A Cloud-Based Road Infrastructure Analysis System using Machine Learning

Thegaran Naidoo[*], *Jaco* Verster[1], and *Stephen* Marais[1]

[1]CSIR, Centre for Robotics and Future Production, 1 Meiring Naude Road, Pretoria, South Africa

Abstract. A cloud-based road infrastructure analysis system was developed to assist as an information management system for detecting and managing information about road defects and road assets. The system uses computer vision and machine learning algorithms and is accessed through a web interface. The detection results are viewed through an online, account-based web interface.

1 Introduction

An efficient road network is vital to any economy [1]. To ensure the high availability of the road network, it is important to regularly inspect and maintain the road infrastructure. The Centre for Robotics and Future Production at the Council for Scientific and Industrial Research (CSIR) has developed a system which aims to assist road engineers and municipalities with the maintenance of road infrastructure.

The system which is called the Road Infrastructure Analysis System (RIA) system is a cloud-based information management system for keeping track of road defects and road assets. Examples of road defects are potholes, cracks, and edge breaks, and examples of road assets are road signs, road markings, and traffic lights.

The system has a scalable cloud-based design [2] that would allow it to manage data from multiple sources and to allow for the detection and classification of multiple road defects and road assets. In the current iteration the data preparation and data analysis plugins were developed to detect potholes in video data.

The system automatically analyses the videos that are uploaded by the user and produces geo-located detection results. The results are stored in a database and can be accessed through a web-based interface. The system also supports human-in-the-loop validation of the automated detections.
Section 2 describes the design and methodology, covering the system overview, the sensor system, the processing pipeline, and dataset creation. Section 3 presents the results that were obtained, and conclusions are presented in Section 4.

* Corresponding author: tnaido@csir.co.za

2 Design and Methodology

2.1 System Overview

The system consists of mobile devices that are used to collect video or image data of the environment and a cloud-based analysis and reporting system.

The typical flow of data during the operation of the RIA system is illustrated in Figure 1. Data in the form of video or still images are captured via cameras or mobile applications and uploaded into the RIA system using a web browser or the RIA API. The RIA system then processes the data, and the analysis results are stored in a database. The results can be viewed on a web-based dashboard which uses a map interface to indicate the location of the road defects.

RIA was built as a scalable, distributed system of components. The component called the ria-ui from Figure 1, is a server that handles all the interactions with the system – either via the web interface or the API. The ria-ui server was built using Node.JS and Express, while the ria-compute server was built using Python and Flask. In Figure 1, the ria-compute server carries out the computation and analysis that needs to be performed on the data. Results are stored in the databases and related images, or video outputs are stored in the object storage.

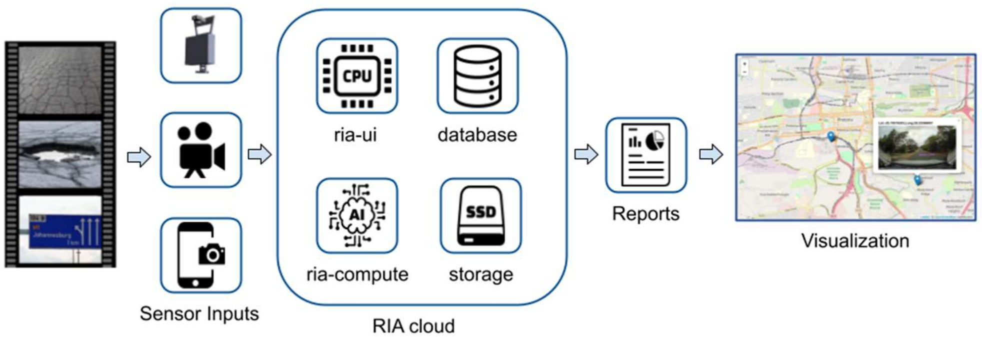
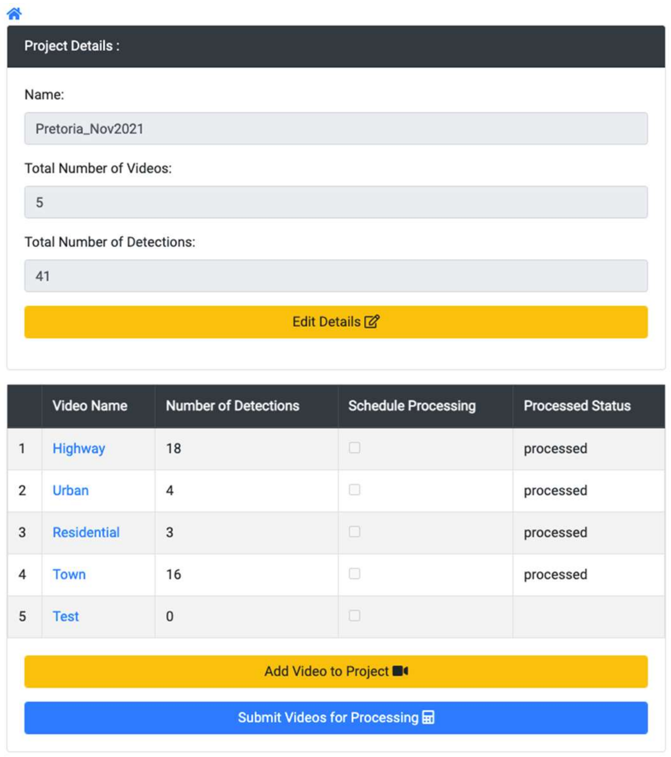


Fig. 1 Operational flow of the RIA System

During a typical use case, the user accesses the RIA system using a web browser. The user can create an account and log into the system. Once logged in, the user can create a project e.g., a project for each region that is to be surveyed. The user can then capture video data of roads in that region and upload them to the project. The videos can then be individually scheduled for processing. Using this workflow, the user does not need to wait online while the videos are being processed. The user can log off and return later to look at the results of the completed processing. The system can be configured to send notifications via email or WhatsApp to alert the user of the processing status. Figure 2 shows a part of the RIA user interface that is used to upload videos and schedule them for processing.



Project Details :

Name:

Pretoria_Nov2021

Total Number of Videos:

5

Total Number of Detections:

41

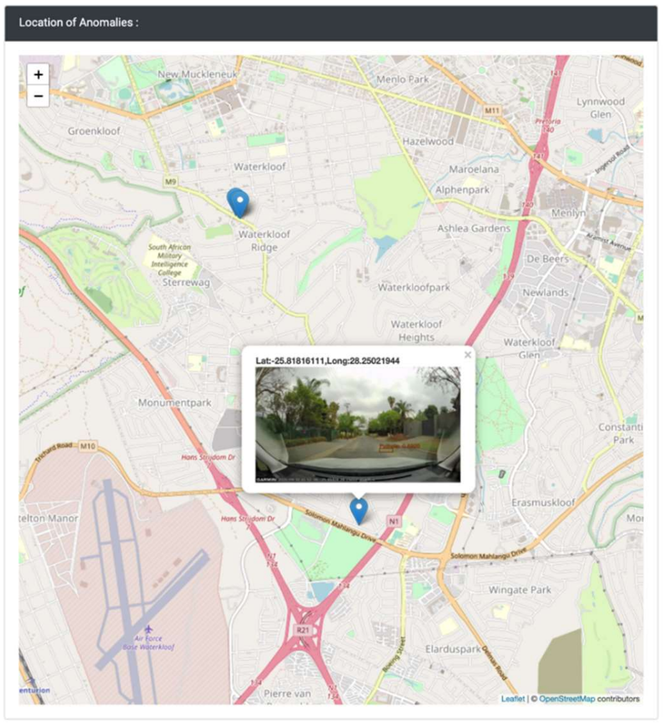
Edit Details

	Video Name	Number of Detections	Schedule Processing	Processed Status
1	Highway	18	☐	processed
2	Urban	4	☐	processed
3	Residential	3	☐	processed
4	Town	16	☐	processed
5	Test	0	☐	

Add Video to Project

Submit Videos for Processing

Fig. 2. RIA web-based user interface



Location of Anomalies :

Lat:-25.81816111,Long:28.25021944

Fig. 3. Map-based interface showing detected potholes

The following sections describes the components of the RIA system in more detail.

2.2 Sensors and System Inputs

In an earlier iteration of the project a custom sensor mounted to an inspection vehicle was used to collect video and global positioning system (GPS) data. The latest iteration was modified to use a Garmin 67W which is a commercial off-the-shelf (COTS) dashcam with built in GPS. The Garmin 67W has a 180-degree field of view, 1440p resolution, and a frame rate of 60fps. The specifications of the Garmin 67W allow for inspections up to a speed of 60km/h.

Reports of road defects sourced from the public using either the call centre or mobile applications can be uploaded into the RIA system using the RIA API. Reports coming through the call centre may or may not have images, but the information can be added into the system and later reported on.

2.3 Processing Pipeline

The ria-compute engine was implemented using Python with a range of supporting libraries. The model uses YOLOv3 object detection as implemented in the OpenCV library [3]. Through transfer learning the YOLOv3 model, trained on the COCO 2017 dataset [10], was retrained to recognise potholes.
There are three processing paths within the main ria-compute processing pipeline. These are the model training pipeline, the operational pipeline, and the model validation pipeline. The three paths are shown in Figure 4.

The model training pipeline is used to train the model to recognise potholes. First, videos of the road were collected. The videos were manually processed to extract frames and hand label the potholes to create a starting dataset.

This dataset was used to train and test a Convolutional Neural Network (CNN) model [4]. The model was then deployed into the operational pipeline.

In the operational pipeline, uploaded videos are pre-processed. The pre-processed image is then passed to the model for analysis. The detections and annotations that mark the location of the potholes are stored in a database and can be reported via the user interface. During the pre-processing step the video metadata is first extracted. The Garmin 67W produces video such that the GPS coordinates of the are embedded into each frame, therefore the GPS co-ordinates are extracted from the frame using Tesseract Optical Character Recognition (OCR) [5]. The output from the pre-processing is a timestamped frame with synchronised GPS information.

The detections can also be passed back to the manual validation pipeline which allows a human expert to validate the detections. The further validated data can then be used to retrain the model. After multiple iterations of training and testing the new model can be deployed into the operational pipeline. The manual validation and retraining, allows for the improvement of the system accuracy over time.

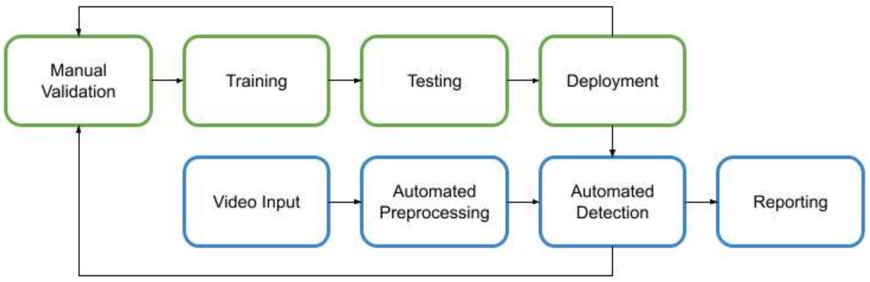


Fig. 4. Processing Pipelines

A variable frame rate processing strategy was used in the operational pipeline to speed up the model inference time. Video capture was performed at a framerate of 60fps. At 60fps a 30 second video contains 1800 frames. Running model inference on all 1800 frames took a significant amount of time. Even with GPU inference at around 50 milliseconds per frame it could take up to 90 seconds to process a 30 second video. With CPU-based inference significantly slower at around 830 milliseconds per frame, the same video could take up to 25 minutes to process.

To reduce the inference-time per video a speed-based approach was used to sample the road at a set distance interval. The vehicle GPS coordinates were extracted from the video and synced with the video frames. The GPS data was used to estimate the speed of the vehicle and extract frames every 10 to 20 meters. Using this method reduced the video processing time with 500% without a loss of road coverage.

2.4 Dataset Creation

Hand labelled pothole images were used to create a ground truth training dataset consisting of 317 images with 470 pothole instances. Images contained a combination of rural, urban and highway road types collected at different times of the day under varying weather and lighting conditions. Only daytime images with sufficient light were included.

As part of the ongoing development to improve the algorithms performance the dataset was extended by collecting new videos. During the data collection process, a modified version of the detection algorithm was used. The modified version allowed many false positives through. In this way the weaker model was used as a first level detector. These detections, with false positives, were then manually validated and hand labelled. The Computer Vision Annotation Tool (CVAT) [6] was used for the annotations and FiftyOne [7] was used to manage and explore the dataset. The new dataset would then be used in one or more cycles through the training pipeline until it is ready for deployment into the operational pipeline.

3 Results

3.1 Model training and comparison

Several modern object detection model architectures were considered for the application, these included YOLOv3 [8], Scaled-YOLOv4 [9], YOLOv5 [10], Faster R-CNN [11].

All the models were pre-trained on the COCO 2017 dataset and used for transfer learning on the pothole dataset. The pre-trained model performance on the COCO dataset is shown in Figure 5.

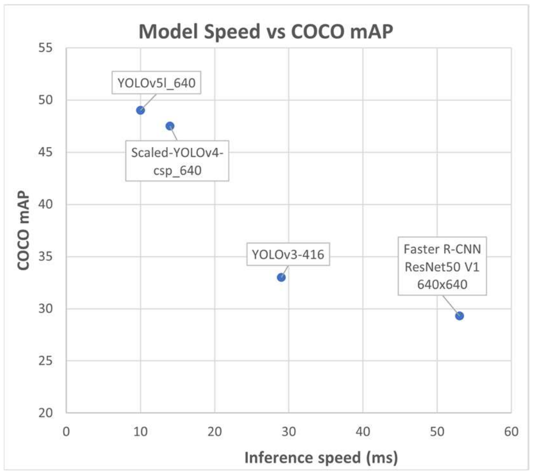


Fig. 5. Pre-trained model performance

Models were trained until convergence using the same test/train/validation data splits. Training was done using a two Nvidia GeForce RTX 2080 GPUs with colour images re-sized to 640x640 pixels and typically took around 2-3 hours with approximately 300 epochs. Various data augmentation techniques were applied as appropriate. Training results are shown in Table 1. below.

Table 1. Model training results

Model	mAP:0.5	Precision	Recall	f1-score
YOLOv3-416	0.75	0.85	0.80	0.83
Scaled-YOLOv4-csp_640	0.72	0.70	0.78	0.74
YOLOv5l_640	0.72	0.63	0.77	0.69
Faster R-CNN ResNet50 V1 640x640	0.73	0.68	0.80	0.73

YOLOv3 had the best performance with the highest mean average precision (mAP) at an intersection over union (IoU) ratio of 0.5. It also had the best precision, recall and f1-score. YOLOv3 is not the fastest model but was still selected due to its good overall performance. If inference speed were more important Scaled-YOLOv4 could be a good alternative at a 3% compromise in mAP:0.5.

The intersection over union (IoU) is a measure of the overlap between the predicted bounding box and the ground truth bounding box. This is calculated for each bounding box. Each bounding box would be associated with a class determined by the highest probability over all classes for the bounding box. The confidence is then given by P(Object) * IoU, where P(Object) is the probability of the object within the predicted bounding box.

3.2 Detection Results

The retrained YOLOv3 model was first tested on images containing potholes that were sourced from an online dataset [13]. The results of these detections can be seen in Figure 6.

Figure 6 shows each pothole surrounded by a bounding box with the confidence metric indicated in the upper left corner next to the label.

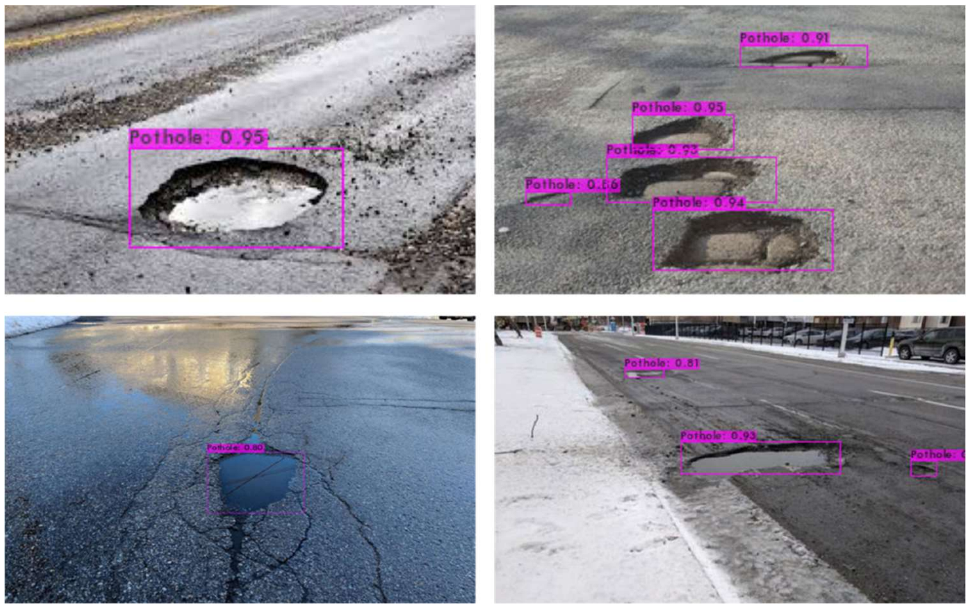


Fig. 6. Detections using externally sourced images containing potholes

The RIA system was then tested on images collected from urban, residential, inner-city roads, and highways on the Pretoria road network. The data was collected during regular day-time traffic conditions. The weather in all cases was clear and bright. The vehicle was driven at approximately 60-80km/h. Figure 7 shows some of the detection results that were achieved.

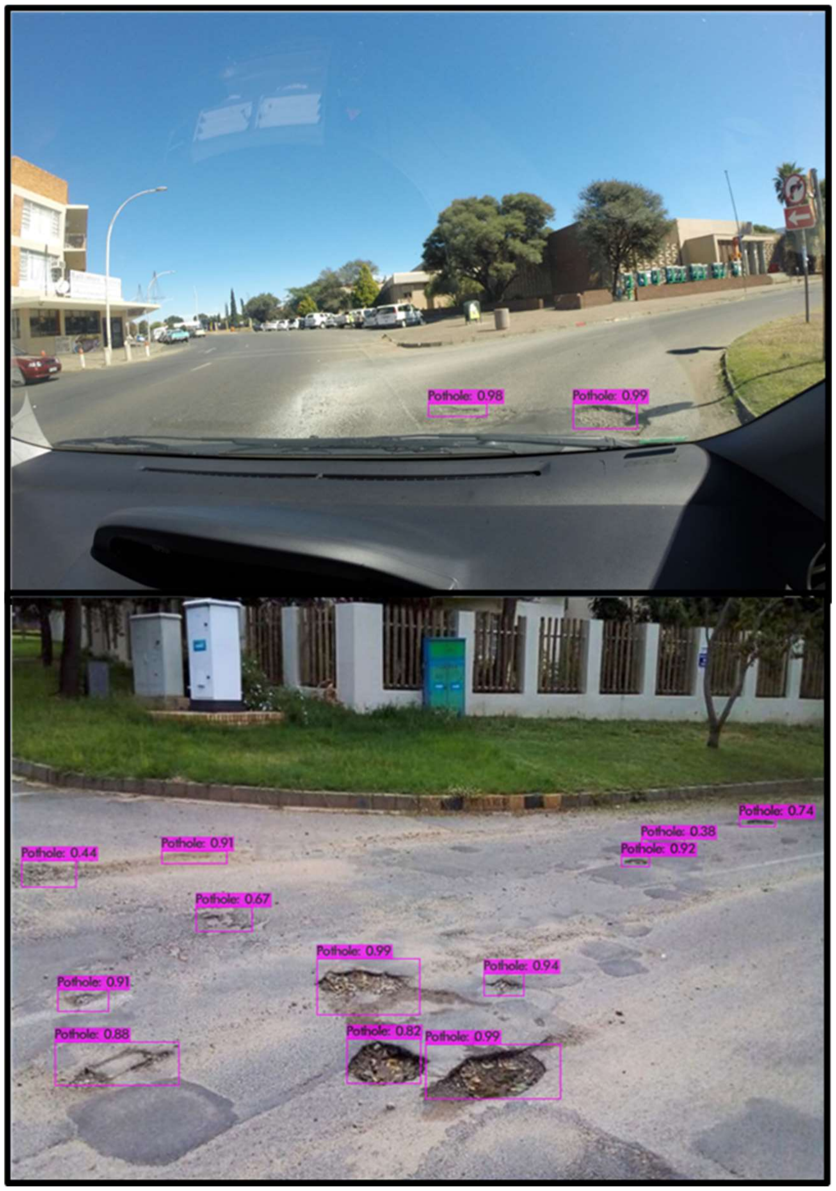


Fig. 7. Detections of potholes using retrained YOLOv3

4 Conclusion

This research presented RIA which is a cloud-based system to collect and analyse road data to detect and keep track of road defects and assets. In this research the pothole detection module was presented. The analysis algorithm was implemented using the YOLOv3 CNN with a detection accuracy of 75% mAP:0.5. Going forward the team will attempt to improve the accuracy of the pothole detection algorithm further.

The team has also experimented with a version of the algorithm that first performs a road segmentation to improve the detection accuracy and the detection speed. Next the algorithm detects the potholes within the area segmented as the road. Then the algorithm uses a calibrated camera matrix to estimate the area of the detected pothole. The estimated area is valuable because it is used to calculate the amount of material that would be required to repair the pothole. Figure 8 shows an output from the algorithm that performs road segmentation, pothole detection and pothole area estimation. In Figure 8 the estimated area is shown in the bottom right corner and is in metres squared. The team will continue to develop this algorithm to improve the accuracy of the detections and estimations.

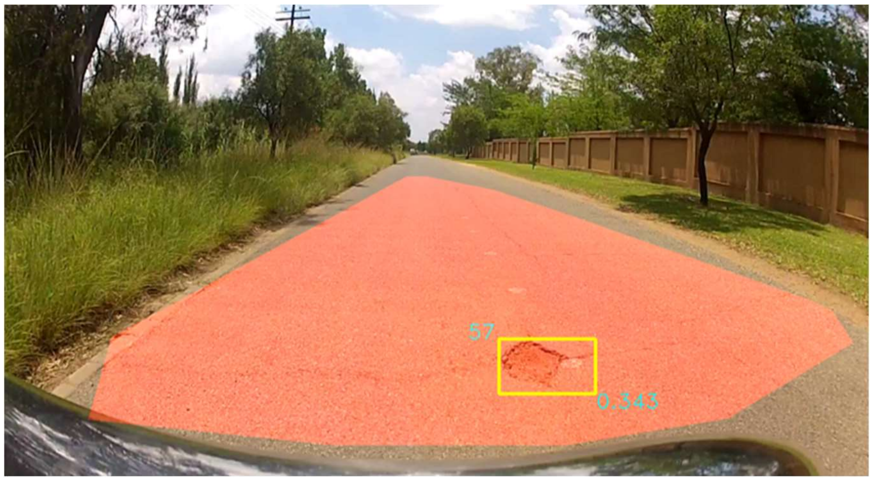


Fig. 8. Detection and size estimation of pothole

The team has also experimented with an algorithm to detect road signs. Figure 9 shows an output from the road sign detection algorithm. The road sign detection algorithm would be used as an input into road maintenance and compliance plans to determine the condition of road signs as well as if any signs may be missing.

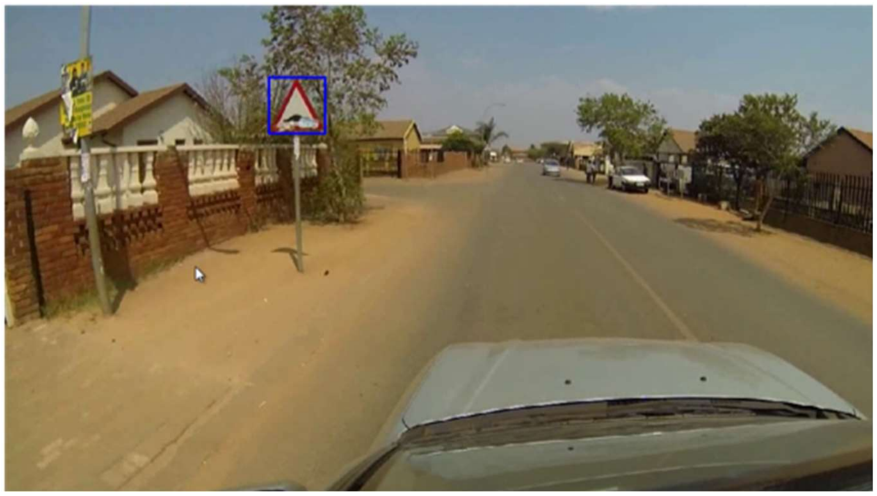


Fig. 9. Output from the road sign detection algorithm

References

1. Development Bank of Southern Africa (DBSA), *How improving road infrastructure in South Africa can help benefit the economy, society and health care,* https://www.dbsa.org/article/how-improving-road-infrastructure-south-africa-can-help-benefit-economy-society-and-health, last accessed 23 September 2022
2. Redhat, *What is Cloud Architecture,* https://www.redhat.com/en/topics/cloud-computing/what-is-cloud-architecture, last accessed 23 September 2022.
3. Bradski G. *The OpenCV Library*. Dr Dobb&#x27;s Journal of Software Tools. 2000.
4. Goodfellow I, Bengio Y, Courville A, *Deep Learning*, MIT Press, 2016.
5. Kay, A, Tesseract: an open-source optical character recognition engine, Linux Journal, Volume 2007, Issue 159, July 2007.
6. Intel, *Computer Vision Annotation Tool (CVAT)*, https://github.com/openvinotoolkit/cvat, June 2018, last accessed: 28 June 2022.
7. Moore, B.E., Corso, J.J., *FiftyOne*, `GitHub. Note:` https://github.com/voxel51/fiftyone, 2020, last accessed: 28 June 2022.
8. Redmon, J., and Ali, F. *Yolov3: An incremental improvement.* arXiv preprint arXiv:1804.02767, 2018.
9. Wang, C. Y., Bochkovskiy, A., and Liao, H. Y. M. *Scaled-YOLOv4: Scaling cross stage partial network.* arXiv 2020. arXiv preprint arXiv:2011.08036, 2020.
10. Jocher, G. *YOLOv5.* 2020. Accessed on: March 31, 2022. https://docs.ultralytics.com/
11. Ren, S. et al. Faster R-CNN: Towards real-time object detection with region proposal networks. Advances in neural information processing systems 28, 2015.
12. Lin, Tsung-Yi, et al. *Microsoft coco: Common objects in context,* European conference on computer vision. Springer, Cham, 2014.
13. Nienaber, Sonja. Detecting potholes with monocular computer vision: A performance evaluation of techniques. Diss. Stellenbosch: Stellenbosch University, 2016.