 6: IOU and F1 score distribution for baseline model.

5 Architecture

As stated earlier, our group decided to build our model off YOLOv5; You Only Look Once. In this section we provide a high-level perspective of the facets that make YOLOv5 unique in its architecture and effectiveness in detecting objects. YOLO can be broken down into two primary sections, the backbone, where feature extraction takes place, and the detection head, where bounding box predictions are made.

5.1 Backbone

All iterations of YOLO pass the input image through a backbone, a section of the network designed to extract representations of spatial features through feature maps. As a result, backbones typically consist of pretrained CNNs such as DarkNet or EfficientNet due to their demonstrated profiency.

Our backbone's exact architecture is shown below (Figure 7a). The blocks labelled C3 and SPPF represent *Cross Stage Partial* and *Spatial Pyramid Pooling* blocks respectively. CSP blocks are a unique arrangement of convolutions and concatenation connections that were introduced with CSPNet, a CNN that has outperformed models including ResNet and Darknet. Spatial Pyramid Pooling Layers represent a concatenation of feature maps created from convolutions with different-sized kernels, using padding so that every feature map possesses the same output resolution as the input.

```
# YOLOv5 v6.0 backbone
backbone:
  # [from, number, module, args]
  [[-1, 1, Conv, [64, 6, 2, 2]],  # 0-P1/2
   [-1, 1, Conv, [128, 3, 2]],  # 1-P2/4
   [-1, 3, C3, [128]],
   [-1, 1, Conv, [256, 3, 2]],  # 3-P3/8
   [-1, 6, C3, [256]],
   [-1, 1, Conv, [512, 3, 2]],  # 5-P4/16
   [-1, 9, C3, [512]],
   [-1, 1, Conv, [1024, 3, 2]],  # 7-P5/32
   [-1, 3, C3, [1024]],
   [-1, 1, SPPF, [1024, 5]],  # 9
  ]
```

(a) Our YOLOv5 backbone architecture. The number of channels at each component expands from 3 (input) to 1024.

```
# YOLOv5 v6.0 head
head:
  [[-1, 1, Conv, [512, 1, 1]],
   [-1, 1, nn.Upsample, [None, 2, 'nearest']],
   [[-1, 6], 1, Concat, [1]],  # cat backbone P4
   [-1, 3, C3, [512, False]],  # 13

   [-1, 1, Conv, [256, 1, 1]],
   [-1, 1, nn.Upsample, [None, 2, 'nearest']],
   [[-1, 4], 1, Concat, [1]],  # cat backbone P3
   [-1, 3, C3, [256, False]],  # 17 (P3/8-small)

   [-1, 1, Conv, [256, 3, 2]],
   [[-1, 14], 1, Concat, [1]],  # cat head P4
   [-1, 3, C3, [512, False]],  # 20 (P4/16-medium)

   [-1, 1, Conv, [512, 3, 2]],
   [[-1, 10], 1, Concat, [1]],  # cat head P5
   [-1, 3, C3, [1024, False]],  # 23 (P5/32-large)

   [[17, 20, 23], 1, Detect, [nc, anchors]],  # Detect(P3, P4, P5)
  ]
```

(b) Our YOLOv5 detection head architecture. The last line describes at which layers detection is actually performed, which also takes place at different resolutions.

Figure 7: YOLOv5 Model Architecture

5.2 Detection Head

The detection head is where bounding boxes and labels for each object are generated. The first YOLO implementation segmented an image into an NxN grid, generating the following vector for each cell: $[b_x, b_y, b_w, b_h, c_o, c_1, c_2, ... c_n]$. c_o represents the models confidence in the existence of an object whose bounding box center lies in that cell, $c_1, c_2, ...c_n$ are a softmax distribution of the different possible classes, and b_x, b_y, b_w, b_h represent the (x, y) coordinates and the width/height of the bounding box *globally* (meaning that the bounding box can span past the area of the cell), and are all floats between 0 and 1.

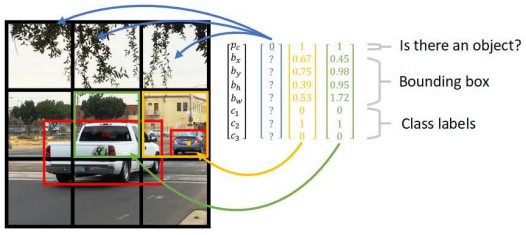


Figure 8: A visualization of the NxN grid partitioning performed by YOLO during object detection, as well as the vector output

The learning/loss function is a sum of 3 separate metrics: the predicted versus actual confidence of the existence of an object in the cell, the accuracy of the predicted versus the ground truth bounding box (only added if there was an object whose center existed in that cell), and the predicted versus actual class probability (only added if there was an object whose center existed in that cell). MSE is used for each of these functions, although the square roots of the widths and heights are taken rather than their actual values.

YOLOv5 features several improvements to the original implementation. The most notable ones include storing prior bounding boxes to assist in bounding box predictions (YOLOv2), gained ability to predict the existence of multiple objects per cell (YOLOv2), and varying grid resolutions to

increase effectiveness in capturing both large and small objects with ease (YOLOv3). Our detection head architecture is shown below.

5.3 Producing the Final Model

In training our model, we had 5 YOLOv5 network presets to choose from, each having a different number of parameters. We ended up choosing the YOLOv5x (extra large) model, with 445 layers, and 86,217,814 parameters. As for our other hyperparameters, we chose batch size of 16, an initial learning rate of 0.00175, a final learning rate of 0.000175, SGD for our optimization function, a momentum of 0.937, and a weight decay of 0.005. These parameters were chosen based on mean average precision IOU (mAP) and F1 scores, and the results are fully explored in Johannsson (2023a).

5.4 Results

Our model contains 445 layers with just over 86 million trainable parameters. The size of this model meant that an instance with access to a GPU was required for training. We trained our model on 100 epochs of the data (roughly 3 hours), freezing the backbone weights. The model was trained by cloning the and following the instructions to use a pretrained model with a custom dataset.

Since our model predicted bounding boxes, we chose to evaluate performance of our fully-trained model quantitatively using IOU and F1. IOU is a metric used in YOLO models which divides the area of intersection of the prediction and ground truth by the total area (see Figure 9). F1, on the other hand, provides insight into model performance by representing perfect precision and recall with a score of 1.

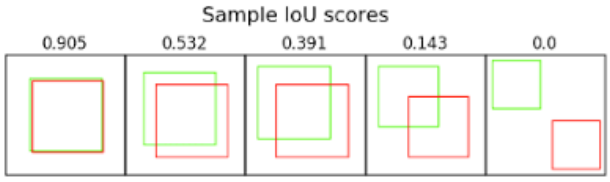
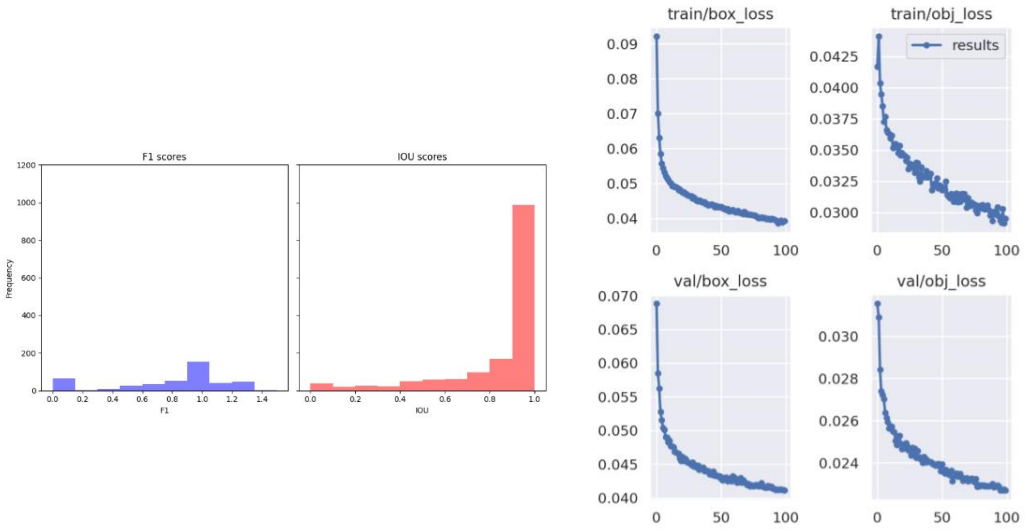


Figure 9: Visualization of different IOU scores.

The concave-up decrease between epochs in binary cross entropy and MSE losses for the training and validation sets indicates effective learning. During training, we also logged mAP, which stands for mean average precision and represents the area under the precision-recall curve (a value between 0 and 1, where 1 represents perfect detection and 0 represents no detection) (see Figure 10b). Using a threshold of 0.5, we obtained relatively good performance, indicating that the model is able to tell roughly in which regions of the image the potholes are located, but not to a great degree of accuracy.

At inference time, bounding boxes with a confidence level greater than 0.25 were isolated and merged in overlapping regions to produce the final bounding boxes. When tested on the data test set, the model achieved an average F1 score of 0.80 and an average IOU score of 0.84, drastically outperforming the baseline model. Qualitatively, the model performed well over a variety of different scenes with different quantities and appearances of potholes, even in scenes with disturbances such as shadows.



(a) IOU and F1 score distribution for YOLO model.

(b) Binary Cross-entropy (obj) and MSE (box) loss curves for training and test datasets.

Figure 10: Loss diagrams of our final model

Conversely, due to the wide range of potential colors of the pavement containing the pothole, the model seemed to have difficulty learning how to detect potholes in images where there is less contrast between the potholes and the surrounding pavement. Moreover, the model seems to be susceptible to false positives in more "distracting" images with other objects such as grass and manhole covers, which is can be mistaken for potholes (see Figure 11).

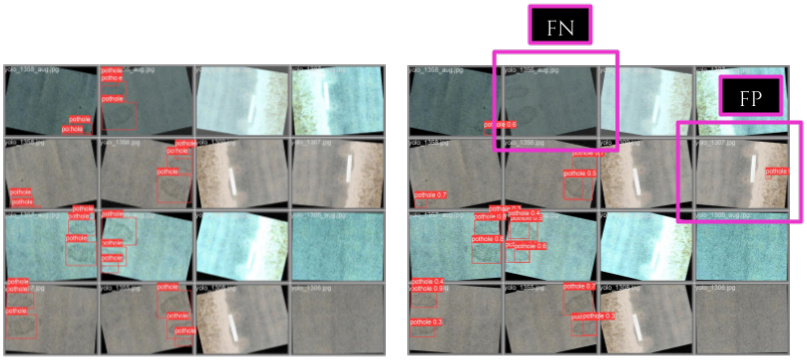


Figure 11: Binary Cross-entropy (obj) and MSE (box) loss curves for training and test datasets.

Some challenges with creating this model were reorganizing the dataset to fit the required format for the pretrained model, figuring out how to recreate their tutorial with different data, and learning how to navigate the complexity and manipulate the cloned repository.

6 Evaluation on Unseen Data

Though the test set was unseen at the time of evaluation (results in section 5.4), our group still wanted to get a sense of how well our model would perform on the streets of Toronto, say if it was attached to a dashcam for a car or bike. To do this, our group manually took 27 photos of potholes from the streets of Toronto to evaluate them with the model.

Figure 12: A handful of manually taken images, and the corresponding model's prediction on where the pothole is in the image, along with the probability of it being a pothole.

As seen in Figure 12, the model performed as well as a human would on close-up data, drawing tight boxes around areas that our group also considers potholes. However (and as discussed in section 7), the model's performance noticeably drops for far-away instances of potholes, or an ambiguous collection of information in a photo, like dirt or puddles. Since our model performance scores don't qualitatively describe how well the model is performing on our dataset, we will instead visually compare our model's outputs to outputs of state-of-the-art pothole detection software. Figure 13 shows an example image from CityRover's pothole detection model. Visually, in comparing this to Figure 12, our model's performance is slightly worse but similar to that of CityRover's implementation.

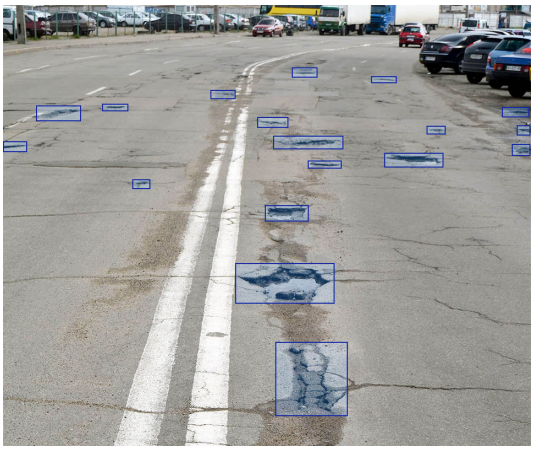


Figure 13: City rover's pothole detection implementation Source: CityRover (2023)

7 Discussion

Overall, our deep learning model significantly outperformed our baseline. It was very accurate at predicting potholes in almost every domain, reaching levels similar to a normal human being. Besides the common traits possessed by all CNNs (convolutions, encoding) we believe this high performance was due to the YOLOv5 architecture's incorporation of multiple detection heads, which enabled it to learn multiple relationships at different size scales.

Our group also observed that our model sometimes incorrectly classified background objects as potholes (Figure 14), such as puddles and dark regions. This was likely due to our datasets which contained mostly potholes at close range. Our datasets also only contained images taken in sunny/fair weather; our model would likely suffer a large decrease in performance on images taken in other conditions such as rain, snow, or at night.

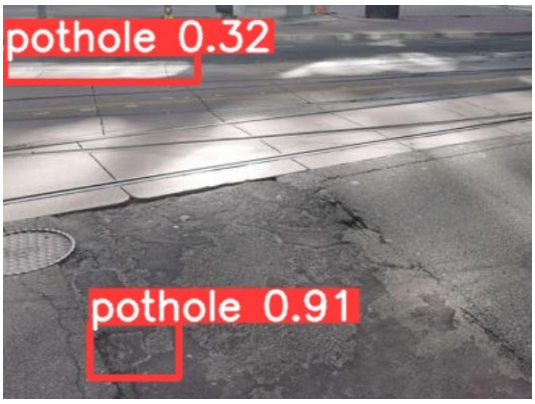


Figure 14: Our model mistakenly detects a pothole at long distance

However, something to consider is that for the purpose of pothole detection in vehicles is that the most important potholes to detect are close-up potholes and therefore, our model is still prioritizing the correct range.

8 Ethical Considerations

As pothole detection models become paired with/integrated into self-driving systems, biases in evaluation and deployment need to be taken into consideration. With respect to evaluation biases, datasets used to train/validate the model need to contain a wide and balanced variety of environments and scenarios, in order for it to perform at a high level in the real world. One possible deployment bias exists where pothole detection models are used as collision avoidance tools; this could cause the vehicle to behave erratically, endangering the lives of pedestrians and the passengers inside.

In addition, although our group used the YOLO architecture to identify potholes on the road, the strength of YOLO in a variety of contexts raises the questions of scenarios where similar models could be used for morally ambiguous purposes. One example is in the military where a YOLO model has been shown to effectively identify tanks and other defence-related objects (Ali et al. (2023)). As artificial intelligence becomes prevalent in more fields, greater care needs to be taken to ensure that its applications are for the benefit of society.

9 Project Difficulty

The problem our group chose to address required us to carefully plan out our data and choice of model. Potholes can take many different shapes and forms, and the environment in which they are found can have a large impact on well they are detected. Thus, we believe our model performed extremely well considering the frontier we selected. It was able to identify potholes at a high degree of accuracy over wide variety of perspectives, image qualities, and locations.

The concept of object detection was also an interesting challenge for our group to understand and implement. At the beginning of the project, the majority of our group's knowledge in image classification, CNNs and encoders/decoders, was that which was gained in the course. As a result, our group had to put in additional time and effort into understanding how YOLO and its descendants learned to efficiently predict bounding boxes.

There were still many other challenges our group had to overcome throughout the course of this project. These included writing code to augment and preprocess our datasets to be used with the YOLOv5 architecture, designing a baseline model in computer vision, and learning how to wield/modify the pretrained YOLOv5 model.

REFERENCES

Sikandar Ali, Abdullah, Ali Athar, Maisam Ali, Ali Hussain, and Hee-Cheol Kim. Computer vision-based military tank recognition using object detection technique: An application of the yolo framework. In *2023 1st International Conference on Advanced Innovations in Smart Cities (ICAISC)*, pp. 1–6, 2023. doi: 10.1109/ICAISC56366.2023.10085552.

Atikur Rahman Chitholian. Pothole object detection dataset - raw, Nov 2020. URL https://public.roboflow.com/object-detection/pothole/1.

CityRover. Cityrover- detect potholes using cityrover ai technology, 2023. URL https://www.cityrover.com/.

Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *2014 IEEE Conference on Computer Vision and Pattern Recognition*, pp. 580–587, 2014. doi: 10.1109/CVPR.2014.81.

Jacob Johannsson. Hyperparameter testing. April 14th, 2023a. URL https://docs.google.com/document/d/18n-YTE1b_5vvf2NlMLUG7G4AuZF1tORXPJMw1mIeLbI/edit?usp=sharing.

Jacob Johannsson. data_processing, 2023b. URL https://github.com/JacobJohannsson/APS360-Pothole-Detection/blob/main/data_processing.ipynb.

Yaqi Li, Christos Papachristou, and Daniel Weyer. Road pothole detection system based on stereo vision. In *NAECON 2018 - IEEE National Aerospace and Electronics Conference*, pp. 292–297, 2018. doi: 10.1109/NAECON.2018.8556809.

Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, and Alexander C. Berg. SSD: Single shot MultiBox detector. In *Computer Vision – ECCV 2016*, pp. 21–37. Springer International Publishing, 2016. doi: 10.1007/978-3-319-46448-0_2. URL https://doi.org/10.1007%2F978-3-319-46448-0_2.

David G. Lowe. Distinctive image features from scale-invariant keypoints. *Int. J. Comput. Vision*, 60(2):91–110, nov 2004. ISSN 0920-5691. doi: 10.1023/B:VISI.0000029664.99615.94. URL https://doi.org/10.1023/B:VISI.0000029664.99615.94.

Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection, 2016.

Roboflow. Pothole detection object detection dataset (v1, 2023-02-08 4:54pm) by drone, Feb 2023. URL https://universe.roboflow.com/drone-zh0ho/pothole-detection-zdizt/dataset/1.

Luís Augusto Silva, Héctor Sanchez San Blas, David Peral Peral García, André Sales Sales Mendes, and Gabriel Villarubia Villarubia González. An architectural multi-agent system for a pavement monitoring system with pothole recognition in uav images. *Sensors*, 20(21), 2020. URL https://www.mdpi.com/1424-8220/20/21/6205.