Pothole Detection App

Avinash Nair, Jacob Guiang, Zane Sanchez, Brady Cash
The University of Texas at Dallas
800 W. Campbell Road, Richardson, TX

Abstract

This paper will discuss our pothole detection application, which is meant to detect and keep track of potholes in real time. The application required two parts: training a model to detect potholes, and building a frontend to display and keep track of the model's predictions. The model used was the Ultralytics YOLO v8 model. We used a Roboflow annotated dataset to train this model on pothole detection, aiming for high accuracy. The frontend is a React application, and includes a map that keeps track of detected potholes at the user's location. We believe that the performance results of our model and the possible incorporation into the frontend can offer great utility to drivers and local governments for road maintenance.

1. Introduction

For our project, we wanted to utilize our theoretical knowledge of computer vision algorithms and techniques to solve a real world problem. As such, we chose the application route for our project, so we could apply what we learned to build a solution with real world utility. We chose to tackle potholes as our problem to solve.

Potholes are a major issue on many roads and hinder the safety and efficiency of transportation. Annually, potholes are responsible for nearly $3 billion in vehicle damages in the United States alone, and affect millions of drivers. Aside from vehicle damages, potholes themselves remain a lasting issue due to how expensive they are to repair, both in time and money.

Early detection and tracking of potholes can help reduce the expenses generated by them. If drivers are aware of potholes in their immediate vicinity, they can avoid them, and if these potholes are tracked, local governments can target specific roads to repair to maximize efficiency. Therefore, we chose to design an application for pothole detection with two primary motivators: improving driver safety, and saving on overall costs.

Our project thus involves two parts: a trained model for real time pothole detection, and a frontend app to keep track of the detected potholes.

The model used was the Ultralytics YOLO v8 model [5], which we trained using a Roboflow annotated image set of potholes. This allowed us to detect potholes from images or live video feed. Considering that a major part of solving our problem was object detection, we decided to use a YOLO model. We chose the Ultralytics YOLO v8 due to it being the current top of the line object detection model, and because it was fairly straightforward to use after some experimentation.

As for the frontend, we built a React application and used the Leaflet.js to actually map where detected potholes were found. Our motivation behind building a functional frontend was that we wanted our project to be very user friendly. We were unfortunately unable to directly connect the model to the React app, but we are confident that having such an application can be of great use to drivers that want to avoid risky roads, or to governments that need to pinpoint problem areas to fix.

Overall, our two part project combines to build a unique solution to the pervasive issue of potholes. By not only training a model to detect potholes but also building an app to track them, our project can cut costs and promote safe and efficient travels.

2. Related Work

Recent works have approached pothole detection with vision-based techniques such as YOLO (You Only Look Once) and CNNs (Convolutional Neural Networks). Other recent works have approached pothole detection with machine learning techniques such as Naive Bayes, Support Vector Machine Classification (SVC), and Forensic-Based Investigation (FBI). In addition to vision-based and machine learning techniques, sensors and 3D reconstruction techniques have been used to detect potholes.

2.1. YOLO

Park et al. [4] compared the performances of YOLO v4, YOLO v4-tiny, and YOLO v5s used in pothole detection in

images. YOLO v4 ran 4000 epochs and took more than seven hours to train on a dataset of 665 pothole images, whereas YOLO v4-tiny ran 6000 epochs and took an hour and 7 minutes to train on the same dataset. Training YOLO v5 took 1 hour and 43 minutes to run 1200 epochs. YOLO v4-tiny had the highest mean average precision of 78.7% compared to YOLO v4 (77.7%) and YOLO v5s (74.8%). While the results are decent, they only applied to pothole images under clear weather conditions and sufficient lighting. Additionally, all 3 models had low confidence values for small potholes located at further distances in the image.

Model	Epochs	Time	MAP
YOLO V4	4000	7 hours	77.7%
YOLO V4-TINY	6000	1 hour 7 minutes	78.7%
YOLO V5	1200	1 hour 43 minutes	74.8%

Table 1. Comparing YOLO v4, YOLO v4-tiny, and YOLO v5.

Bucko et al. [2] used YOLO v3 and sparse R-CNN to detect potholes in images under adverse weather conditions. Despite the availability of newer YOLO versions, YOLO v3 was chosen for its favorable results under lower hardware requirements. YOLO v3 was up to three times smaller than sparse R-CNN, however, sparse R-CNN outperformed YOLO v3 in almost all adverse weather conditions, while YOLO v3 performed better in clear weather conditions. Bucko et al. suggested that adopting YOLO v4 or YOLO v5 could potentially enhance the results compared to using YOLO v3. This is supported by Park et al. as they stated that YOLO v4 and later versions are improved in speed and accuracy.

2.2. Machine Learning

Azhar et al. [1] approached pothole detection in images with machine learning techniques through several steps. They computed histograms of oriented gradients (HOG) features for input images, trained and classified features using Naive Bayes to label input as pothole or non-pothole, and located potholes in images through a normalized graph cut segmentation scheme. This approach performed better than other approaches used on the same dataset with 90% accuracy, 86.5% precision, and 94.1% recall. Additionally, it reduced the processing time of non-pothole images by 30% compared to the average processing time of pothole images of 0.673 seconds. Rather than creating bounding boxes around potholes in images like YOLO models, the normalized graph cut segmentation algorithm used by Alzar et al. locates potholes in images by highlighting their contour, which fits pothole shapes more closely than a bounding box.

Hoang et al. [3] also approached pothole detection in images with machine learning techniques. Instead of solely detecting potholes, they distinguished between patched and unpatched potholes. They used SVC to separate input data into two class labels of patched and unpatched potholes. They used the FBI algorithm to fine-tune the SVC hyperparameters, optimizing the SVC performance. This approach yielded very high results of a 94.833% classification average rate, 93.5% precision, and 96.4% recall.

2.3. Sensors and 3D Reconstruction Techniques

According to Bucko et al. [2] sensors such as the accelerometer, vibration sensor, magnetometer, and GPS used to detect potholes have high accuracy and are not dependent on visibility conditions. But they suffer from a low response time and require driving through a pothole to perform the detection. Laser scanning or stereo vision techniques can be employed to generate a 3D road profile. This can retrieve information about the size, depth, and position of potholes, but stereo vision is computationally complex and sensitive to motion and vibrations.

3. Method

YOLOv8, or "You Only Look Once version 8," is an object detection algorithm that belongs to the YOLO (You Only Look Once) family. It's known for its speed and accuracy, making it suitable for real-time object detection tasks. We chose YOLOv8 as we discovered that it would be the quickest model to use for our application, along with the fact that it offers a lot of resources to learn from. Further its ability to be easily integrated and being the fastest of all YOLO models made it an attractive choice. Finally, YOLO v8 offers a plethora of additional features such as segmentation, tracking, classification, etc. even though for our project we only needed object detection. YOLOv8 has several components that contribute to its overall architecture and performance:

3.1. Backbone Architecture - CSPDarknet53

- YOLOv8 employs CSPDarknet53 (CSP stands for Cross Stage Partial), which is a variant of the Darknet neural network architecture.
- CSPDarknet53 consists of cross-stage connections that facilitate information flow between early and late stages of the network. This promotes the efficient extraction of features from different scales, an especially important attribute, as we intended to use the model for live video feed and needed quick feature extraction.

3.2. Detection Head - YOLOv3-style

- The detection head is responsible for making predictions based on the features extracted by the backbone. YOLOv8 uses a detection head inspired by YOLOv3,

which involves multiple prediction layers at different scales.

- These prediction layers output bounding boxes, class probabilities, and objectness scores. Each prediction layer is associated with specific anchor boxes, allowing the model to handle objects of different sizes effectively.

3.3. Input Resolution

- YOLOv8 processes input images at a specified resolution. The default resolution is often set to 416x416 pixels. This resolution is a trade-off between accuracy and speed, allowing the model to maintain a good balance between precision and real-time performance.

3.4. Training Process

- YOLOv8 is trained using a dataset of annotated images. The training process involves optimizing the model's parameters using backpropagation and stochastic gradient descent (SGD). The model is trained to minimize the difference between its predictions and the ground truth annotations.

3.5. Data Augmentation

- To improve the model's generalization capabilities and reduce overfitting, YOLOv8 incorporates data augmentation during training. Techniques such as random rotations, flips, and scaling are applied to artificially increase the diversity of the training dataset

3.6. Object Detection

- YOLOv8 employs a one-stage object detection approach, meaning it directly predicts bounding boxes and class probabilities in a single forward pass through the network. This design contributes to the model's speed and efficiency

3.7. Loss Function

- The loss function used during training is crucial for guiding the optimization process. YOLOv8 typically employs a combination of localization loss, confidence loss (objectness), and class probability loss. The loss function penalizes deviations between the predicted values and the ground truth

3.8. Anchor Boxes

- YOLOv8 utilizes anchor boxes, which are predefined bounding box shapes that help the model handle objects of various sizes. Each prediction layer is associated with a set of anchor boxes to improve the model's ability to detect objects at different scales.

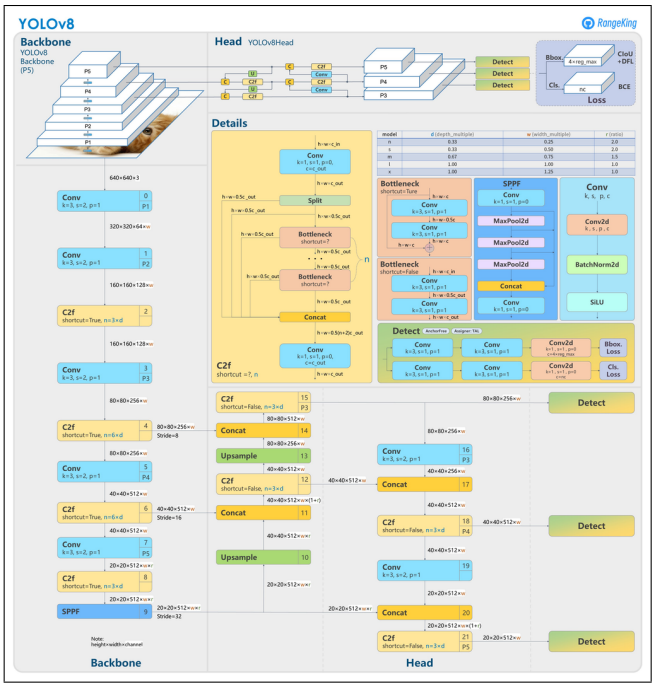

Figure 1. YOLOv8 Architecture

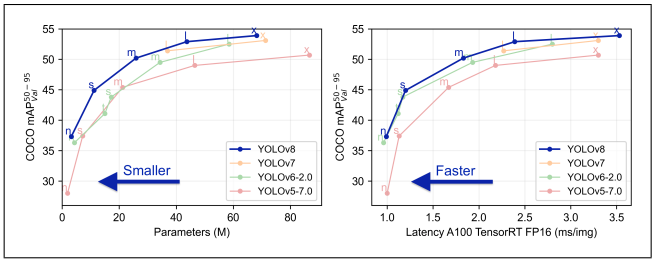

Figure 2. YOLOv8 versus previous versions

4. Experiments

4.1. Dataset

For our dataset we originally planned on using a smaller dataset since we wanted to use YOLO and that would require the group to get a bunch of pothole images and label them using some third party software. But were fortunate to come across a pothole dataset on Roboflow that had around 650 pre-labeled photographs of potholes that would be perfect for our YOLO model. The dataset was split into 3 different sections, one being the training set of 465 images, validation set of 133 images and the test set of 67 images.

4.2. Evaluation Metrics

As far as evaluation metrics, we focused mostly on improving accuracy. YOLO v8 comes with the ability to track multiple metrics, as shown below:

Figure 3. Example of dataset image

Metric	Value
Instances	330
Images	133
Box(P)	0.824
R	0.637
Accuracy	0.87
mAP50	0.773
mAP50-95	0.481

Table 2. Evaluation metrics for validation set of pothole detection using YOLOv8.

Epoch	box_loss	cls_loss	dfl_loss	Box(P)	R	mAP50
1/25	1.771	3.605	1.69	0.405	0.344	0.329
2/25	1.575	1.91	1.486	0.152	0.23	0.101
3/25	1.657	1.856	1.561	0.0269	0.0909	0.0115
4/25	1.682	1.799	1.603	0.289	0.194	0.16
5/25	1.668	1.693	1.568	0.513	0.324	0.368
6/25	1.607	1.608	1.514	0.316	0.33	0.266
7/25	1.585	1.542	1.527	0.514	0.436	0.438
8/25	1.543	1.469	1.491	0.526	0.44	0.452
9/25	1.491	1.391	1.463	0.55	0.461	0.439
10/25	1.486	1.468	1.452	0.642	0.522	0.576
11/25	1.446	1.415	1.476	0.698	0.539	0.593
12/25	1.4	1.3	1.416	0.619	0.531	0.557
13/25	1.388	1.302	1.389	0.703	0.581	0.65
14/25	1.366	1.335	1.394	0.735	0.59	0.648
15/25	1.34	1.18	1.368	0.683	0.588	0.64
16/25	1.3	1.224	1.355	0.625	0.556	0.621
17/25	1.317	1.284	1.359	0.711	0.603	0.671
18/25	1.253	1.124	1.339	0.734	0.624	0.703
19/25	1.22	1.041	1.301	0.71	0.667	0.711
20/25	1.213	1.01	1.288	0.77	0.612	0.702
21/25	1.179	1.001	1.258	0.738	0.64	0.736
22/25	1.144	0.98	1.24	0.705	0.664	0.727
23/25	1.108	0.9084	1.22	0.817	0.638	0.771
24/25	1.095	0.8743	1.203	0.819	0.603	0.747
25/25	1.06	0.8055	1.174	0.789	0.657	0.757

Table 3. Training Metrics for YOLOv8 Pothole Detection

4.3. Results and Analysis

Overall our model was very accurate at detecting potholes. The only major issue we faced was that the model would identify manholes and street cones near potholes as potholes.

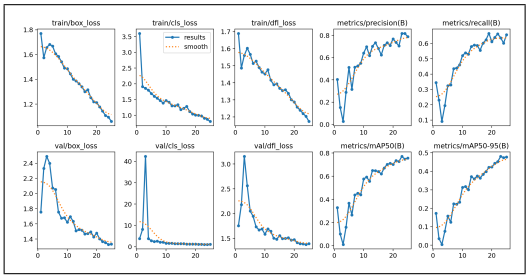

Figure 4. Model Metrics

Some of the things we think that would have improved our model accuracy was if we were to simply increase the dataset size so that it would be more fine tuned to detect potholes. Another solution I thought of was that we could train

it also detect street cones near potholes and to detect manhole coverings. However due to time constraints and lack of data we didn't get the chance to implement these things or improve our accuracy. Additionally, due to manholes being recorded in city records I think it would be possible to just keep track of the manholes and not having to worry about training our model to prevent false positives when detecting manhole coverings. We also came to the realization that extremely high accuracy and having a huge dataset isn't necessary for this application since majority of potholes aren't a life or death type of situation whereas having false positives or false negatives in a surgical or health related application would be.

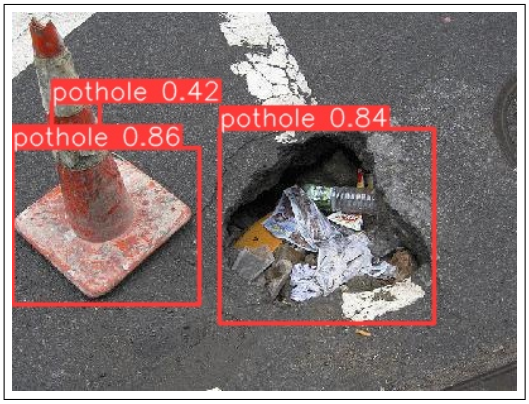

Figure 5. Street Cones False Positive

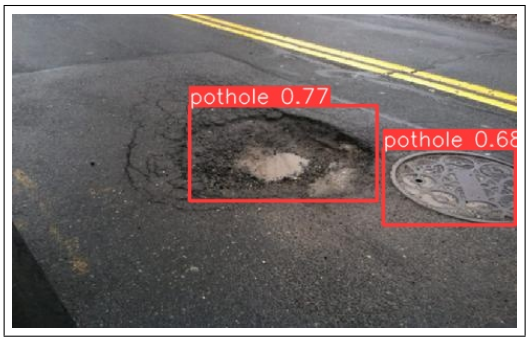

Figure 6. Manhole False Positive

5. Conclusion

In conclusion, the implementation of the Ultralytics YOLO v8 model for pothole detection, coupled with a frontend featuring a map for pothole location tracking, has proven to be a significant success. The model's high accuracy in detecting potholes suggests its potential utility for both individual drivers and local governments engaged in road maintenance. The integration of the detection results into a map provides a practical solution for users to monitor potholes in their vicinity.

We faced a few challenges during development, such as the false positives caused by manholes and traffic cones. Additionally, integrating our model in React proved to be challenging. However, we believe even these issues can be tackled with more time to research, and the product that we have built is still sound.

Overall, this project highlights the potential of advanced computer vision models, such as YOLO v8, in creating impactful solutions for real-world problems. The combination of accurate detection and user-friendly interface positions this project as a potential contribution towards improving road safety and maintenance practices.

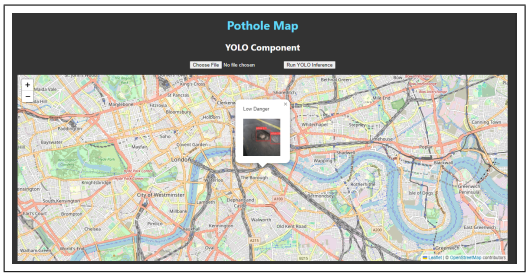

Figure 7. Screenshot of Frontend Map

References

[1] Kanza Azhar, Fiza Murtaza, Muhammad Haroon Yousaf, and Hafiz Adnan Habib. Computer vision based detection and localization of potholes in asphalt pavement images. *2016 IEEE Canadian Conference on Electrical and Computer Engineering (CCECE)*, 2016. 2

[2] Boris Bucko, Eva Lieskovska, Katarina Zabovska, and Michal Zabovsky. Computer vision based pothole detection under challenging conditions. *Sensors*, 22, 2022. 2

[3] Nhat-Duc Hoang, Thanh-Canh Huynh, and Van-Duc Tran. Computer vision-based patched and unpatched pothole classification using machine learning approach optimized by forensic-based investigation metaheuristic. *Complexity*, 2021:1–17, 2021. 2

[4] Sung-Sik Park, Van-Than Tran, and Dong-Eun Lee. Application of various yolo models for computer vision-based real-time pothole detection. *Applied Sciences*, 11, 2021. 1

[5] Jacob Solawetz. What is yolov8? the ultimate guide. https://blog.roboflow.com/whats-new-in-yolov8/. 1