ONTARIO TECH UNIVERSITY
FACULTY OF SCIENCE, COMPUTER SCIENCE

Project April 5, 2024

Helping Municipalities Maintain Roads by Locating Defects Using Computer Vision

by

Samuel Bazinet

Abstract:

In a world where efficiency is more important than ever and with roads being built all the time, finding road defects needs to be as automated as possible to maintain a good driving experience for all citizens. With computers getting faster by the year, we can use this newfound ability to use them to help us automatically find defects such as potholes and cracks without human intervention. This paper looks over how we can setup such a system, including finding an algorithm to find potential defects, as well as training a machine learning model to differentiate between the types of defects.

Keywords: Computer Vision, Infrastructure

The code used for this project can be found at: `https://github.com/samuel-bazinet/cv_project`

1 Introduction

This course on computer vision started with a lab where stop signs had to be identified on an image. This was relatively simple as stop signs are standardized so they can be easily identified. Unfortunately, the same cannot be said for road defects. We cannot simply use a single default template and expect to get a substantial true match rate. This means that a substantial amount of work needs to take place in order to be able to outline the potential defects on the road. This paper will go over the efforts required to get a relatively competent model to detect defects on an image of a road.

2 Methodology

Detecting defects on a road is a somewhat complex procedure involving many steps to get the model going. As we are trying to find cracks and potholes, we cannot use simple pattern matching as they very a lot in size, color, and pattern. We need a machine learning model for this. To achieve the model, a dataset will need to be created, then the model will be trained, before being used to highlight the defects on images.

2.1 Finding Potential Defects

The first part of finding defects on road is to scan the image for potential defects. Cracks and potholes tend to be very well defined on roads, meaning that Canny edge detection is great way to find them. But since roads can contain many edges, a blurring filter is ran beforehands to reduce the noise in the resulting edges. Then, the program scans the image and groups adjacent edges together into their own objects so that we can extract the potential defects as their own images for later usage. To make this easier, a dataclass called ‘Defect’ was created. This dataclass contains the coordinates of the corners of the area containing all the edges, as well as a list of pixels of where the edges were detected. This class has a few helepr methods to merge potential defects contained within each other as the assumption was made that they would be part of the same defect. It also has a method to “recenter” itself around the densest collection of edges for the purpose of training the model with evenly sized images. 50 images of damaged roads were ran through this “potential defect extractor” to collect hundreds of potential defects

that will be used for training the model that will be used to detect the real defects. A subset of those pictures.

2.2 Training a Model to Classify the Defects

There are 3 classes that will be used by the model: crack, pothole, and non-defect. To train the model, the images of the potential defects generated by the last step have been manually classified into 3 folders, one for each class. Keras and TensorFlow were used to train the model as they were easy to use and have plenty of online resources for extra help. The model startes by rescaling the data from a 255 base to 1 base so that it is faster to work on. The model then performs a 2D convolution with 16 3×3 filters, which are activated by ReLU. It then goes through a max pooling layer with a 2×2 window and a stride of 2. There is another 2D convulution but with 32 filters, again activated by ReLU. There is then a flatening layer to convert the data into a 1D vector. To help fight overfitting, there is a dropout layer to drop 30% of the input units. Finally, there is a dense layer with 512 units activated by ReLU. The model then goes through a final dense layer to link up to the number of classes. The data was augmenting using the Keras sequential augmenter, where the data set had images copied with different rotation and zoom levels. This is quite useful as generating more images for training is very time consuming. The model was trained over 100 epochs, where the training accuracy went from low 30% to around 80% and the validation accuracy reached 70% (see plot). The model was saved for access by the actual defect identifier.

2.3 Applying the model to potential defects and highlighting them on an image

Using the same method as defined in Section 2.1, potential defects are found on the target images. The model is then loaded into the Jupyter notebook and the potential defects are passed through it, where the predictions are obtained. Using the prediction from the model, we draw a red line around predicted potholes, and a green line around predicted cracks. Non defects are not highlighted. This results into an image where the defects are highlighted so that the target audience will be able to easily see the work that needs to be done.

3 Results

The model was tested on a few pictures with mixed results. The model is somewhat competent at identifying potholes and cracks, but it often mislabels them and misses quite a few defects as well. It sometimes falsely labels non defects as defects, which is expected from a low accuracy model but is not desirable. It is quite sensitive to the resolution of the images and the low validation accuracy is made quite apparent.

4 Discussion and Conclusions

So is this bad performance just due to the model being sub-par? Quite frankly, the method used to extract potential defaults could be much better as it is sensitive the resolution of the image. This sensitivity is mostly due to the fact that every potential defect needs to be the same size to work on with the model. This causes images of higher resolution to have the defect being cropped, thus making it harder for the model to both train and recognize it. Inversly, images of low resolution may have defects that are too small and may include noise that distracts the model. The method also works under the assumption that defects are often near edges detectable by canny edge detection, but that method has its limits and affects the potential defect extracted from an image. The dataset used to train the model only contained a bit over 400 images, which isn't large and limits the "bounds" of the model so that it will have more trouble on a new image. This is somewhat mitigated by data augmentation, but that method only goes so far as it modifies the current data and does not introduce new "clues" for the model to use.

Would this model be useful for the target audience? In this state, not really. But it is a good fondation that can be built upon by giving it more data and making it less sensitive to the original image resolution. There were not obvious signs of overfitting in the model, so it is possible that more epochs will give a better accuracy, at the cost of longer training. The detection of potential defects can also be updated to either be more sensitive or be able to mask off parts of the image that isn't a potential defect. This could be done by using a filter to put a strong blur on the area that is definitely not part of the defect so that the model will be less distracted.

Overall, this was an insightful dive into finding out what it takes to find defects on

a road through the usage of edge detection and machine learning. This is a rabbit hole of endless optimization but this just went through a very naive approach to solving a very complex problem that has very real consequences.

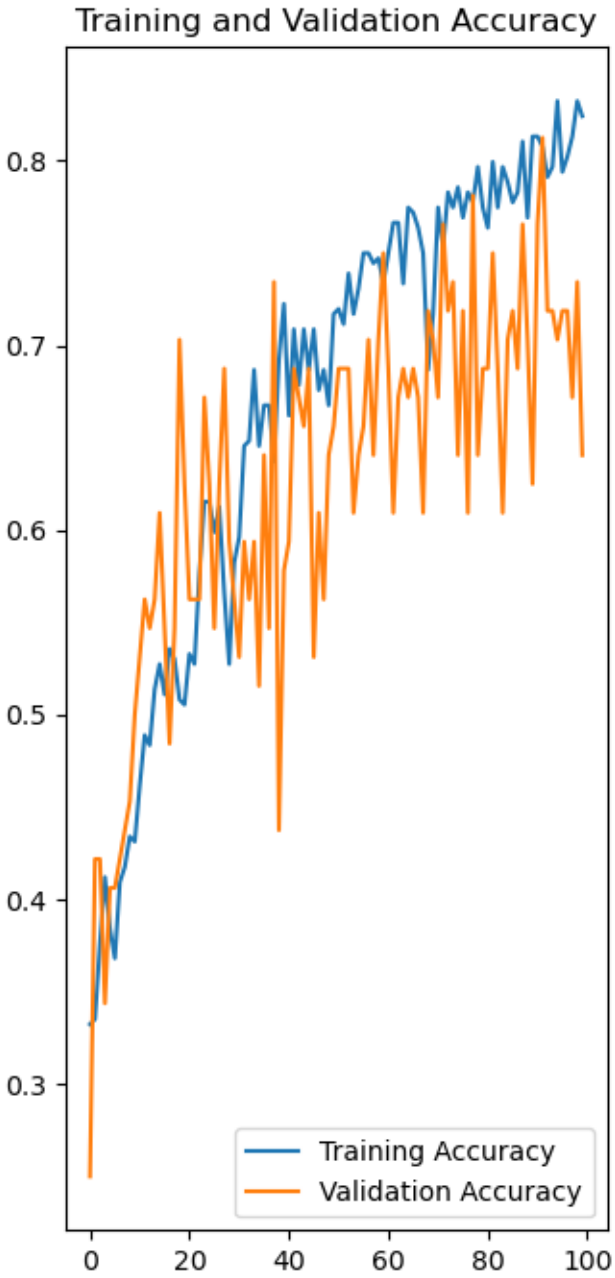
A Sample images used in training the model

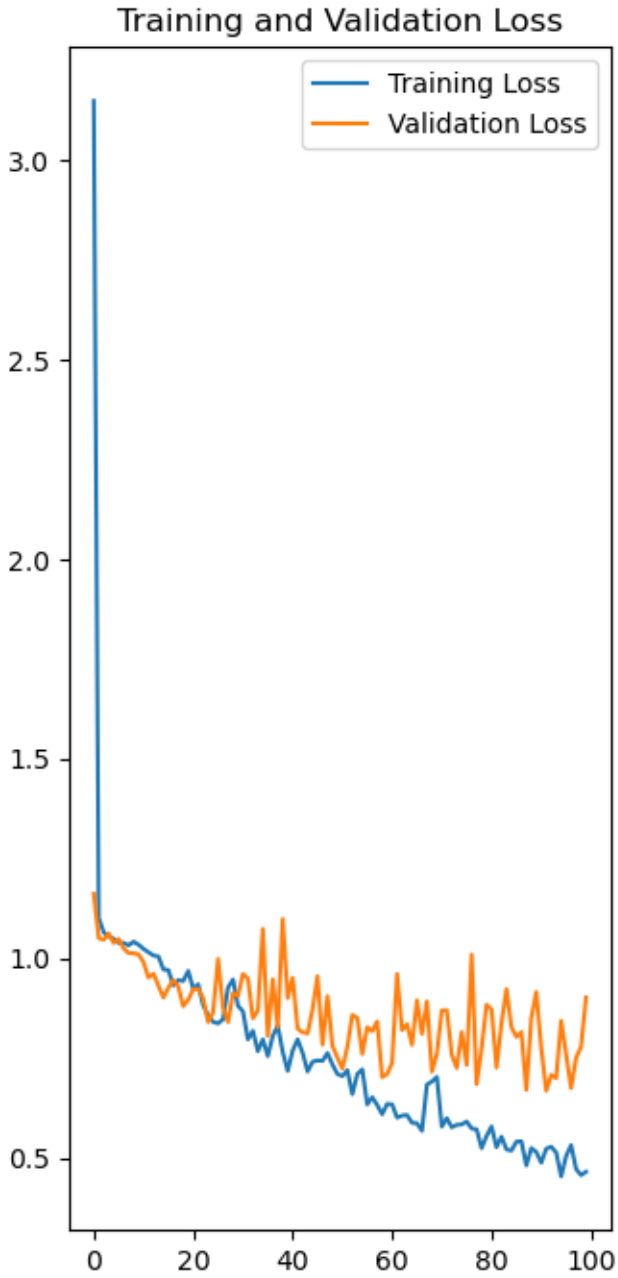
A.1 Non defects

A.2 Cracks

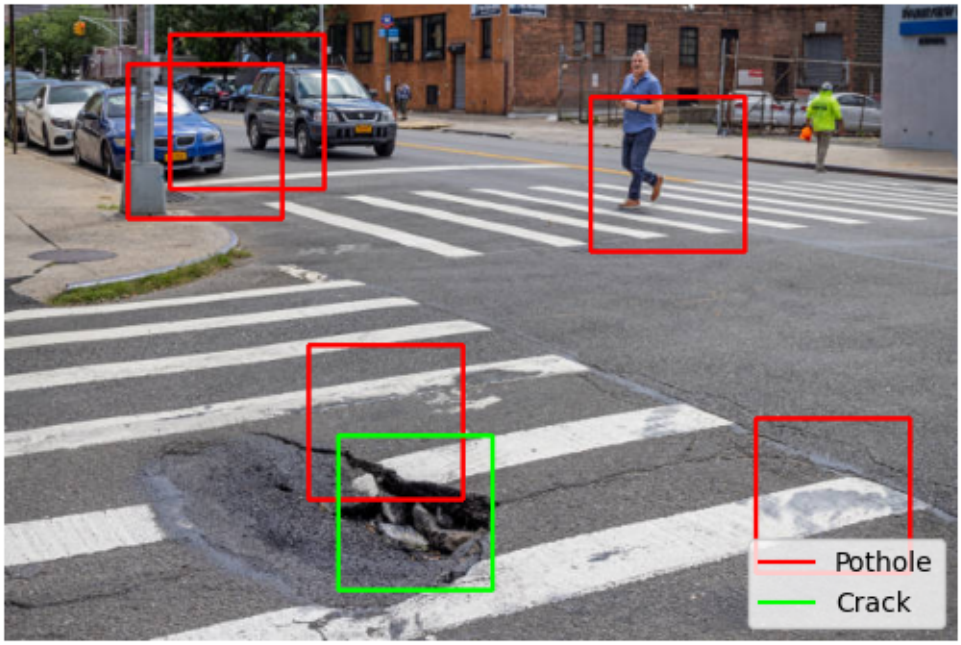
A.3 Potholes

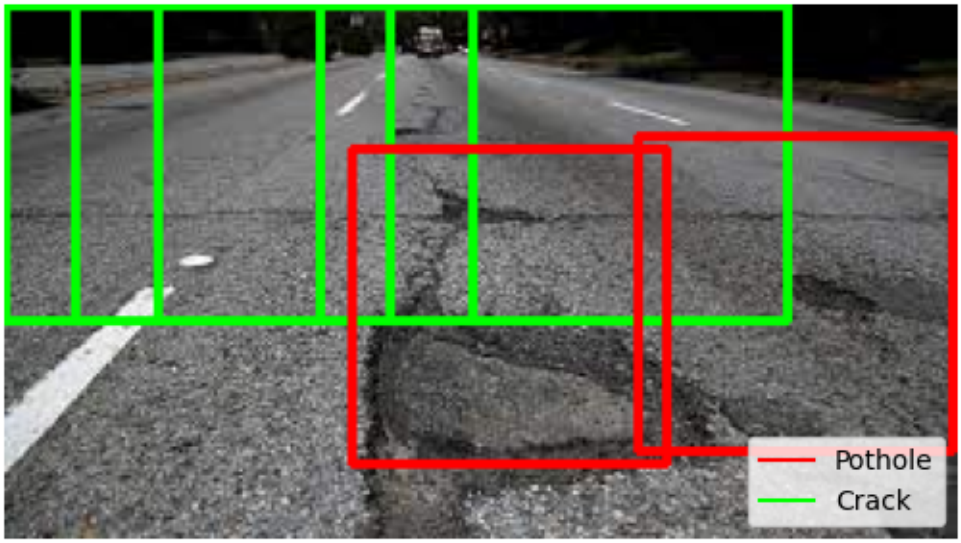
B Model training plot

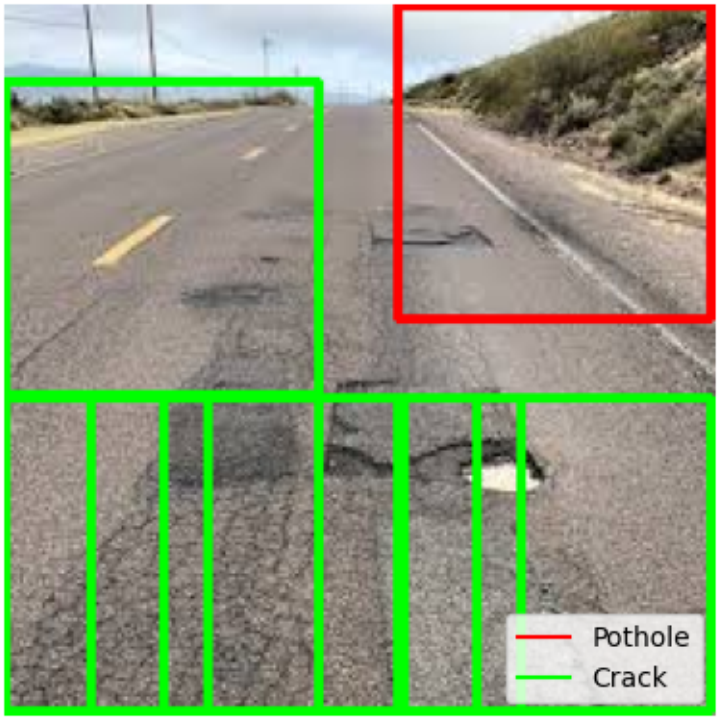


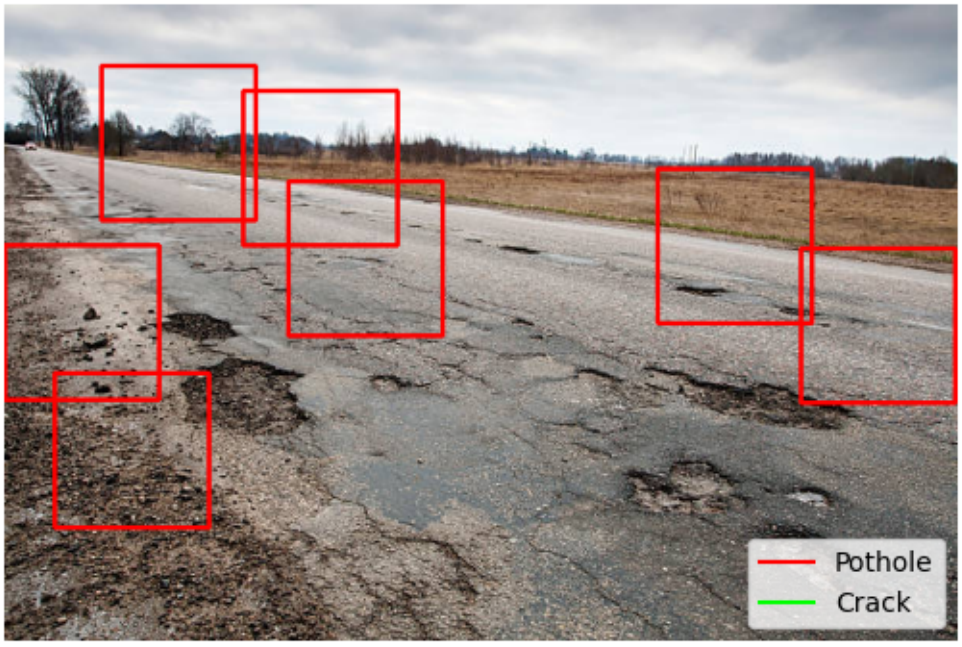


C Sample resulting images






Pothole
Crack


Pothole
Crack