Pothole Object Detection

Gonçalo Silva
DETI
Universidade de Aveiro
Aveiro, Portugal
goncalolsilva@ua.pt

Samuel Teixeira
DETI
Universidade de Aveiro
Aveiro, Portugal
steixeira@ua.pt

***Abstract*—Potholes are a significant issue for road safety and infrastructure, contributing to vehicle damage, increased repair costs, and posing risks to pedestrians and cyclists. Traditional pothole detection methods are often time-consuming and inaccurate, exacerbating the problem. This project aims to develop an efficient, real-time pothole detection system using the power of deep learning, particularly the You Only Look Once (YOLO) algorithm. Our approach demonstrates substantial improvements over traditional methods, with high precision (92.4%) and recall (92%), and an F1-score of 86.1%. The results demonstrate the potential of YOLO for scalable, accurate pothole detection, offering a viable solution for enhancing road safety.**

***Index Terms*—YOLO, Object Detection, Computer Vision, Machine Learning, Deep Neural Networks, Deep Learning, Neural Networks,**

I. Introduction

Potholes are a pervasive issue in urban and rural road networks, significantly impacting transportation infrastructure and road safety. These surface deformities, caused by factors such as weathering, traffic load, and inadequate maintenance, lead to vehicle damage, increased repair costs, and pose safety hazards to drivers and pedestrians alike. Hitting a pothole can cause a driver to lose control of their vehicle, leading to collisions with other cars, cyclists, or pedestrians. The impact of hitting a pothole is estimated to be equivalent to a collision at 35 mph (or 60 km/h). Injuries can range from whiplash and broken bones to spinal cord injuries and internal bleeding. [1]. Efficient detection and timely repair of potholes are critical for reducing these risks and ensuring the longevity of road infrastructure.

Potholes are a major issue for road infrastructure, and unfortunately, in countries like the United Kingdom (UK), they have been increasing drastically, with an estimated number of around 1 million, which is about 6 potholes per mile [2]. The number of incidents involving Potholes also seeing a rise, from around 21,725 in 2020, to 29,333 in 2024 [3] and they show no sign on slowing down as the country continues to not pass meaningful changes to mitigate the issue. The Independent Transport Commission found that pothole-related incidents cause about 1% of all road accidents. For motorcyclists and cyclists, the risks are even higher, with a survey by Cycling UK revealing that 31% of its members had been involved in accidents or near misses due to poor road surfaces, including potholes. [1]

But it's not just in the UK, the United States (US) also has a problem. The American Automobile Association (AAA) estimates that potholes cost U.S. drivers $3 billion annually in vehicle repairs. In New York City alone, potholes and road defects cost the city nearly $138 million in settlements for pedestrian injuries and vehicle damage over a six-year period. [1]

In summary, potholes are a major issue, not only accounting for a big cost in material repairs, but also for loss of life or life conditioning injuries. The solution is simple from a practical point of view, but with dwindling infrastructure, funds and low political attraction, the corrections get delayed and delayed, until the issue is overwhelming to tackle.

Traditional pothole detection methods, such as manual inspections or simplistic image-based approaches, are often time-consuming, resource-intensive, and prone to inaccuracies. Recent advancements in artificial intelligence and computer vision have opened new possibilities for automating and improving the accuracy of detection processes. By leveraging cutting-edge object detection algorithms, it is possible to identify potholes in real-time, offering a scalable and effective solution to this pressing problem.

This project aims to develop a robust system for the real-time detection and classification of potholes, with a focus on accuracy, speed, and practicality. Our system needs to work fast enough, in order to notify the driver/cyclist of the upcoming presence of a Pothole. By facilitating timely identification, this system has the potential to enhance road safety, and reduce the economic costs associated with road damage.

II. State of the Art/Related Work

Object detection is a very important field in Computer Vision, allowing the detection of all kinds of objects, animals and people (faces, cars, animals, health conditions, etc.). In the present, object detection is divided into two main categories, Traditional Machine Learning Methods and Deep Leaning. There exists roughly three types of Machine Learning Methods, which are Generalized Hough Transform, Harris Corner Detection and SIFT (Scale-Invariant Feature Transform), defined by geometric feature extraction, corner detection and local feature description. The three of them have some big disadvantages, being the first two sensitive to geometric features in the image or changes, like size, rotation and gray values, whilst the last one addresses rotation and scale problems, it still suffers from the remaining drawbacks [4].

Moving from traditional Machine Leaning methods, Deep learning came as Holy Grail to object detection, since it didn't require a highly effective feature engineering and was more efficient for training large datasets. With the rapid advancements in the field, specially in the last ten-fifteen years, vast techniques have been suggested and proposed for Object Detection with Deep Learning. With the most common being based in Region Based Convolutional Neural Networks (R-CNN). Quickly newer and faster method were invented, like Fast R-CNN, Faster R-CNN, YOLO and Mask R-CNN[4].

A. *Feature Engineering*

Traditional feature engineering for object detection are based on analyzing specific traits in the image, by reducing unnecessary details to focus on key features such as edges, corners, and textures. Techniques like the Generalized Hough Transform simplified complex shapes into mathematical models, enabling robust pattern recognition under partial occlusion. Harris Corner Detection identified points of interest where intensity varied significantly, making them reliable markers for object localization and motion tracking. The Scale-Invariant Feature Transform (SIFT) further advanced this approach by introducing descriptors invariant to scale and rotation, enhancing robustness against transformations and background clutter [5]. These methods relied heavily on predefined heuristics, making their use extremely situational, since each use case needs to be reviewed, in order to identify the correct features, making their implementation extremely costly. For instance, if aspects like image angle, coloration, among others change, the output can change dramatically.

B. *CNN*

With the introduction of R-CNN in 2014 [6], object detection transitioned from a fragmented, target-specific detection paradigm to a unified framework where Convolutional Neural Networks (CNN)s directly learned features for object localization and classification. R-CNN employs a selective search algorithm to propose regions of interest, which are then processed independently through CNNs for feature extraction and classification, significantly improving detection accuracy but at a high computational cost.

The evolution continued with Fast R-CNN, which addressed R-CNN's inefficiencies by integrating region-of-interest pooling into the CNN pipeline, allowing the entire image to be processed in a single forward pass. Faster R-CNN further optimized this by introducing (Region Proposal Networks (RPN)s) to generate proposals directly within the network, establishing a truly end-to-end architecture. This innovation not only reduced the computational load but also allowed real-time performance to become feasible in object detection tasks [7].

C. *YOLO*

YOLO was proposed in 2016 [8] as a unified, real-time object detection algorithm that simplified the traditional multi-stage pipeline of object detection into a single-shot process [9]. Unlike R-CNN, which breaks down detection into a sequence of steps involving region selection, feature extraction, and classification, YOLO directly predicts bounding boxes and class probabilities from image pixels using a single CNN. This regression-based approach contrasts with the proposal-classification paradigm of R-CNN, making YOLO much faster and more suitable for real-time applications.

YOLO divides the input image into a grid, where each grid cell predicts four values for the bounding box (center coordinates, width, and height), a confidence score indicating how likely it is that an object is present, and the probabilities for each class. This unified framework allows YOLO to incorporate global information from the entire image when making predictions, reducing errors caused by background clutter and making it faster than traditional detection systems that rely on region proposals [4] [9].

The loss function used in YOLO treats small and large bounding boxes equally, which can lead to inaccurate localization, especially for smaller objects. Additionally, when multiple objects appear in a single grid cell, the algorithm struggles with recall, often missing objects or misclassifying them because it assumes that all objects in the same cell belong to the same class (it's worth mentioning that more recent YOLO versions, such as v11n, are adapting solutions for problems like overlapping and small objects). Despite these drawbacks, YOLO remains a top choice for real-time detection, where speed is crucial, even though its accuracy still lags behind methods that use region proposal networks, such as Faster R-CNN [7].

A complete changelog of the evolution of YOLO versions can be seen in Table I.

Based on the fact that we are pursuing a real-time reliable technology, YOLO proves to be the best choice, even with it's limitations, making it the option that we'll be exploring.

III. Dataset

The dataset used belongs to a user from the platform Roboflow, which is composed of 11068 images of training, 1808 images of validation and 891 images of testing. The dataset was lightly reviewed and it was clear that there were a wide variety of labels in different circumstances, such as potholes in dashcam photos, highways, small roads, among others (essentially, following YOLO recommendations for datasets [20] and preventing data leakage [21]). Potholes themselves were in different shapes and textures (Ex: some of them had water inside, some were literally deep holes). Finally, one particular characteristic of this images is that some of them had a lot of blur, which is good for training since it adds an "unintentional" data augmentation to the dataset.

To test the impacts of a reduction of the dataset size, we decided to reduce the dataset by about 80 %, to test it's impacts on the model and also allow the testing of other variables,

Version	Release Year	Key Changes
YOLO v1	2016	Single-shot object detection using a fixed grid to predict bounding boxes and class probabilities directly from image pixels. Struggled with smaller objects. [8]
YOLO v2	2017	YOLO9000 allowed detection of 9000 object classes, added batch normalization, and improved accuracy and feature extraction. [10]
YOLO v3	2018	Used Darknet-53 backbone, added multiple layers for improved accuracy, and new anchor boxes for multi-scale detection. [11]
YOLO v4	2020	Introduced CSPDarknet backbone, cross-stage connections, and improved resolution anchors for better accuracy. More computationally expensive. [12]
YOLO v5	2020	Introduced CSPNet backbone, lighter computation, and improved training methods. Focused on simplicity and smaller model size. [13]
YOLO v6	2021	Introduced SPNet attention mechanism, advanced data augmentation, and optimized for edge devices. Focused on balancing speed and accuracy. [14]
YOLO v7	2022	Integrated CNN and transformer hybrid for multi-scale detection, focused on faster inference with minimal computation. [15]
YOLO v8	2023	Improved feature extraction with depth-wise separable convolutions, added video detection and temporal consistency. [16]
YOLO v9	2024	Refined computational efficiency, robustness against occlusions, and improved recall and localization. [17]
YOLO v10	2024	Introduced Vision Transformer (ViT) attention layers for improved context and feature extraction. Optimized for training. [18]
YOLO v11	2024	Improved high-resolution image support, large-scale detection, and integrated transformer-based anchors. [19]

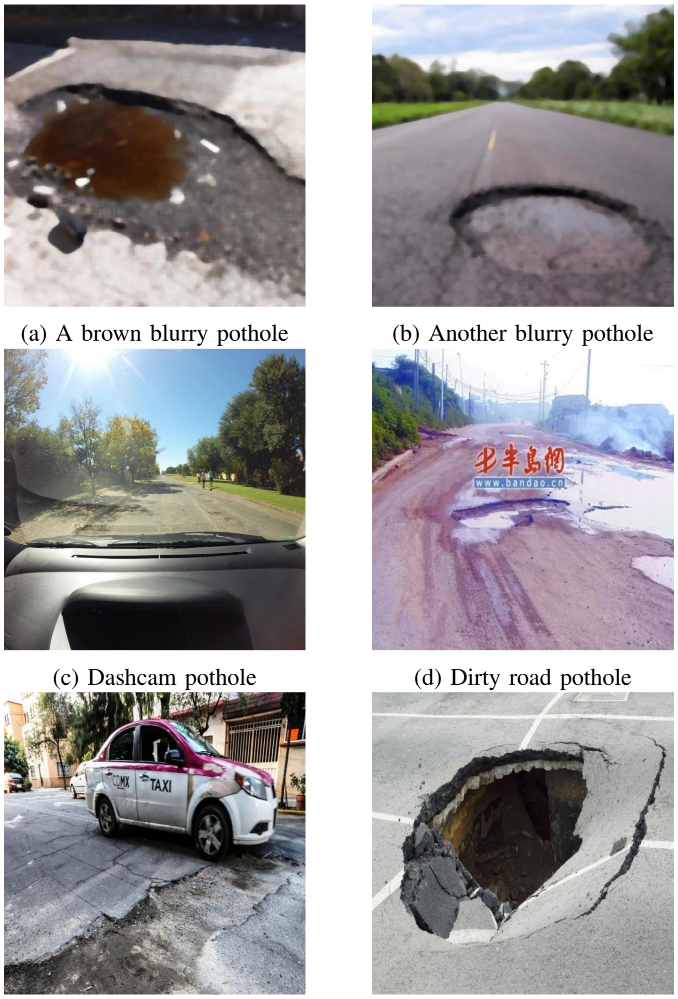
TABLE I: Summary of YOLO Versions and Key Changes

due to reduction in computing power needed. The reduced dataset contains 2400 images for training, and 300 images for validating and testing (each)

The training for both types of dataset includes the following data augmentation techniques [22]:

1) **hsv_h: 0.015**
   - **Impact**: Adjusts the hue of the image by a factor of 0.015. Can help the model generalize better by making it invariant to slight color changes.
2) **hsv_s: 0.7**
   - **Impact**: Adjusts the saturation of the image by a factor of 0.7, helping the model to handle variations in color intensity.
3) **hsv_v: 0.4**
   - **Impact**: Adjusts the value (brightness) of the image by a factor of 0.4, which can help the model generalize to different lighting conditions.
4) **degrees: 0.1**
   - **Impact**: Translates the image by up to 10% of its dimensions, potentially making the model more invariant to small translations of objects within the image.
5) **scale: 0.5**
   - **Impact**: Scales the image by a factor of 0.5, helping it generalize to objects of different sizes.
6) **fliplr: 0.5**
   - **Impact**: Horizontally flips the image with a probability of 0.5, helping the model learn to recognize objects regardless of their horizontal orientation.
7) **mosaic: 1.0**
   - **Impact**: Uses mosaic augmentation, which combines four training images into one. This can help the model learn to detect objects in a more complex and varied context.
8) **erasing: 0.4**
   - **Impact**: Randomly erases parts of the image with a probability of 0.4, which can help the model become more robust to occlusions.

This augmentations help the model to work in different circumstances, not necessarily represented in the training data.



(a) A brown blurry pothole (b) Another blurry pothole (c) Dashcam pothole (d) Dirty road pothole (e) Pothole with unique shape (f) A very deep pothole

Fig. 1: Examples of various potholes in the dataset

Data augmentation is especially useful for small datasets, but can also be helpful in larger ones. Contrary to augmentation in other deep learning models, which can add too much noise and hinder the model's performance, the ones provided here for YOLO are relatively simple, straightforward and are based on the ones suggested by Ultralytics [23] [24].

IV. Training

In this section, we'll be describing our training process, the models and hyper-parameters used. We also describe the results of this process based on the models performance metrics, which are generated based on data from the three stages, training, validation and testing.

The following metrics can be obtained during the training stage:

- **box_loss**: The bounding box regression loss, measures how accurately predicted bounding boxes align with ground truth boxes during training.
- **cls_loss**: The classification loss, evaluates the model's ability to correctly classify objects during training.
- **dfl_loss**: The Distribution Focal Loss, is used for improving regression tasks by focusing on key features during training.

The next metrics can be obtained during the validation/testing stage:

- **box_loss**, cls_loss and **dfl_loss** can also be computed in this stage
- **Intersection over Union (IoU)**: Measures the overlap between predicted bounding boxes and ground truth boxes to evaluate localization quality.
- **Precision(B)**: Precision is calculated across validation/testing batches, indicates the proportion of true positive predictions among all positive predictions.
- **Recall(B)**: Recall is calculated across validation/testing batches, measuring the proportion of true positive predictions among all actual positives.
- **Average Precision (AP)**: The precision averaged over different recall levels
- **Mean Average Precision (mAP)**: The mean of average precision scores across all classes
- **mAP50(B)**: Mean Average Precision calculated at a fixed IoU threshold of 0.50 across validation/testing batches.
- **mAP50-95(B)**: Mean Average Precision calculated across multiple IoU thresholds (from 0.50 to 0.95, in increments of 0.05) across validation/testing batches, providing a comprehensive performance measure.
- **F1 Score**: The harmonic mean of precision and recall, offering a balanced evaluation of detection performance.
- **Confusion Matrix**: Presents the classification performance, comparing the models predictions (True Positives, True Negatives, False Positives or False Negatives).

A. Batch Size

Batch Size is the number of training examples processed in one iteration, during model training. It plays a crucial role in determining training efficiency, speed, and performance. A smaller batch size requires less memory, making it suitable for hardware with limited resources, while a larger batch size utilizes computational power more effectively but demands more memory. Smaller batch sizes also help reduce overfitting by introducing noise during training, allowing the model to generalize better, but can also lead to higher variance in gradient estimates. For example, by processing less images before updating weights, the model might focus more on certain features to classify objects simply because they are more present in that small batch, while other features that are not as much represented there might be a lot more present on the remaining dataset, leading to incorrect weight measurements and loss values going up and down). In contrast, larger batch sizes can lead to smoother convergence but may require careful adjustment of the learning rate [25].

Take in consideration a dataset of 11,068 images and a batch size of 4, it's number of Iterations per Epoch are calculated as follows:

$$\text{Iterations per Epoch} = \left\lceil \frac{\text{Total Images}}{\text{Batch Size}} \right\rceil = \left\lceil \frac{11068}{4} \right\rceil = 2767$$

In our case, since we are dealing with devices that have limited computing power, we had to make compromises. During testing, it became apparent that only the Kaggle platform (which has usage limits), using their Nvidia Tesla P100 could handle a batch size of 16 (default), while our other options supported stably, a size of 4. With this in mind, we tested a model for a small number of Epochs, with both batch sizes, using YOLOv11. It's Precision numbers, available in Figure 2, show that the model with batch size of 4, suffered less variance than the one with 16, since it generalizes a bit better due to analyzing less examples in each iteration, however, the results will be pretty much the same on the model's performance, as can be seen in Figure 3, where mAP50-95, mAP50 and mAP75 show similar numbers (withing margin of error), for both models. Thus proving that in our case, the impacts of the Batch size don't affect the models performance, only the training time.

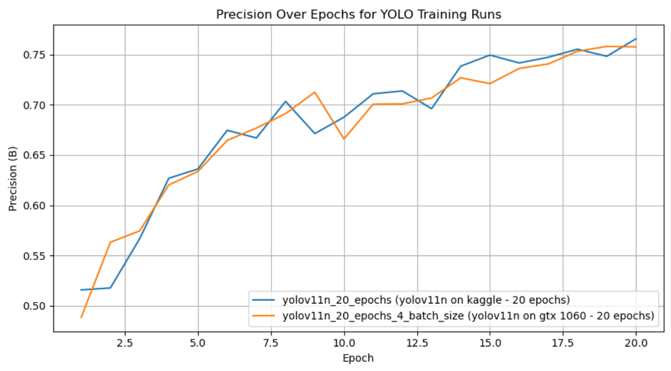


Fig. 2: Batch Size impacts in model Precision

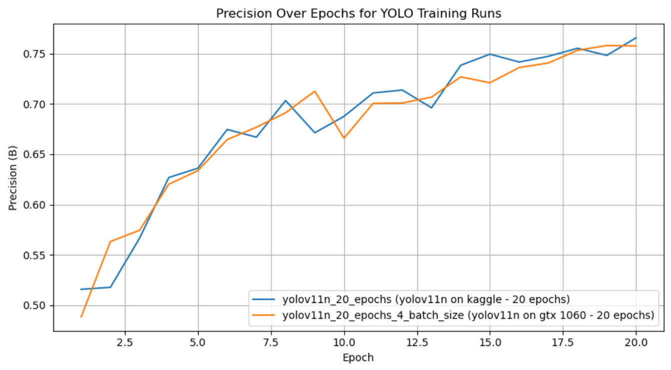


Fig. 3: Batch Size impacts in AP metrics

B. Epochs

Epochs are a fundamental concept in machine learning, referring to a single complete pass through the entire training dataset. During an epoch, every data point in the dataset is processed by the model, which adjusts it's weights to minimize errors and improve predictions. Multiple epochs are typically required to enable the model to learn meaningful patterns without merely memorizing the data [26].

The number of epochs directly affects the depth of the learning process. Too few epochs may lead to **underfitting**, where the model fails to capture the underlying patterns in the data. On the other hand, too many epochs risk **overfitting**, where the model learns noise and specific details rather than general trends. To strike the right balance, techniques such as **early stopping** can be employed to halt training once the model's performance ceases to improve on validation data [26].

Considering our full dataset of 11,068 images and the batch size of 4, as explained in subsection IV-A. Each epoch processes the entire dataset, requiring multiple iterations based on the batch size. The total number of iterations per epoch is given by:

$$\text{Iterations per Epoch} = \left\lceil \frac{\text{Total Images}}{\text{Batch Size}} \right\rceil = \left\lceil \frac{11068}{4} \right\rceil = 2767$$

For example, if the training is set to run for 10 epochs, the model will complete:

$$\text{Total Iterations} = \text{Epochs} \times \text{Iterations per Epoch}$$

$$= 10 \times 2767 = 27670$$

With each epoch, the model becomes progressively better at identifying patterns, provided that the training is monitored to avoid overfitting.

Since the more epochs a model have, the more time it takes, we also present the training time of each model, based on number of epochs, dataset, batch size and hardware used. This results can be seen in Figure 13 (Note that Kaggle platform uses Nvidia Tesla P100).

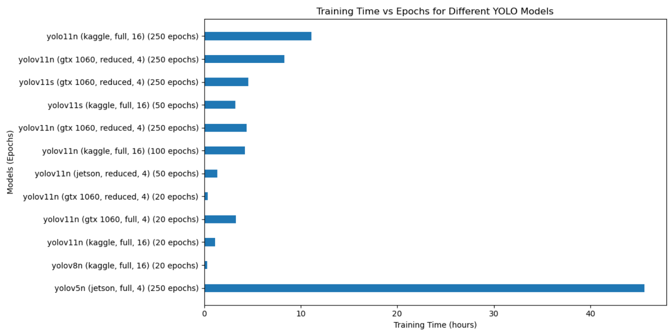


Fig. 4: Training time, based on number of epochs, model, batch size and hardware

C. Optimizer

Optimizers in YOLO are algorithms that adjust the model's weights during training in order to minimize the cost function.

They are very important since they influence how efficiently the model learns from data.

Among optimizers, we have some, like the three below: Stochastic Gradient Descent (SGD), RMSprop and Adaptive Moment Estimation (Adam)

SGD updates model parameters by taking steps proportional to the negative gradient of the loss function, controlled by a fixed learning rate. While efficient and widely used, SGD struggles with issues like oscillations near saddle points and the need for careful tuning of the learning rate, making it less effective in complex scenarios. [4]

RMSprop introduces an adaptive learning rate mechanism. It scales the learning rate for each parameter based on the recent history of squared gradients, allowing it to navigate loss surfaces with varying curvature effectively. This makes RMSprop particularly suited for non-stationary objectives and sparse gradients. However, it can sometimes stagnate in flat regions or over-adjust parameter updates. [4]

Adam combines the strengths of SGD and RMSprop, integrating momentum (from SGD) to smooth updates and adaptive learning rates (from RMSprop) for stability. Adam dynamically adjusts its step sizes for each parameter while correcting biases from initialization, enabling fast convergence with minimal hyper-parameter tuning. [27]

Because of this, Adam was the chosen optimizer. In most cases, it reaches the optimal solution due to its adaptability and faster convergence.[4] [28]

For watching the improvements of Optimizers, two models were trained based on YOLOv11n with Adam Optimizer and with Auto Optimizer. Auto chooses the best suited optimizer based on the dataset.

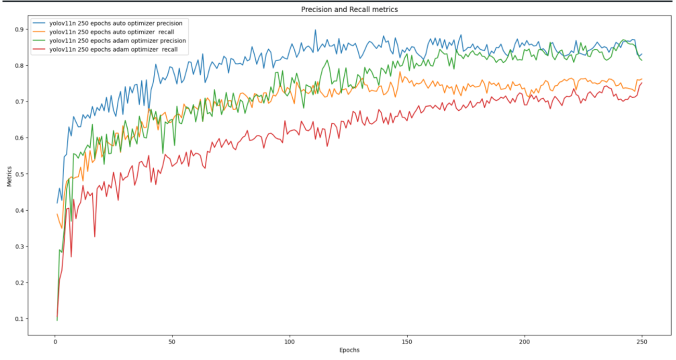


Fig. 5: Precision/Recall: Auto vs Adam Optimizer

The following results show that the Auto Optimizer overcame the Adam Optimizer. Reasons behind this, include the fact that, perhaps, our dataset wasn't well suited for the Adam Optimizer. Adam's optimizer suits better for bigger and more complex datasets. Since Adam updates learning rates and weights more aggressively, it is possible that the model might've felt into a suboptimal range of metrics, compared to SGD, which tends to explore the parameter space more broadly and be more forgiving to suboptimal hyper-parameters.[28]

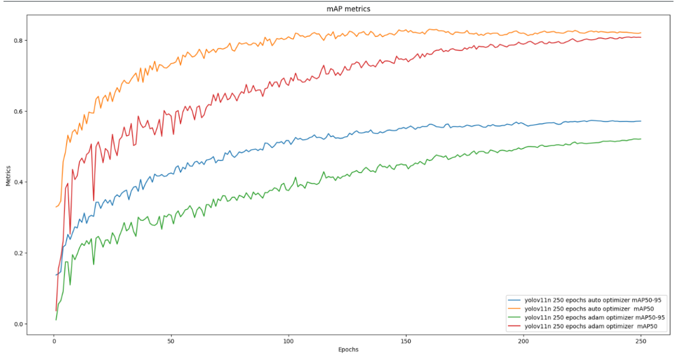


Fig. 6: mAP50/mAP50-95: Auto vs Adam Optimizer

D. *YOLO versions comparison*

Comparing versions is important in order to show the evolution of algorithms

For that, 2 models were compared: YOLOv5nu, an older yet popular YOLO version, and YOLOv11n.

All statistics shown in the graphics above clearly show that YOLOv11n surpassed YOLOv5nu, in precision/recall, with a PR curve of 0.861, compared to 0.838 of YOLOv5nu, better mAP scores and less validation losses.

The confusion matrix corroborates this values. YOLOv11 had 3733 true positives, 754 false negatives and 578 false positives. YOLOv5nu 3580 true positives, 907 false negatives and 597 false positives.

YOLOv11n is a more accurate model, specially with the high aspect ratio that our dataset has (and is more suitable for those needs). But YOLOv5nu also performed well in comparison, being a better choice for less computational power devices, such as IoT devices, embedded and such.

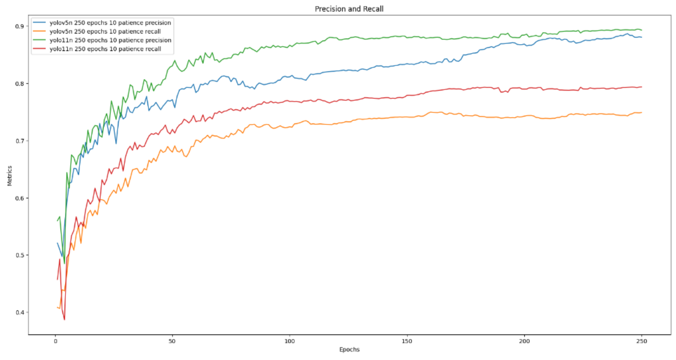


Fig. 7: Precision/Recall Across different models

E. *Reduced vs Complete dataset*

During our training, we decided to reduce the original dataset's size due to lack of computational power. Since we have models with a complete dataset and a reduced dataset, we decided to see the impact of more or less data in the training.

We have 2 ideal candidates for this measurement: YOLOV11n model with 250 epochs and 10 patience with com-

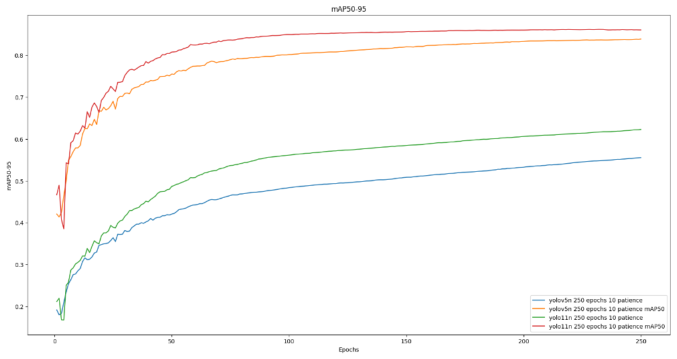


Fig. 8: mAP50/mAP50-95 Across different models

plete dataset and another model with the same circumstances, but with reduced dataset.

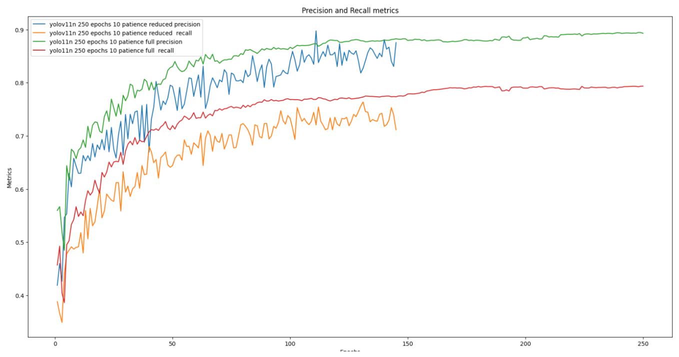


Fig. 9: Precision and Recall: Reduced vs Complete Dataset

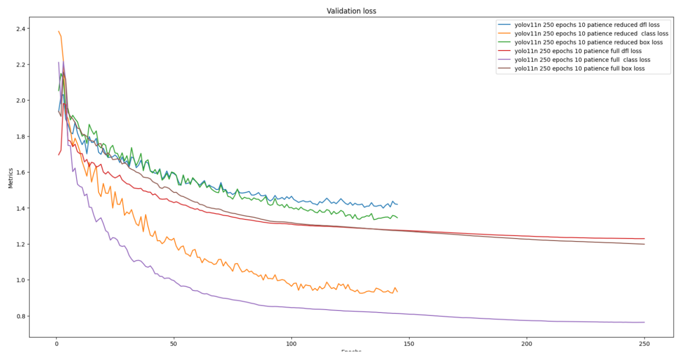


Fig. 10: Validation Loss: Reduced vs Complete Dataset

The reduced dataset stopped at 146 epochs. This means that it didn't find relevant improvements on validation metrics for 10 consecutive epochs. On the other side, the other model with the same level of patience kept improving and reached 250 epochs. It is also noticeable that the model with a complete dataset didn't have much discrepancy, the learning process was more linear. On the other side, the model with reduced dataset had a lot of "jiggling" values going up and down.

This can be explained due to the fact that, the more data that exists (assuming that the data has a wide range of different scenarios of potholes), the easier it is for a model to generalize and learn [20]. For the complete model, there was much more data, so the model kept learning at a steady rate, reaching a near convergence/plateau state (it still kept improving at a very slow pace).

F. Patience

Patience is a simple measure of how many iterations will the module make without any improvements in validation metrics before it stops. For example, for a patience of 3, if a module doesn't get better (lower) validation results after 3 iterations, the training stops. This can be beneficial because it might prevent the model from doing unnecessary training that might overfit the model [28].

With a reduced dataset, 2 YOLOv11n models were trained with 250 epochs each, only differing in the patience attribute (one with 10 and another with 100).

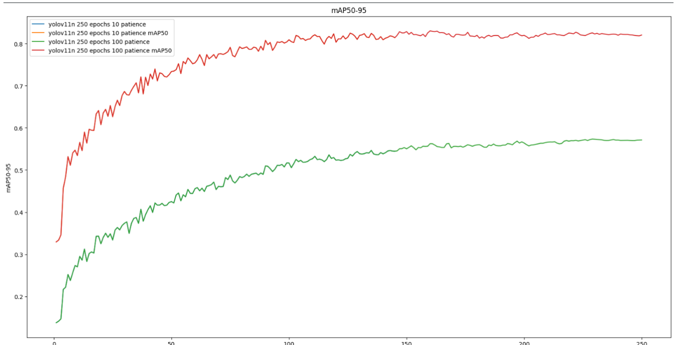


Fig. 11: mAP scores for the patience models

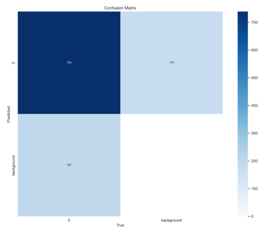
Since the models had the same conditions, there was an overlap in mAP scores. The model with patience 10 stopped at 146 epochs.

The first model (default patience of 100) has some solid statistics, but few issues detected.

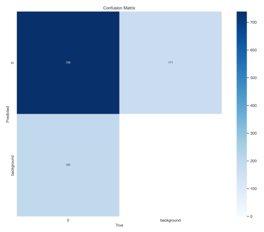
The validation loss were about 0.3 to 0.4 higher than training loss (class, box, dfl). Also, Precision and recall reached plateau state, while having a lot of "jiggle". Values going up and down. Those values can be explained due to the reduced validation dataset (300 images) with a lot of noises, which causes the validation to be more prone to variation [20]. Also, the plateau state that extended for over 100 epochs on every validation could've ended up in overflow, but it probably didn't, since the confusion matrices results are similar between patience 10 and 100.

Validation losses on the second model were similar to training and to the previous model, but precision and recall still oscillated a lot, which emphasizes the idea of a bigger dataset.

The PR curves and the confusion matrices show that the models had similar outputs and, while the 100 patience model had extended plateaus of over 100 epochs, those metrics show that the none of the models overfit. The reason behind this is probably because YOLOv11 already has a lot of ways inside its algorithm to prevent overfitting even when training a lot more epochs than necessary [28]



(a) Confusion Matrix for 10 patience



(b) Confusion Matrix for 100 patience

Overall, both models were very similar, but the 10 patience one needed much less computational power to give a near equal result, which emphasizes the importance of patience argument.

V. Results

Based on the results chosen in the previous section, we decided to go with YOLOv11n as our final model to detect potholes. This model was trained with 250 epochs, patience 10 and batch size of 16. In this section we'll be evaluating it's performance in real-world scenarios.

A. Prediction/Inference time

Our use-case and one of YOLO selling points, is to be able to detect objects, in this case Potholes in real-time. To test if our trained model was feasible in such case, we proceeded to test it, using an Nvidia Jetson, with sixteen random images of the *test* dataset. Here, we managed to obtain a total image processing time of 36.50 milliseconds (ms), with 1.32 ms of variance. Of those, 29.33 ms are taken by the Inference of the model (with 1.22 ms of variance), as can be seen in Figure 13. Using the following equation:

$$\text{Frames per Second (fps)} = \lfloor\frac{1000}{\text{Total Time}}\rceil = \lfloor\frac{1000}{36.50}\rceil = 27\text{fps}$$

we can prove that the model is able to process 27 fps, thus proving that the model can perform real time pothole detection.

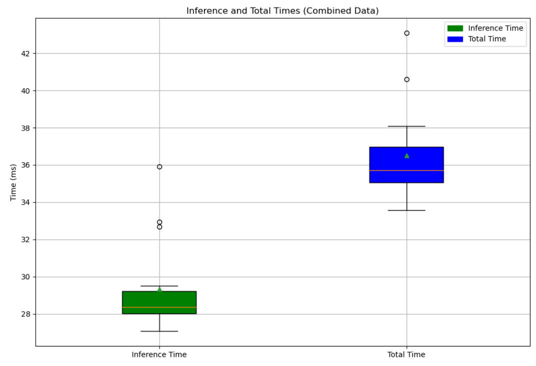


Fig. 13: Average and confidence interval for the Inference and Total time spent processing each frame

B. Check Predictions

In terms of showing the capabilities of the model, based on it's testing set, we used the sixteen random images defined above and joined the YOLO predictions, with the labels in the data, where we reach Figure 14. In here we can observe that YOLO had some missed predictions, however, when it predicted a Pothole, the bounding box was accurate, even through small objects and difficult environments, with low lighting conditions and reflections.

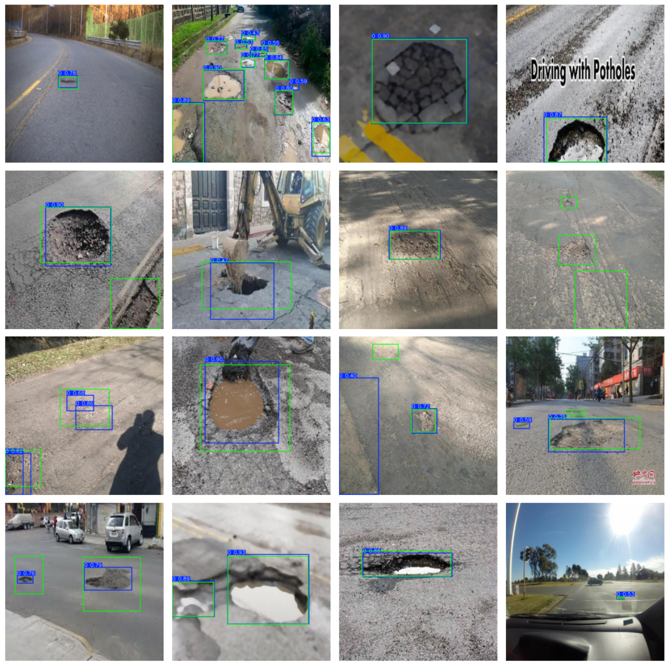


Fig. 14: Labels vs. Prediction of the model

C. Test with a video

In order to test the model in a more real-world scenario, we decided to use it directly in a video taken from Youtube, with a small excerpt of the results available in Google Drive. In here, we can see that the model performed well in difficult lighting conditions, with a lot of reflections on the ground, being at the dashcam level and the car traversing at 100 km/h.

D. Comparisons with State Of Art

Our work demonstrates significant advancements in pothole detection by leveraging YOLO v11n on a diverse dataset encompassing varied conditions such as dashcam frames, still photos, low resolution, and wet surfaces. The results obtained, summarized in Table II, highlight the improvements achieved in precision, recall, and mAP, compared to state-of-the-art methods.

VI. Conclusion

Our approach with YOLO v11n significantly outperforms previous methods. By training on a diverse and challenging dataset, we achieved an F1-confidence of 84 at 38.1%, precision-confidence of 1 at 92.4%, recall-confidence of 92%,

Method	Type	Approach	Precision	Recall	F1-Score	mAP@0.5
Koch and Brilakis [29] [30]	Traditional Computer Vision	Histogram-based thresholding, elliptical shape analysis	82%	86%	N/A	N/A
Tedeschi and Benedetto [30]	Machine Learning	LBP features with multiple classifiers	70%	70%	70%	N/A
Buza et al. [29]	Traditional Computer Vision	Spectral clustering for shape extraction	N/A	N/A	N/A	81%
Lokeshwor et al. [29]	Machine Learning	Frame classification using texture, shape, and dimensions	95%	81%	N/A	N/A
Staniek [30]	Machine Learning	Stereo vision with Hopfield neural network	N/A	N/A	N/A	N/A
Mir Tahmid [31]	Deep Learning	Compact Convolutional Transformers (CCT)	90%	N/A	High	N/A
Stpete_ishii [32]	Deep Learning	YOLO-NAS on pothole-size-focused dataset	0.95%	95.77%	1.89%	37.56%
Our Work (YOLO v11n)	Deep Learning	YOLO v11n on a diverse dataset	**92.4%**	**92%**	**86.1%**	**84%**

TABLE II: Comparison of Methods for Pothole Detection

and mAP@0.5 of 84%. These results reflect the robustness and scalability of our model for real-world applications, addressing the limitations of previous works and setting a new benchmark. We also were able to observer that YOLO is a relatively forgiving model, with generally difficult problems like overfitting, being easily mitigated, using it's feature set, as long as our dataset is good and varied. Better results could even possibly be obtained by adjusting and testing more hyper-parameters, however, such couldn't be observed due to our limits in computational capability.

VII. Future Work

The work conducted in this report could be improved in the following ways:

- Use segmentation an try to estimate the size of the pothole, as to only notify of potentially dangerous ones.
- Use more classes, classifying the potholes based on size
- Increase the number of epochs of the final model to 600 or 1200 epochs [28], since it didn't overfit, some gains may have been left out
- Experiment with other hyper-parameters, such as learning rate, momentum, warm up other optimizers (especially SGD)

VIII. Contributions

The distributions for the work here conducted are as follows:

- Gonçalo Silva, 103244, 50%
- Samuel Teixeira 103325, 50%

IX. Acknowledgments

The research and writing of this paper were conducted with the use of Artificial Intelligence (AI) tools such as ChatGPT, Copilot, Gemini or Claude.

References

[1] Vialytics, *The dangers of potholes: A growing threat to public safety*, Aug. 2024. [Online]. Available: https://www.vialytics.com/blog/dangersofpotholes.

[2] L. Woodhead, *Potholes: What are they and why are they dangerous?* Jan. 2024. [Online]. Available: https://www.bbc.com/news/uk-england-67958426.

[3] *Rac pothole index – statistics and data for uk roads.* [Online]. Available: https://www.rac.co.uk/drive/advice/driving-advice/rac-pothole-index-statistics-data-and-projections/.

[4] X. Zou, "A review of object detection techniques," in *2019 International Conference on Smart Grid and Electrical Automation (ICSGEA)*, 2019, pp. 251–254. DOI: 10.1109/ICSGEA.2019.00065.

[5] D. G. Lowe, *Distinctive image features from scale-invariant keypoints - international journal of computer vision*. [Online]. Available: https://link.springer.com/article/10.1023/B:VISI.0000029664.99615.94#citeas.

[6] R. Girshick, J. Donahue, T. Darrell, and J. Malik, "Rich feature hierarchies for accurate object detection and semantic segmentation," in *2014 IEEE Conference on Computer Vision and Pattern Recognition*, 2014, pp. 580–587. DOI: 10.1109/CVPR.2014.81.

[7] R. Girshick, J. Donahue, T. Darrell, and J. Malik, *Rich feature hierarchies for accurate object detection and semantic segmentation*, Oct. 2014. [Online]. Available: https://arxiv.org/abs/1311.2524.

[8] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, *You only look once: Unified, real-time object detection*, May 2016. [Online]. Available: https://arxiv.org/abs/1506.02640.

[9] R. Kaur and S. Singh, "A comprehensive review of object detection with deep learning," *Digital Signal Processing*, vol. 132, p. 103 812, Nov. 2022. DOI: 10.1016/j.dsp.2022.103812.

[10] J. Redmon and A. Farhadi, *Yolo9000: Better, faster, stronger*, Dec. 2016. [Online]. Available: https://arxiv.org/abs/1612.08242.

[11] J. Redmon and A. Farhadi, *Yolov3: An incremental improvement*, Apr. 2018. [Online]. Available: https://arxiv.org/abs/1804.02767.

[12] A. Bochkovskiy, C.-Y. Wang, and H.-Y. M. Liao, *Yolov4: Optimal speed and accuracy of object detection*, Apr. 2020. [Online]. Available: https://arxiv.org/abs/2004.10934.

[13] Ultralytics, *Ultralytics/yolov5: Yolov5 in pytorch ¿ onnx ¿ coreml ¿ tflite*. [Online]. Available: https://github.com/ultralytics/yolov5/.

[14] Z. Ge, S. Liu, F. Wang, Z. Li, and J. Sun, *Yolox: Exceeding yolo series in 2021*, Aug. 2021. [Online]. Available: https://arxiv.org/abs/2107.08430.

[15] C.-Y. Wang, A. Bochkovskiy, and H.-Y. M. Liao, *Yolov7: Trainable bag-of-freebies sets new state-of-the-art for real-time object detectors*, Jul. 2022. [Online]. Available: https://arxiv.org/abs/2207.02696.

[16] [Online]. Available: https://yolov8.com/.

[17] C.-Y. Wang, I.-H. Yeh, and H.-Y. M. Liao, *Yolov9: Learning what you want to learn using programmable gradient information*, Feb. 2024. [Online]. Available: https://arxiv.org/abs/2402.13616.

[18] A. Wang *et al.*, *Yolov10: Real-time end-to-end object detection*, Oct. 2024. [Online]. Available: https://arxiv.org/abs/2405.14458.

[19] Ultralytics, *Ultralytics/ultralytics: Ultralytics yolo11*. [Online]. Available: https://github.com/ultralytics/ultralytics.

[20] Ultralytics, *Tips for best training results*, Sep. 2024. [Online]. Available: https://docs.ultralytics.com/yolov5/tutorials/tips_for_best_training_results/.

[21] Ultralytics, *A guide on model testing*, Oct. 2024. [Online]. Available: https://docs.ultralytics.com/guides/model-testing/#signs-of-underfitting.

[22] Ruman, *Yolo data augmentation explained*, Jun. 2023. [Online]. Available: https://rumn.medium.com/yolo-data-augmentation-explained-turbocharge-your-object-detection-model-94c33278303a.

[23] Ultralytics, *Augment*. [Online]. Available: https://docs.ultralytics.com/reference/data/augment/.

[24] Ultralytics, *Train*, Nov. 2024. [Online]. Available: https://docs.ultralytics.com/modes/train/#train-settings.

[25] Ultralytics, *Batch size*, Oct. 2024. [Online]. Available: https://www.ultralytics.com/glossary/batch-size.

[26] Ultralytics, *Epoch*, Oct. 2024. [Online]. Available: https://www.ultralytics.com/glossary/epoch.

[27] D. Giordano, *7 tips to choose the best optimizer*, Jul. 2020. [Online]. Available: https://towardsdatascience.com/7-tips-to-choose-the-best-optimizer-47bb9c1219e.

[28] Ultralytics, *Tips for model training*, Oct. 2024. [Online]. Available: https://docs.ultralytics.com/guides/model-training-tips#other-techniques-to-consider-when-handling-a-large-dataset.

[29] Y.-M. Kim, Y.-G. Kim, S.-Y. Son, S.-Y. Lim, B.-Y. Choi, and D.-H. Choi, "Review of recent automated pothole-detection methods," *Applied Sciences*, vol. 12, no. 11, 2022, ISSN: 2076-3417. DOI: 10.3390/app12115320. [Online]. Available: https://www.mdpi.com/2076-3417/12/11/5320.

[30] A. Dhiman and R. Klette, "Pothole detection using computer vision and learning," *IEEE Transactions on Intelligent Transportation Systems*, vol. 21, no. 8, pp. 3536–3550, 2020. DOI: 10.1109/TITS.2019.2931297.

[31] M. Tahmid, *Potholes cnn transfer learning cct 90*, Sep. 2024. [Online]. Available: https://www.kaggle.com/code/tahmidmir/potholes-cnn-transfer-learning-cct-90.

[32] Stpeteishii, *Potholes yolo-nas train and predict*, May 2023. [Online]. Available: https://www.kaggle.com/code/stpeteishii/potholes-yolo-nas-train-predict/notebook.

X. Acronyms

AI	Artificial Intelligence
UK	United Kingdom
US	United States
R-CNN	Region Based Convolutional Neural Networks
CNN	Convolutional Neural Networks
RPN	Region Proposal Networks
YOLO	You Only Look Once
IoU	Intersection over Union
AP	Average Precision
ms	milliseconds
fps	Frames per Second
SGD	Stochastic Gradient Descent
Adam	Adaptive Moment Estimation