# 

# 

# COMP 4613 Machine Learning

# Classification Project:

# Nova Scotia Pothole Detection

#### Prepared By:

#### Benjamin Kelln

#### 100145180

##### 

##### 

##### Date:

##### March 12th, 2020

## 0. Table of Contents

[0. Table of Contents](#_44z7snuuahih) 2

[1. Abstract](#_j18sk0u7qqi0) 3

[2. Introduction](#_rl1qsi3wyddv) 3

[2.1 Problem Definition](#_9q6qiju0ffhh) 3

[2.2 Overview of Pothole Feature Detection Image Theory](#_v56ap0ftqlid) 4

[2.3 Expected Success Criteria](#_9q6qiju0ffhh) 5

[3. Data Description, Collection and Preparation](#_7rhf3f5i51xe) 6

[3.1 Description](#_aio13fd45dpu) 6

[3.2 Collection](#_aio13fd45dpu) 6

[3.3 Preparation](#_43p0yqohlr4o) 7

[3.4 Image Preparation for Enhanced Image Augmentation](#_w131d8ufcg9r) 7

[4. Overview of Approach](#_yv0cjczeudzq) 8

[4.1 CNN Architecture with VGG Blocks and Baseline](#_ll9lcydog7ib) 8

[4.1.1 Introduction to CNN, VGG Blocks and a Baseline](#_l923wc4paf3q) 8

[4.1.2 Establishing a Baseline](#_ovc5ts396vr6) 9

[4.2 Machine Learning Methods](#_ll9lcydog7ib) 10

[4.3 Experimentation w/ Machine Learning Methods](#_ll9lcydog7ib) 10

[4.4 Exploring our Augmented Data](#_7msz9bo0obv5) 12

[4.5 Exploring Alternative Methods](#_v8e90zgx2edd) 12

[4.6 Final Model](#_kq8i4bhav6tu) 14

[5. Results Evaluation](#_6y62ypmy2vqr) 15

[5.1 Normal Model Accuracy on Original Dataset](#_63fm4jxd8grq) 15

[5.2 Normal Model Accuracy on Nova Scotia Dataset](#_68dcsfv7s6qc) 16

[5.3 Model Accuracy on Original Dataset with Enhanced Image Augmentation](#_63fm4jxd8grq) 16

[5.4 Model Accuracy on Nova Scotia Dataset with Enhanced Image Augmentation](#_63fm4jxd8grq) 17

[6. Conclusion](#_jaygsbwqo904) 18

[6.1 Overall Evaluation](#_ctxf3590iw3r) 18

[6.1.1 Normal Model Accuracy on Original Dataset](#_eqyc90s86jn9) 18

[6.1.2 Normal Model Accuracy on Nova Scotia Dataset](#_x41ibyrkphlb) 18

[6.1.3 Augmented Model Accuracy on Original Dataset](#_rgusntj2vc7p) 18

[6.1.4 Augmented Model Accuracy on Nova Scotia Dataset](#_vdf99a4luj93) 18

[6.1.5 Results Comparisons](#_650jxpymki1) 19

[6.2 Future Work](#_ctxf3590iw3r) 20

[7. References](#_wt03t6hu0kwm) 22

## 

## 1. Abstract

Potholes are extremely dangerous. Not only can potholes damage vehicles but they also pose a legitimate threat to human life inside and outside of vehicles. Potholes can be very difficult to see for the human eye for a variety of reasons such as bad weather, being covered in water or even it being too dark at night. Nova Scotia is known for having infamously bad road conditions due to the defined dry and wet seasons, which leads to many many potholes. In this project, I hope to develop a model that can classify stock images of roads as having a pothole or not, and then to expand our classification to images that I have taken of Nova Scotian’ roads with and without potholes. For this project we will be using 100x100 greyscale images of roadways to build CNN models that can accurately detect potholes in an image of the road, and then determining whether our model can accurately predict on images of Nova Scotia roads.

## 

## 

## 

## 2. Introduction

### 2.1 Problem Definition

As we’ve previously mentioned, potholes can be very dangerous. Although it’s hard to say exactly how dangerous they are as there isn’t too much research done on specifically potholes, there have been articles such as in India about how potholes can cause upwards of 4,000 deaths a year or even 10 deaths a day [1]. I believe that this can largely be attributed to lack of detection of potholes by the human eye, and especially when seeing conditions are already poor such as in bad weather or at night time. Furthermore, as artificial intelligence continues to become more and more popular, the popularity of autonomous vehicles that are self-driving will also continue to rise. If we want to ensure that these vehicles are safe, (especially on Nova Scotian roads) we want to be able to ensure that these vehicles can easily detect and avoid potholes in the road. Specifically, we want to address the problem of pothole detection by teaching a convolutional neural network how to detect and classify whether a photo has or does not have a pothole.

### 

### 2.2 Overview of Pothole Feature Detection Image Theory

I will go more in-depth into the image dataset that we will be using to develop our pothole detection model, but I first want to dive into a hypothesis that I thought of while reviewing the dataset. Upon coming across our dataset of roads with and without potholes, I started to realize that there are a lot of similar features of images with a pothole. They typically are very round, a different color than the road around it, and a lot of the time they are filled with water and thus look very different from its pavement surroundings. My theory is that if we can emphasize these features that I’ve noticed using image augmentation, maybe we can develop a better model that can be even more accurate at detecting potholes. The main data augmentation that I would like to experiment with is contrast and exposure. Take a look at the photos below and see if you can see what I am talking about. The pictures on the left are taken directly from our pothole road dataset, while the pictures on the right are the same picture but with data augmentation of increased exposure and contrast. To me, the pictures on the right have much more distinct pothole features than the pictures on the left. The very last image is a road without a pothole where you can see that our augmentation doesn’t really change the features of the photo. With this in mind, I plan to try and develop an entirely different model by training it only on augmented images such as the ones below, and see how it compares to our original model. For future reference, we will refer to pictures with this augmentation to be called the ‘augmented dataset’.

| Original Image | Augmented Image (↑Contrast + Exposure) |
| --- | --- |
|  |  |
|  |  |
|  |  |
|  |  |

### 2.3 Expected Success Criteria

For our problem of pothole detection, we have a couple different success criterias that we expect to reach as we need to separate the success criteria between the original dataset and the augmented dataset, as well as distinguish between stock road images and Nova Scotia road images. The first set of success criteria that we will define will be whether or not we can accurately detect potholes from an image of a road using our original dataset. We hope to reach a conservative 75% accuracy with this model, and we hope to have a 70% accuracy with this model on the prediction for Nova Scotia roads. Our next set of success criteria is the same as the last, but instead we train our model and make our predictions using the augmented dataset. Because we expect this model to do better, we hope to reach a conservative 80% accuracy of the detection of potholes on images of our augmented stock road images dataset, and a 75% accuracy on the detection of potholes on Nova Scotia roads.

## 3. Data Description, Collection and Preparation

### 3.1 Description

The data that we selected for this project is a collection of images of roads. The dataset comes from the website Kaggle.com where users can upload their own data. Our specific pothole dataset comes from the user Atulya Kumar who explained that it is a collection of more than 600 images of roadways which were simply scraped from the internet. The images are in full color and have no set dimensions, which is something we need to take into consideration when preparing our data. The structure of the dataset is fairly simple, in that the classification of a photo (whether or not it has a pothole) is clarified simply by the folder that it is within. This is a visual of what the structure of the downloaded dataset looks like:

### 3.2 Collection

As we have already experimented with convolutional neural networks from class assignments, I had learned quite a bit about how to import data into Google Colaboratory by simply uploading to Google Drive, and thus that is the strategy that I decided to implement again. But, it was not as simple as just downloading the dataset and uploading it to the drive. Instead of splitting our data into train and test sets within our Google Colab code, I would manually split the data into separate folders beforehand. This was done for ease of use, but also because of how we plan to develop multiple models using different training data (the augmented data set). When training our final model, we will ensure that we iteratively use k-fold cross validation splits of our training set for our validation set, so that we can guarantee the accuracy we ultimately achieve. Our individual test sets contain 75 images each of their respective classification/folder. Very shortly we will dive deeper into why we manually separated our datasets when we talk about how we prepared our data for the augmented image dataset, but for now this is what the structure of our dataset looked like after splitting into a train and test set:

### 

### 3.3 Preparation

Of course, in order to develop our convolutional neural network we need to convert our image into usable input. We can easily do this by using Keras image preprocessing methods, and more specifically the ImageDataGenerator function. This will allow us to convert our images into batches of tensorflow image data that we can then use to train and develop our model. Not only that, but this function also allows us to augment our images to fit any other image augmentation requirements that we have. For example, we of course will need to normalize our pixel values by scaling by 1/255. We will definitely be using the grayscale function to change our images to grayscale color values, and we may implement other randomly generated image augmentations in order to make a more robust model.

One last point we need to make falls back to the sizes of our images. Convolutional neural networks require that all images be of the same dimensions, and thus I have set the dimensions of all of our images to 100 x 100 pixels. This should allow us to maintain the prevalent features within our images that our model will use to detect, while compressing our data to more manageable chunks.

### 3.4 Image Preparation for Enhanced Image Augmentation

Our augmented dataset proved to be a little bit difficult to create. My first approach was to simply use Tensorflow’s image preprocessing methods on the fly as we processed the images, but I found this method to be inconsistent and more importantly didn’t accomplish what I wanted in increasing the exposure levels of the augmented images.

I decided that instead I would see if there was a way for me to augment these images myself using an external tool, and then to do the same as before in manually creating directories/sub-directories for each of the categories (train, test, pothole, normal). And this is exactly what I had done. Using a program called *ImBatch*, I was able to upload and edit hundreds of images at the same time, then save copies of these augmented images to new folders. The data augmentations in the exact order of application that I decided upon are: brightness + 25%, contrast + 60% and exposure levels + 0.5. Refer back to section 2.2 to see what these augmented images look like compared to the original image. Our new file structure is the exact same as before, except now there is another archive folder that contains our augmented images. The augmented images dataset is divided by the same train and test sets, of which each of these sets contains the exact same images as it’s original image dataset counterpart.

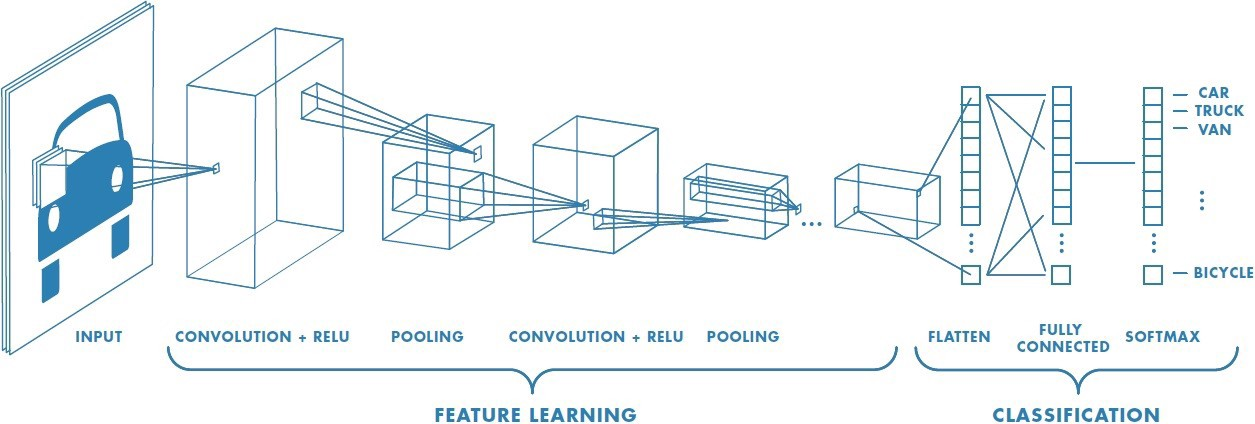
## 

## 4. Overview of Approach

### 4.1 CNN Architecture with VGG Blocks and Baseline

#### 4.1.1 Introduction to CNN, VGG Blocks and a Baseline

As we’ve said, we’ll be using a convolution neural network to build our model. These types of networks are great for image classification problems, such as ours, as CNNs are great at identifying features. In essence, that is what and how a CNN works; by assigning weight or importance to extracted features to try and identify what is within image data. Below is a great diagram showing a breakdown of a CNN architecture:



<https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53>

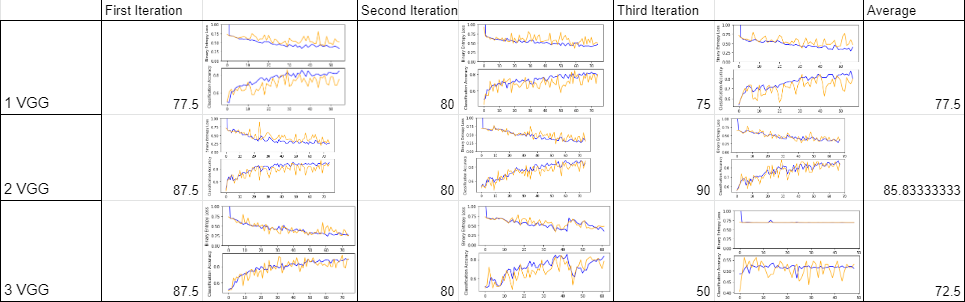
The next topic that I’d like to quickly explain before diving into our code is that of VGG Blocks and establishing a baseline. VGG-style networks use ‘VGG-blocks’ of feature detection in the form of a single convolution layer followed by a single max pooling layer; then these blocks are stacked on top of each other for feature detection. VGG-blocks are extremely useful not only because of their easy implementation and great ability for feature detection, but because they provide us with a perfect baseline that we can try and improve upon [[x](https://machinelearningmastery.com/how-to-implement-major-architecture-innovations-for-convolutional-neural-networks/)]. Using methods that we will discuss more in depth shortly, we will try to achieve a better result (accuracy in our case) than what we achieve on our baseline VGG model.

#### 

#### 4.1.2 Establishing a Baseline

To establish our baseline, I wanted to first see how many VGG blocks we should be using. From previous assignments, we have noticed that typically 3 VGG blocks work really well for image classification, but we should still try different numbers of VGG blocks to see if we get different results.

The following table displays the results from running 3 iterations of building CNN models with various numbers of VGG blocks. Each iteration ran for 75 epochs (unless early stopping stopped it between 50-75), on images of size 100x100 pixels, using activation functions relu except for the final activation function being sigmoid for our binary classification. The value in the table on the left of each iteration represents the accuracy of classification on the test set of 40 images after the model had been built.



<https://docs.google.com/spreadsheets/d/1TEknWGXQ26uPiLzGefmva5B49xwJSEZtHOuT7gXWae0/edit?usp=sharing>

At first glance it appears that our results are all over the place, but when we take a deeper dive into the graphs we can see that our model is not consistently generalizing to our validation and test sets, and thus the varying results of averages. If you look at the blue lines of our graphs that represent our training data accuracy, the model is performing fairly well around the 75% accuracy mark while the yellow validation accuracy line is sporadically jumping up and down. We also notice that for some reason in our 3 VGG block: Third Iteration we have a terrible accuracy of 50%. It’s hard to say exactly why this occurred and honestly it’s a little concerning, but we would theorize that this is because the model is given a measly 75 epochs to train and was even stopped early. Had the model been given more time to train, I would predict that this outlier would not have occurred. We’ll use at least 100 epochs now for the rest of our experiments, and I realize looking back at the fact that this was a poor execution of setting a baseline.

As for setting a baseline, it appears that 2 and 3 VGG blocks had performed very similarly, and given that our problem is not overly complex (as it is only a binary classification), we expect that 2 VGG blocks should be more than enough. Our results for only 1 VGG block was a better accuracy than we anticipated from the get-go, and so getting an even better accuracy with 2 VGG blocks will work perfectly for our baseline accuracy. We will cautiously state that our baseline accuracy on 2 VGG blocks is 75%, as we are well aware that only 3 iterations of model generation is not enough to warrant certainty.

### 4.2 Machine Learning Methods

As we mentioned, our model was performing very well in the high-70% range on our training set but was not generalizing well to our validation + test sets. This tells me that we absolutely need some generalization and normalization techniques to try and develop a more robust model. The machine learning methods that I would like to test and help find a more accurate model are: Dropout, Batch Normalization, Learning rate variation and momentum variation.

Dropout will absolutely help us with generalization as dropping nodes while training is a great way to stop our model from relying on specific features, while batch normalization should provide us with a faster-learning model that can generalize to our held-out validation set more accurately. Finally we’ll take a look at slightly altering our learning rate and momentum to see if we can find a more optimal model.

### 4.3 Experimentation w/ Machine Learning Methods

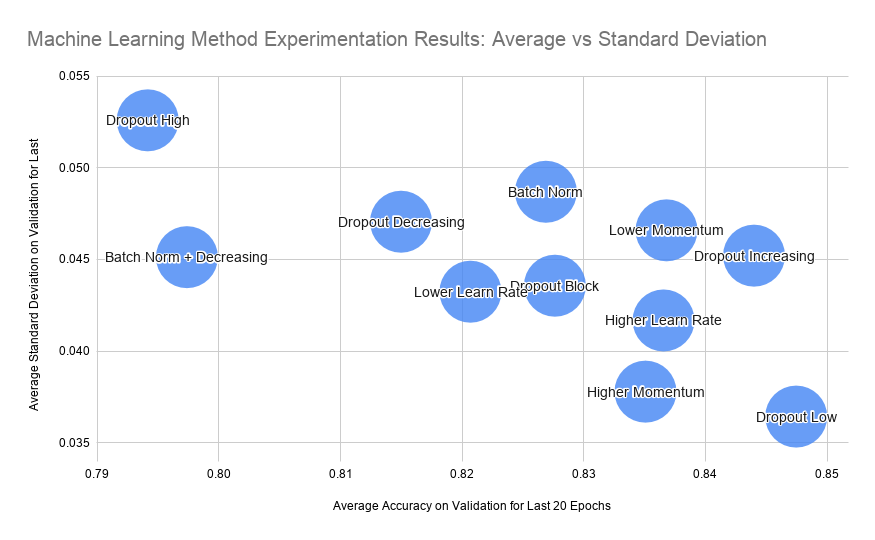
The data in the table below was collected from the following experiment:

As we want to experiment with our machine learning methods to try and develop a more robust model, there are a lot of different variations to the methods that we can make. For this experiment, I decided upon the following categories of variation and model generation: Dropout, Higher Dropout, Lower Dropout, Increasing Dropout, Decreasing Dropout, Batch Normalization, Lower Learning Rate, Higher Learning Rate, Lower Momentum, Higher Momentum. All categories with ‘Higher’ or ‘Lower’ is relative to the value used for our baseline with 2 VGG blocks, or to the baseline found through this experiment (only for Dropout).

Our goal with this experiment was to make our model able to generalize better, and so the accuracy that we want to improve upon is the accuracy of our validation set being compared as our model is being trained. We approached this experiment very similarly to our baseline, in which we train 3 different models with the specific category in mind; all the while seeing the accuracy on our validation set; for what is now 100 epochs. All images are 100x100 pixels, color shifted to grayscale, and are trained on convolutional neural networks with 2 VGG blocks. The data points within the table were obtained by averaging the validation accuracy / validation accuracy’s standard deviation on the last 20 epochs of only 1 of the 3 model iterations, and then are again averaged between the 3 different model iterations. Here are our results :

| **Last 20 Epochs** | **Average** | **Standard Deviation** |
| --- | --- | --- |
| **Dropout Block** | 0.8276301587 | 0.04356482104 |
| **Dropout Low** | 0.847468254 | 0.03642638741 |
| **Dropout High** | 0.7941492063 | 0.05258543303 |
| **Dropout Increasing** | 0.8440015873 | 0.0452043898 |
| **Dropout Decreasing** | 0.8149777778 | 0.04705502894 |
| **Batch Norm** | 0.8268873016 | 0.04868935373 |
| **Batch Norm + Decreasing** | 0.7973761905 | 0.04512202017 |
| **Lower Learn Rate** | 0.8206857143 | 0.04323775529 |
| **Higher Learn Rate** | 0.8365619048 | 0.04166979254 |
| **Lower Momentum** | 0.8368063492 | 0.0465851982 |
| **Higher Momentum** | 0.8350761905 | 0.03778685155 |

[Link to Spreadsheet](https://docs.google.com/spreadsheets/d/1bTKGxAV_QX9t-urhkSYdYlU6gRjCe2b2bwGK9RbYZoQ/edit?usp=sharing)



There’s quite a lot for us to unpack here, so let’s see what information we can take away from this experiment. As we’ve said, we want a highly accurate model that can generalize well. This means that for our chart we want the highest accuracy percentage, and the lowest standard deviation which corresponds to being as far right on the x-axis and as far down on the y-axis as possible. Before we go into the categories, right off the bat we can see that our accuracy has improved. For our baseline of 2 VGG blocks we established a baseline accuracy of about 75% on our test set, yet these new models can already be seen to have an average accuracy on our validation set as over 79%, with most of these experiment models having accuracy of about 84%. I would theorize that this is due to our increased number of epochs from 75 to 100, but am positive that these machine learning methods are helping us out as well.

A surprising piece of data that we can attain from the table is the difference between High and Low Dropout. As we’ve said, the machine learning method Dropout is already helping with our validation average but it appears that our model was more accurate with less dropout as opposed to more. Not only that, but a smaller percentage of dropout did better than our ‘baseline’ percentage of dropout (called Dropout Block on the bubble graph), and so we will use small percentages of dropout for our final model. We can also see from that graph that our validation accuracy was better with an increasing dropout, which will definitely help with not rolling over our global minima if we decide to implement batch normalization. The last piece of information that I’ve taken from this experiment is the fact that varying our learning rate and momentum didn’t seem to do much, and thus we will keep both of those values to something regular.

### 4.4 Exploring our Augmented Data

Unfortunately, it took absolutely forever for Google Colab to process our giant experiment previously mentioned as well as the experiment that we will talk about in the next section. If any more time was spent trying to develop an entirely new model based off of our augmented dataset, we would run the risk of not being able to finalize our pothole classification model and produce a great report. Unless time permits, from this point onwards we will not be trying to develop an entirely new model structure based on our augmented data + experiments, and we will instead see whether or not our model structure for our normal dataset can be used to train a new model based on our augmented dataset, and still accurately detect potholes. I do not necessarily expect this augmented model to be more accurate than our normal model now, but I will absolutely remember to take more consideration into how long experiments within a time frame may vary in the future.

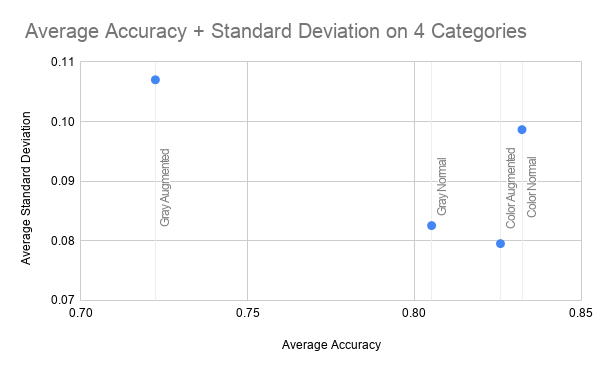
### 4.5 Exploring Alternative Methods

While working through our experimentation, I started to wonder how big of a factor color would play into our feature detection. When looking through the dataset it is quite easy to see that our images without potholes are much more colorful than our images with potholes, and I was curious to see whether our CNN would be able to see that as well, and use color to it’s advantage of detecting potholes. On top of that, I was also curious to see whether color would be a factor in our augmented data set as well. I felt as though because we were altering the levels of contrast and exposure that it could actually play a large part in how well our model can classify.

The following table and graphs data was collected by the following experiment, with calculations that can be found within the [Project Calculations Spreadsheet](https://docs.google.com/spreadsheets/d/1bTKGxAV_QX9t-urhkSYdYlU6gRjCe2b2bwGK9RbYZoQ/edit?usp=sharing).

This experiment consisted of 4 different models with the following titles: Gray Normal, Gray Augmented, Color Normal, Color Augmented. The term Gray v.s. Color is comparing whether the data was retrieved as a grayscale image or with rgb colors, and the term Normal v.s. Augmented refers to whether the model was developed and evaluated on our normal or our augmented image dataset. Each category was trained on 150 epochs for 3 iterations using a convolutional neural network that contained 2 VGG blocks and machine learning methods batch normalization and dropout. This is the resulting averages of the last 50 epochs for each category:

|  | Average Accuracy | Average STD DEV |
| --- | --- | --- |
| Gray Normal | 0.805145098 | 0.08256258968 |
| Gray Augmented | 0.7224248366 | 0.1070103494 |
| Color Normal | 0.8322078431 | 0.09865393818 |
| Color Augmented | 0.8257738562 | 0.07953756325 |



Honestly, I was surprised by these results. You could argue that this is a small sample size to be making predictions but to me it appears that each of our categories performed very similarly except for our Gray Augmented. This category performed much worse with an average accuracy of about 72% while the others were in the low-mid 80% range. I had figured that color would play a factor into our augmented set, but I’m now well aware that it is necessary for our Augmented dataset, otherwise we may actually be hurting our ability to detect potholes.

### 4.6 Final Model

Finally, it’s time to create our final model. As we’ve said, we unfortunately were not able to spend time developing a model architecture specifically for our augmented dataset. Instead we will be using the same model architecture as our normal dataset as for our augmented dataset, and compare our results to see if there is a difference. However our augmented dataset will use an rgb color scale as opposed to a grayscale color scale as we have seen that this greatly affects our accuracy.

To create our final model, we need to have the ability for our model to ‘store’ the model structure / node weight values as it is being trained, instead of using the model that is returned at the end of our training. To do this we can easily implement Keras’ model checkpoint and callback methods. Using these methods we are able to monitor our accuracy on our validation set as we are training, where we can save the model weights that achieve the highest validation accuracy; every time we get a higher validation accuracy, the last model weights are overwritten with these new weights. Thus we are able to save our best and most accurate model.

Lastly, we will need to be able to prove that our model is accurate no matter what data it is tested and trained on. Previously during our experiments we have only been running 3 iterations where we use the exact same training, validation and test data sets with each iteration. Now we need to prove that our model was not getting ‘lucky’ each time and that it would be able to be trained on any data that it is presented or tested on. To do this, we can implement k-fold cross validation. Basically, we iteratively ‘cut’ a section of our training data off to be used as the validation set, train and save a model, evaluate and store the results, and then start all over again with a new ‘cut’. Each time we train and save a new model will be on partially new and totally new validation sets. Doing this lets us say that we are not getting ‘lucky’, and that we have developed an architecture that is able to ‘learn’ and accurately detect potholes. We will be using 10-folds, which means that our validation set will contain 10% of our training data and that we will create 10 different final models. We can choose which is the best model based on the results of testing on our test and Nova Scotia data sets.

And that’s it! We ran our Google Colaboratory code to develop our models, tested on our original dataset as well as our images of Nova Scotia roads and got our results. We will discuss our findings within the next sections.

## 5. Results Evaluation

### 5.1 Normal Model Accuracy on Original Dataset

After running 10-fold cross validation on 150 epochs, we saved the 10 most accurate models on our validation set and then evaluated the accuracy on our test set of 151 images. The following table is our results of this evaluation, where the value represents the percentage of images accurately classified:

|  | **Normal - Original Images** |
| --- | --- |
| Iteration 1 | 0.874 |
| Iteration 2 | 0.88 |
| Iteration 3 | 0.795 |
| Iteration 4 | 0.894 |
| Iteration 5 | 0.894 |
| Iteration 6 | 0.841 |
| Iteration 7 | 0.828 |
| Iteration 8 | 0.702 |
| Iteration 9 | 0.788 |
| Iteration 10 | 0.834 |
| Average: | 0.833 |
| Standard Deviation: | 0.05967690787 |

[Link to spreadsheet](https://docs.google.com/spreadsheets/d/1TEknWGXQ26uPiLzGefmva5B49xwJSEZtHOuT7gXWae0/edit#gid=0)

Wow that’s really good! We will discuss our results within section 6.1 when we talk about all of our results.

### 

### 5.2 Normal Model Accuracy on Nova Scotia Dataset

Using the models that we created in the previous section, we then evaluated our model on our 17 Nova Scotia roadway images. This is the results of our evaluation:

|  | **Normal - NS Images** |
| --- | --- |
| Iteration 1 | 0.882 |
| Iteration 2 | 0.882 |
| Iteration 3 | 0.882 |
| Iteration 4 | 0.882 |
| Iteration 5 | 0.882 |
| Iteration 6 | 0.883 |
| Iteration 7 | 0.882 |
| Iteration 8 | 0.882 |
| Iteration 9 | 0.882 |
| Iteration 10 | 0.883 |
| Average: | 0.8822 |
| Standard Deviation: | 0.0004216370214 |

[Link to spreadsheet](https://docs.google.com/spreadsheets/d/1TEknWGXQ26uPiLzGefmva5B49xwJSEZtHOuT7gXWae0/edit#gid=0)

### 5.3 Model Accuracy on Original Dataset with Enhanced Image Augmentation

Yet again we ran 10-fold cross validation on 150 epochs to create 10 of our most accurate models. The only difference with our model structure with this is that we are using rgb color scale images as opposed to grayscale. This is how well each of our models was at accurately predicting potholes on our test set:

|  | **Augmented - Original Images** |
| --- | --- |
| Iteration 1 | 0.914 |
| Iteration 2 | 0.927 |
| Iteration 3 | 0.854 |
| Iteration 4 | 0.921 |
| Iteration 5 | 0.914 |
| Iteration 6 | 0.848 |
| Iteration 7 | 0.834 |
| Iteration 8 | 0.894 |
| Iteration 9 | 0.907 |
| Iteration 10 | 0.914 |
| Average: | 0.8927 |
| Standard Deviation: | 0.03413388671 |

[Link to spreadsheet](https://docs.google.com/spreadsheets/d/1TEknWGXQ26uPiLzGefmva5B49xwJSEZtHOuT7gXWae0/edit#gid=0)

### 5.4 Model Accuracy on Nova Scotia Dataset with Enhanced Image Augmentation

And finally, we find how well our models perform on our Nova Scotia images as well. This is the results of this evaluation:

|  | **Augmented - NS Images** |
| --- | --- |
| Iteration 1 | 0.706 |
| Iteration 2 | 0.706 |
| Iteration 3 | 0.824 |
| Iteration 4 | 0.882 |
| Iteration 5 | 0.647 |
| Iteration 6 | 0.647 |
| Iteration 7 | 0.824 |
| Iteration 8 | 0.706 |
| Iteration 9 | 0.588 |
| Iteration 10 | 0.882 |
| Average: | 0.7412 |
| Standard Deviation: | 0.104507097 |

[Link to spreadsheet](https://docs.google.com/spreadsheets/d/1TEknWGXQ26uPiLzGefmva5B49xwJSEZtHOuT7gXWae0/edit#gid=0)

## 

## 6. Conclusion

### 6.1 Overall Evaluation

#### 6.1.1 Normal Model Accuracy on Original Dataset

I am very happy to see that our final models on our original dataset performed very well and exceeded our baseline accuracy. Not only was it accurate but it’s accuracy across all of the final models was very stable with a standard deviation of only ~6.00. It is important to note that there was an outlier with a final accuracy on our test set of just above 70%, and thus there is absolutely room to improve upon our model structure.

#### 6.1.2 Normal Model Accuracy on Nova Scotia Dataset

Our pothole detection accuracy on our Nova Scotia dataset using our trained and developed models was amazing! I can’t believe it performed so well on our dataset with an average accuracy of ~88% and standard deviation of only 0.04! In all honesty, it is slightly suspicious that our accuracies were so close together with only 2 out of the 10 iterations having values different than 88.2, but I would suspect that this may be due to the fact that we are only testing on 17 images of Nova Scotia roads. I would be interested to see if it is the same images that are misclassified each time.

#### 6.1.3 Augmented Model Accuracy on Original Dataset

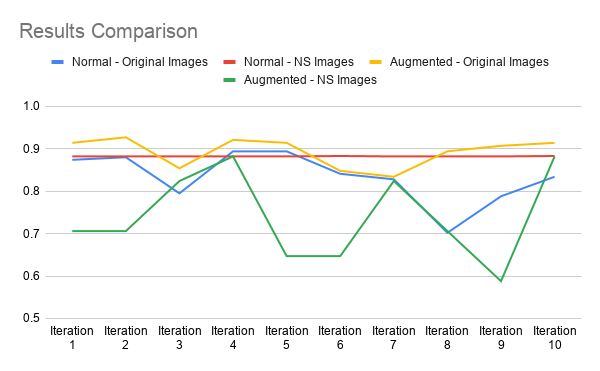
Yet again, we are very impressed by these results. With an average accuracy of almost 90%, it seems as though our model was extremely good at detecting potholes within our dataset, and even better than our non-augmented trained model. We will be able to really tell how accurate our model is within the next section when it is presented with our never-before-seen Nova Scotia images.

#### 6.1.4 Augmented Model Accuracy on Nova Scotia Dataset

Uh oh. Our accuracy as well as our standard deviation of accuracies went way off. We kind of expected this as we were not able to develop a model specific to our augmented dataset, but it is reasonable to ponder why this low accuracy occurred. This tells us that our models are either not trained enough, are generally just not able to accurately detect potholes, or what I believe could be a big problem is the fact that it is looking for features within our dataset that are not potholes such as the background color. I believe that because we introduced colors to our training set as our model is training, that would model started looking for colors more so than it did for the features of a pothole, and so all of our images of grey Nova Scotia were being misclassified as a pothole even when it was just a picture of a road.

#### 6.1.5 Results Comparisons

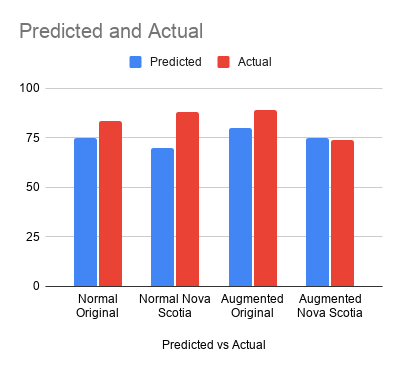
The graph below plots all of our accuracies through each iteration on each of our dataset categories. Take into consideration the fact that our vertical axis only goes from 50% accuracy to 100%.



[Link to spreadsheet](https://docs.google.com/spreadsheets/d/1TEknWGXQ26uPiLzGefmva5B49xwJSEZtHOuT7gXWae0/edit#gid=0)

The first thing we notice is that our augmented models on our Nova Scotia images are obviously the most inaccurate and most unstable. This follows suit with what we said in section 6.1.4 in that we believe that our model was not necessarily looking for pothole features and thus would definitely be unable to accurately detect potholes in newly presented images. The other thing that I would like to point out is the low 70% accuracy outlier that we saw during our Normal Original Images results evaluation. As we can see, this model on iteration 8 was still able to accurately detect potholes on our Nova Scotia dataset, so it’s interesting to see that it performed so much worse on our test set. This tells me that we definitely have room to improve our model.

Lastly, I’d like to compare our final averages of accuracies with our predicted accuracies before we started this project. The graph below compares these values.



[Link to spreadsheet](https://docs.google.com/spreadsheets/d/1TEknWGXQ26uPiLzGefmva5B49xwJSEZtHOuT7gXWae0/edit#gid=0)

Wow, we may have been to conservative in our predictions! Except for our final augmented accuracy on our Nova Scotia images, our model was much more accurately able to detect potholes than we predicted that it would. We actually predicted that on our normal model we were going to have a worse accuracy on our Nova Scotia images, when in reality we had a better accuracy! Overall, we were successfully able to develop a convolutional neural network model that is able to accurately detect potholes on an image of a roadway; whether that image is scrapped from the internet or from actual images of the road from Nova Scotia.

### 6.2 Future Work

I believe that it is absolutely possible that you could continue working on our pothole detection problem by building upon our model to make it better and trying to figure out how to implement our solution in a real world sense, but it would absolutely require a lot of work and a huge development process, especially if we wanted to use it for self-driving cars. My first step would of course be to gather better images of roads and potholes preferably from the perspective of wherever a camera may be placed in a self-driving vehicle as opposed to our scraped images. We would of course then need to build a much more accurate model structure and final model based on these images. I believe the next step after that would be to try and figure out how we hook our model up to a self driving vehicle, or even a regular vehicle for the time being to figure things out, but I am skeptical to believe that this would be the best route to take. Right off the top of my head, I can think of a lot of scenarios that we would need to deeply explore before we are able to confidently say anything about how accurate our solution is at detecting potholes in a real world sense. For example, how are the images being taken on the fly as our vehicle drives? Are we taking still images of a video? And would our model even be fast enough to keep up with a car moving at high speeds? How would our model perform at night time? What would the vehicle even do if it detects a pothole? Swerving when driving can be very dangerous so should the vehicle just slow down instead and risk damaging its own brakes? These are just some of the questions that I would want to explore before really diving into our solution of pothole detection

In reality, there is already a lot of work being done in the area of roadway condition detection. By a quick google search you can find quite a few scholarly articles on how self-driving vehicles detect potholes, how some researchers made a vehicle that can detect and save the location of potholes in the road as they drive and even detection solutions that not only detect potholes but can detect humps and even measure the dimensions of these humps/potholes [[y](https://www.researchgate.net/publication/277658928_Automatic_Detection_and_Notification_of_Potholes_and_Humps_on_Roads_to_Aid_Drivers)]. It seems as though there are solutions that use deep neural network architectures such as we did [[z](https://ieeexplore.ieee.org/document/8615819)], but that the more common method is to use Ultrasonic sensors [y]. There is even discussion from the leading self-driving car manufacturers Tesla’s CEO Elon Musk about the eventual adoption of 8 cameras around a vehicle that would be able to detect potholes / road irregularities, make and save mini-maps of these features and then be able to share these mini-maps with other Tesla car users [[t](https://electrek.co/2020/02/05/tesla-autopilot-detect-potholes-mini-maps-remember-them/)]. There is still a lot of work to do and is currently being done in the realm of self-driving vehicles and roadway feature detection, and I’m looking forward to seeing where it may lead.

## 

## 7. References

[1] Dhillon, A. (2018, July 24). 'More deadly than Terrorism': POTHOLES responsible for killing 10 people a day in India. Retrieved April 7, 2021, from <https://www.theguardian.com/world/2018/jul/25/more-deadly-than-terrorism-potholes-responsible-for-killing-10-people-a-day-in-india>

[2] Brownlee, J. (2019, July 05). How to Develop VGG, inception AND Resnet modules from scratch in Keras. Retrieved April 7, 2021, from <https://machinelearningmastery.com/how-to-implement-major-architecture-innovations-for-convolutional-neural-networks/>

[3] Madli, R., Hebbar, S., Pattar, P., & Golla, V. (2015). Automatic detection and notification of potholes and humps on roads to AID DRIVERS [Abstract]. *IEEE Sensors Journal,* *15*(8), 4313-4318. doi:10.1109/jsen.2015.2417579

[4] Anand, S., Gupta, S., Darbari, V., & Kohli, S. (2018). Crack-pot: Autonomous Road crack and pothole detection [Abstract]. *2018 Digital Image Computing: Techniques and Applications (DICTA)*. doi:10.1109/dicta.2018.8615819

[5] Lambert, F. (2020, February 05). Tesla autopilot is going to detect potholes and MAKE MINI-MAPS to remember them. Retrieved April 7, 2021, from <https://electrek.co/2020/02/05/tesla-autopilot-detect-potholes-mini-maps-remember-them/>