Ryerson University

Department of Aerospace Engineering
Faculty of Engineering & Architectural Science

Semester, Year:

Semester, Year:	☑ Fall ☐ Winter ☐ Spring/Summer 2023
Course Code:	AER 850
Course Title:	Introduction to Machine Learning
Section Number:	01
Instructor:	Reza Faieghi
Submission:	☐ Assignment ☐ Lab Report ☑ Project Report ☐ Thesis ☐ Other: _____
Due Date:	November 26, 2023

Project 1 Report

November 26, 2023

Authors/Contributors:	Student Number (XXXX12345):	Signature*:
Kotasthane, Gaurang	500922614	G.K.

**By signing the above you attest that you have contributed to this submission and confirm that all work you contributed to this submission is your own work. Any suspicion of copying or plagiarism in this work will result in an investigation of Academic Misconduct and may result in a "0" on the work, and "F" in the course, or possibly more severe penalties, as well as a Disciplinary Notice on your academic record under the Student Code of Academic Conduct, which can be found online at www.ryerson.ca/academicintegrity/students/ryersons-academic-integrity-policy-policy-60/*

Table of Contents

Discussion

Data Processing

1. The image shape was defined to be (100, 100, 3)

```
1. image_shape = (100, 100, 3)
```

2. The data was already split so this step was already completed.
3. In this lab, two methods to import datasets were implemented. After researching online for methods to import datasets, two methods were found.
    a. *Imagedatasetfromdirectory*
        i. The data augmentation options provided by this library include *RandomFlip, RandomRotation, RandomZoom, Rescaling* and some others. It is important to note the data augmentation options as this severely affects the training accuracy and validation accuracy. This is explored in the latter half of the report.

    b. *ImageDataGenerator*
        i. The *ImageDataGenerator* class has been the standard method to import datasets and for data augmentation and has a lot more options. One of the requirements for the project is to add shear in the data augmentation process which is only available if this specific class is used. Again, it is important to note this since using certain data augmentation techniques over others seemed to have affected the training accuracy versus validation accuracy.
    c. Since there were two different data augmentation pipelines explored in this project, different methods were used.
        i. *Imagedatasetfromdirectory:*
            1. *RandomFlip*
            2. *RandomZoom*
            3. *RandomRotation*
        ii. *ImageDataGenerator*
            1. *Rescaling*
            2. *Shear*
            3. *Zoom*
    d.
        i. Step 4 says to create train and validation generator using *imagedatasetfromdirectory,* but data augmentation using this method doesn't need generators, instead the data augmentation is defined in the model as layers of the model.

ii. I created training, validation and test generators using *ImageDataGenerator* class for my second approach to data augmentation.

Neural Network Architecture Design

The model used to solve the machine learning problem is listed below but the following methodology was used.

1. Start with 2 Convolutional layers and 2 MaxPooling Layers to see how the model performs.
2. **Kernel Size and Strides:** To begin with Kernel size values, inspiration was taken from Alex Net that uses Kernel size of 11x11 and strides 4x4, after experimenting with different values for Kernel size and strides, following observations were made.
    a. Higher kernel size value for the initial CNN layers increase accuracy but with lower kernel size worked better on the latter CNN layers.
    b. Strides of 2x2, 3x3, 4x4 worked to give the maximum accuracy. Anything beyond this usually resulted in lower accuracy or didn't perform any better.
3. **BatchNormalization:** Batch normalization was implemented after every convolution layer, since this helps with performance and accuracy.

Hyperparameter analysis

1. **Number of layers and number of neurons:** The ideology followed to optimize these numbers is as follows.
    a. Make the model more complex until the training accuracy reaches a satisfying level of performance and make. So, if the training accuracy is reaching, let's say 70%, the model can be made more complex, by either increasing the number of neurons or increasing the number of layers.
    b. If the model has a satisfying level of training accuracy but lower validation accuracy, this could be a case of overfitting, in this case my approach was to reduce the number of neurons.
    c. In terms of *MaxPooling2D* and *Dense* there wasn't much to optimize, except the number of dense layers and the number of neurons. The following are the observations.
        i. Increasing the number of Dense layers: Severely affected training and therefore validation accuracy
        ii. Increase the number of neurons, helped with accuracy until a certain point (4096 neurons) and then there was no improvement in performance. In some cases, over fitting was observed and the validation accuracy couldn't match the training accuracy.

2. *'relu'* and *'leaky relu'* were both implemented and tested and '*relu'* outperformed in terms of accuracy and hence this was implemented as the activation function for the CNN layers.
3. In terms of loss function and optimizer, *categorical crossentropy* and *adam* were implemented.
4. Another important parameter that was optimized for better accuracy was the learning rate. A learning rate of 0.01 outperformed 0.1 in terms of accuracy and for computational efficiency it was chosen over 0.001.
5. To improve the performance of the CNN model, A kernel regularizer was also implemented which helps in avoiding overfitting by penalizing the assignment of higher value weights to any layer. A kernel regularizer (L2) with 0.01 was chosen.

Model Summary

```
1.
2. def build_model(image_shape):
3.     model = models.Sequential([
4.         # Convolutional and Pooling Layers
5.         layers.Conv2D(64, (11, 11), strides=(4, 4), activation='relu', input_shape=image_shape,
padding='same'),
6.         layers.BatchNormalization(),
7.         layers.MaxPooling2D((3, 3), strides=(2, 2)),
8.         layers.Conv2D(128, (5, 5), activation='relu', padding='same'),
9.         layers.BatchNormalization(),
10.         layers.MaxPooling2D((3, 3), strides=(2, 2)),
11.         layers.Conv2D(64, (3, 3), activation='relu', padding='same'),
12.         layers.BatchNormalization(),
13.         layers.MaxPooling2D((3, 3), strides=(2, 2)),
14.
15.         # Flattening the 3D output to 1D
16.         layers.Flatten(),
17.
18.         # Dense Layers with L2 Regularization
19.         layers.Dense(2048, activation='relu', kernel_regularizer=l2(0.01)),
20.         layers.Dropout(0.5),
21.
22.         # Output Layer
23.         layers.Dense(4, activation='softmax')
24.     ])
25.
26.     return model
27.
```

Summary:

```
In [6]: runfile('G:/My Drive/Gaurang Files/TMU/Year 4/AER 850 Intro
Found 1600 images belonging to 4 classes.
Found 800 images belonging to 4 classes.
Found 192 images belonging to 4 classes.
Model: "sequential_5"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_15 (Conv2D)          (None, 25, 25, 64)        23296

 batch_normalization_15 (Bat  (None, 25, 25, 64)       256
 chNormalization)

 max_pooling2d_15 (MaxPoolin  (None, 12, 12, 64)       0
 g2D)

 conv2d_16 (Conv2D)          (None, 12, 12, 128)       204928

 batch_normalization_16 (Bat  (None, 12, 12, 128)      512
 chNormalization)

 max_pooling2d_16 (MaxPoolin  (None, 5, 5, 128)        0
 g2D)

 conv2d_17 (Conv2D)          (None, 5, 5, 64)          73792

 batch_normalization_17 (Bat  (None, 5, 5, 64)         256
 chNormalization)

 max_pooling2d_17 (MaxPoolin  (None, 2, 2, 64)         0
 g2D)

 flatten_5 (Flatten)         (None, 256)               0

 dense_15 (Dense)            (None, 2048)              526336

 dropout_10 (Dropout)        (None, 2048)              0

 dense_16 (Dense)            (None, 4)                 8196

=================================================================
Total params: 837,572
Trainable params: 837,060
Non-trainable params: 512
```

Figure 1: Model Summary and trainable parameters

Model Evaluation

1. Epochs = 100

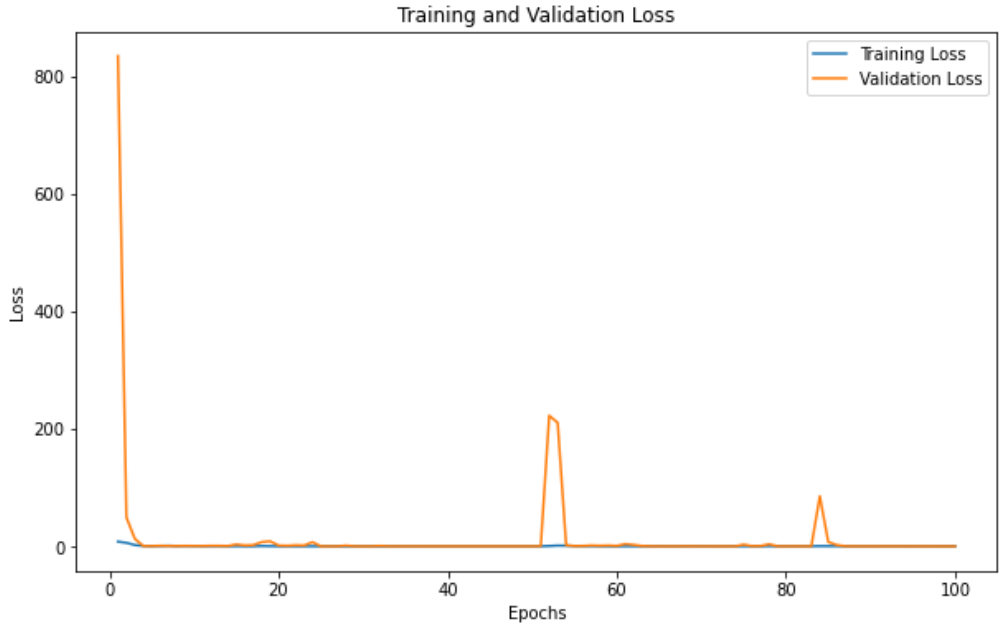

Figure 2: Training/Validation Loss (Epoch 100)

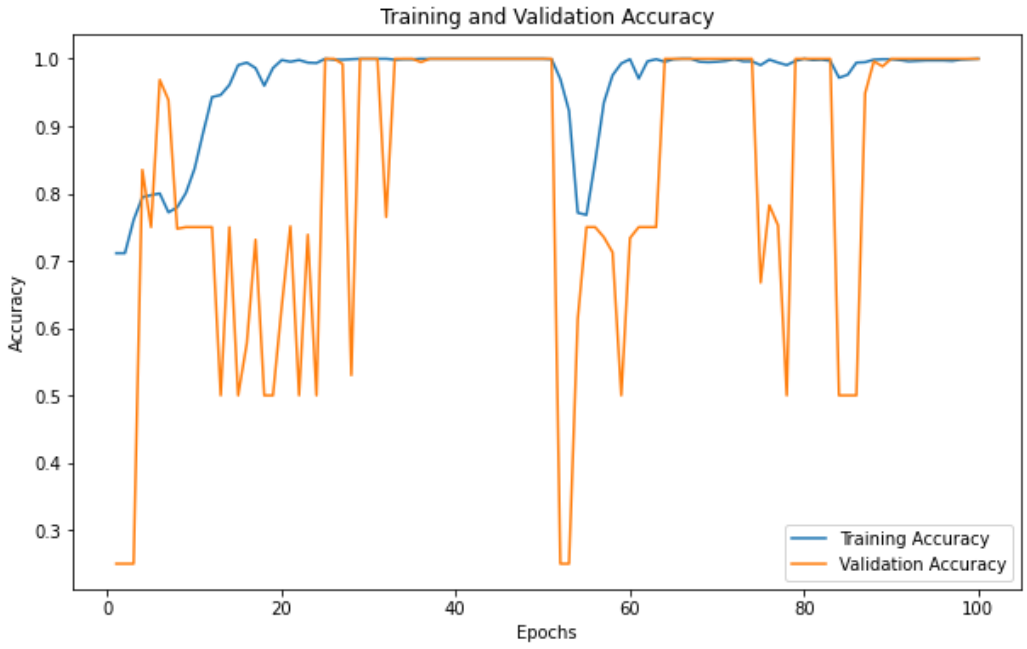

Figure 3: Training/Validation Accuracy (Epoch 100)

```
Epochs = 32
```

Figure 4: Training/Validation Loss (Epoch 32)

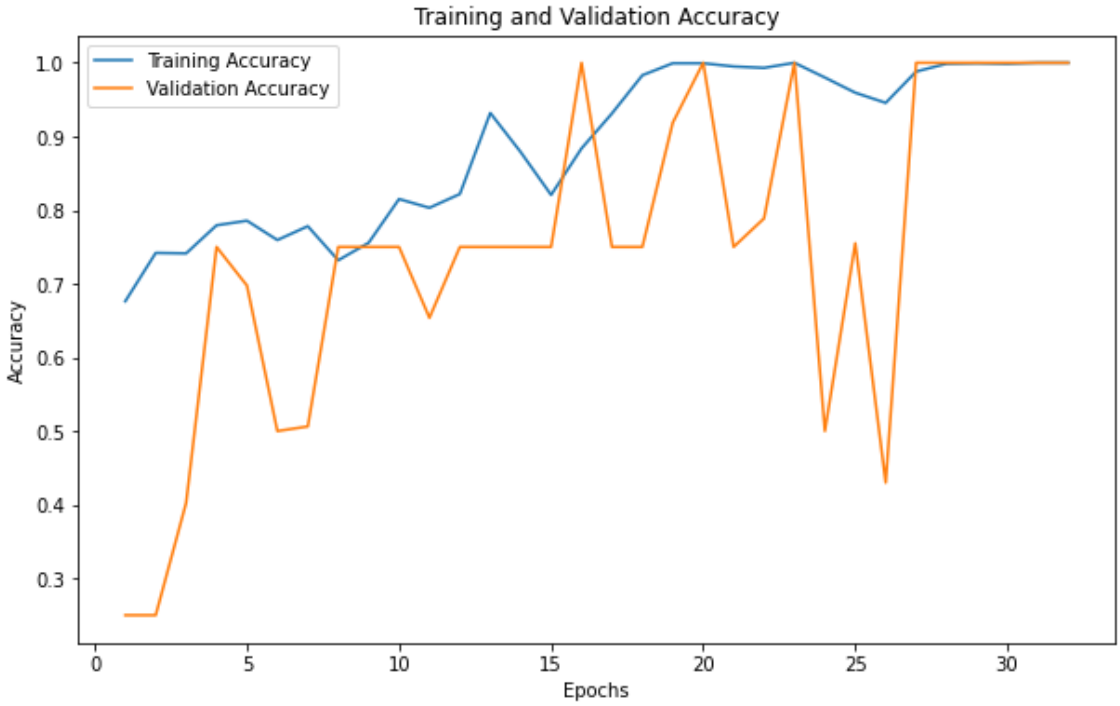

Figure 5: Training/Validation Accuracy (Epoch 32)

1. Epoch =5

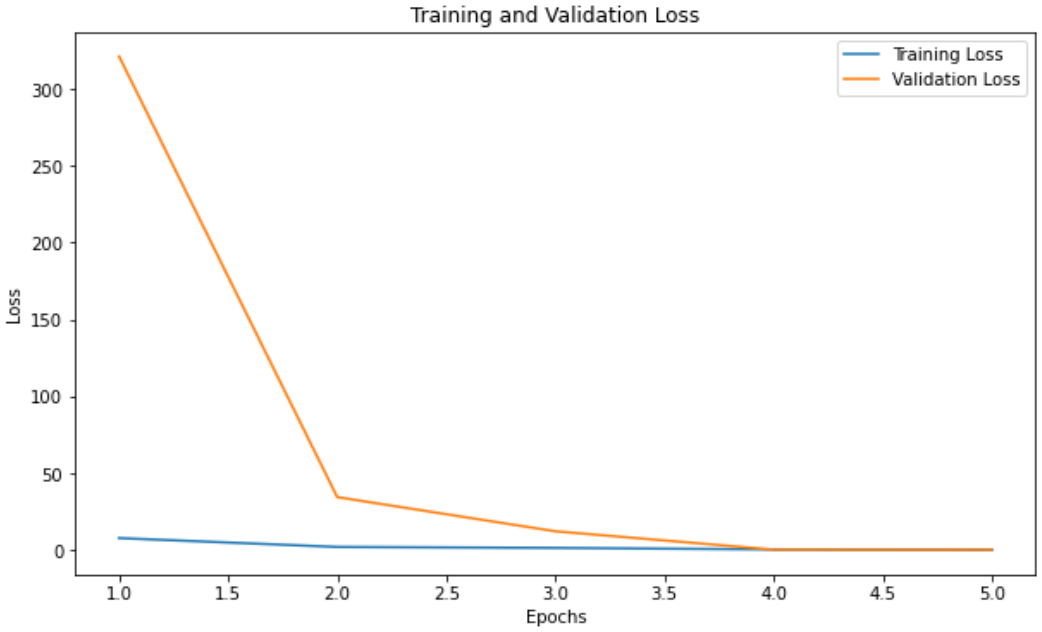

Figure 6: Training/Validation Loss (Epoch 32)

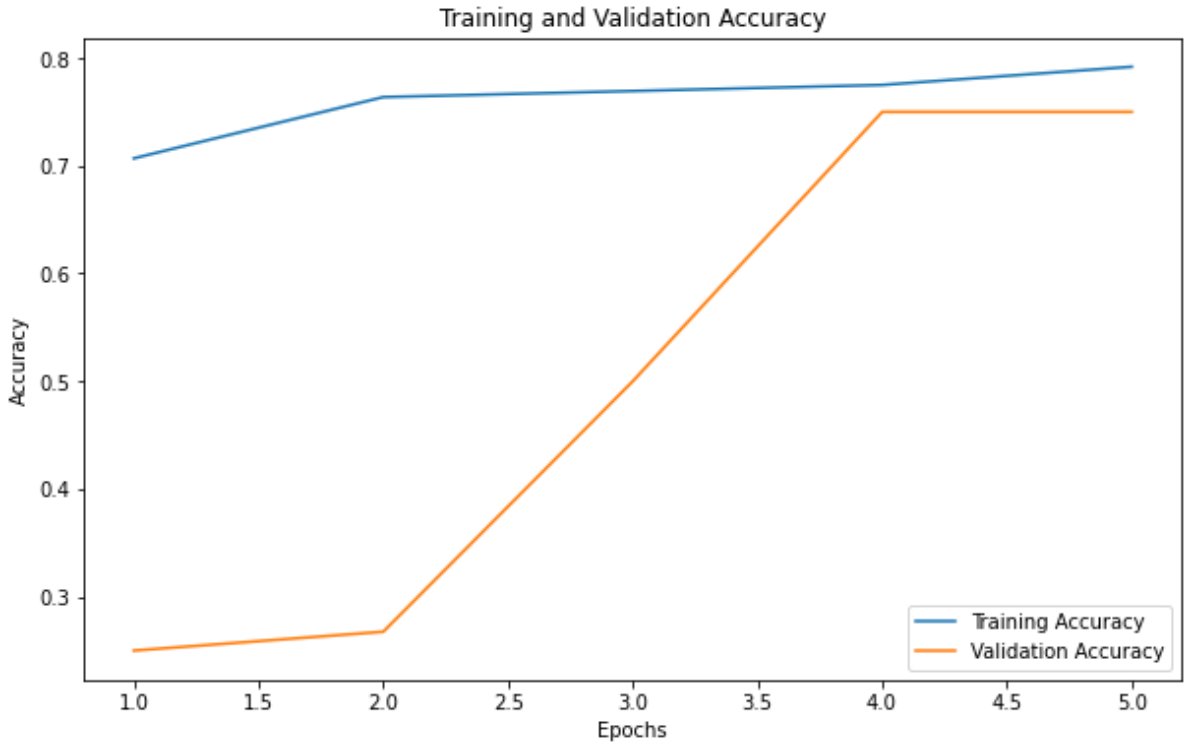

Figure 7: Training/Validation Accuracy (Epoch 32)

Model Evaluation – Discussion

There are a few things to discuss here:

1. Epochs: Number of epochs help until a certain number and after that it has no effect on the accuracy. For example, extremely low number of epochs (epochs=5) are too low to achieve higher accuracy. But beyond 10-15, the accuracy isn't affected by number of epochs (as evident from the figures attached above)

2. Data Augmentation: The methods used to augment data heavily impacted training and validation accuracy. Especially given the nature of dataset. For example, using RandomFlip, RandomRotation avoided the model from achieving a high validation accuracy. Looking at the current accuracy graph for training and validation, the spikes are because of the data augmentation methods used. Data augmentation is only done on the training set; hence it is impossible for the validation set to achieve high accuracy on those images. Consider the following images.

Figure 8: Large Crack

This is a large crack image from the train dataset, it is easy to notice the crack (highlighted in red). But now let's consider the small and medium cracks.

Figure 9: Medium Crack

Figure 10: Small Crack

These are medium and small images respectively. Now if I were to apply data augmentation methods to these methods and increase or reduce the brightness of these images, the actual crack is almost invisible to the human eye (specifically if I reduce the brightness) and so it is very hard to understand if the model is learning to recognize crack by looking the actual crack or the big hole in the centre. Hence Data augmentation methods are extremely cruicial in determining how the model will perform (on top of other machine learning optimization techniques). There's also the chance that it's not distinguish well between small cracks and no cracks.

Hence achieving 100 validation accuracy without data augmentation is a relatively easier process but with data augmentation it was considerably harder (this was discussed with the professor as well).

3. In terms of model performance it would be helpful to establish a baseline accuracy, since there are some of the small cracks that are invisible to human eye and it hard to tell what the model is exactly learning on to dififerentiate between small cracks and no cracks, especially wiith brightness related data augmentation

Model Testing

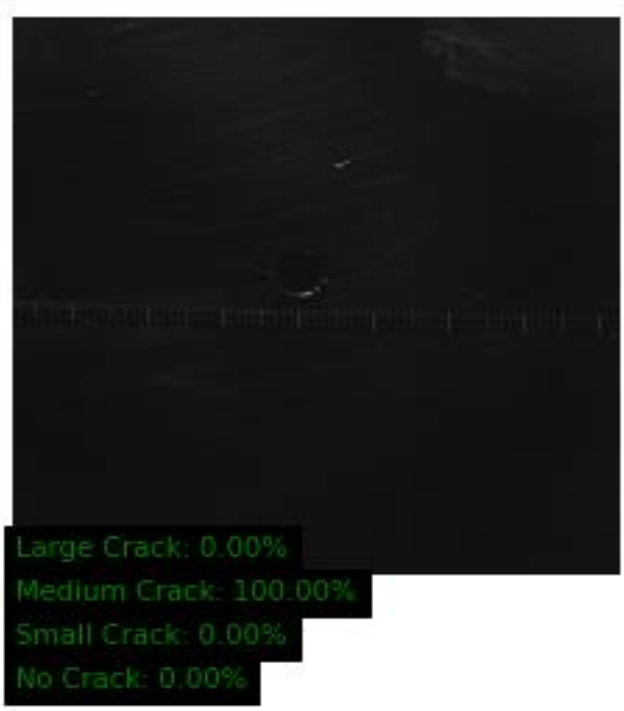

Figure: Prediction for *Crack__20180419_06_19_09, 915.bmp (Truth: Medium)*

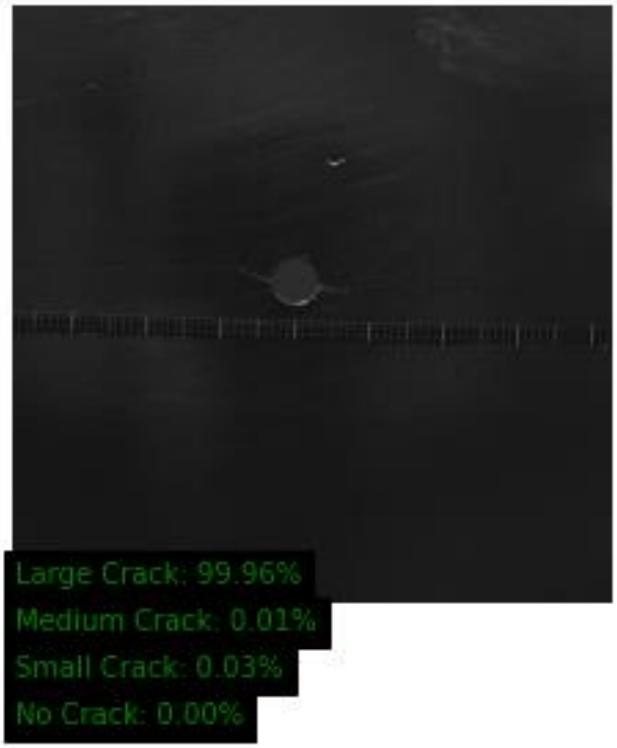

Figure: Prediction for *Crack__20180419_13_29_14, 846.bmp (Truth: Large)*