

**Honours Project - MHW225671**

**FINAL REPORT**

**2020-2021**

**Department of Computing**

**Submitted for the Degree of:**

**BSc Computing**

**Project Title: An analysis of cracks in concrete structures using a neural network with an investigation into activation functions.**

**Name: Ben Cody Maxwell**

**Programme: BSc Computing**

**Matriculation Number: S1917932**

**Project Supervisor: Dr Mark Jenkins**

**Second Marker: Richard Holden**

**Word Count: 10,628 Words**(***excluding contents pages, figures, tables, references and Appendices***)

**“Except where explicitly stated, all work in this report, is my own original work and has not been submitted elsewhere in fulfilment of the requirement of this or any other award”**

**Signed by Student: Ben Maxwell Date: 06/10/2020**

# Abstract

The issue of aging concrete structures going into disrepair due to poor maintenance and defects is a major issue in the modern world. Visual inspection of these structures is costly and requires a large amount of time to examine the entire structure requiring skilled inspectors during the entire process, often smaller defects are left unnoticed and gradually become a risk over time. Letting these structures fall into disrepair can lead to high maintenance costs to repair damage and could lead to accidents occurring if the structure is left in disrepair for a long amount of time, threatening the safety of people around the structure.

The aim of this project is to produce a neural network that can use image segmentation to detect cracks with an investigation into the performance differences that different activation functions will affect within the network.

The development and testing stages of this project produced a neural network that can identify defects within images of concrete with an investigation into the different activation functions, providing results that can be drawn upon in the evaluation to prove that it is possible to use a neural network and image segmentation to find cracks within concrete structures.

Contents

[Abstract 2](#_Toc69305383)

[Table of Figures 5](#_Toc69305384)

[Table of Tables 5](#_Toc69305385)

[1. Introduction 6](#_Toc69305386)

[1.1 Project Background 6](#_Toc69305387)

[1.2 Project Overview 8](#_Toc69305388)

[1.2.1 Project Outline 8](#_Toc69305389)

[1.2.2 Project Aims and Objectives 8](#_Toc69305390)

[1.3 Hypothesis 10](#_Toc69305391)

[2. Literature Review 11](#_Toc69305392)

[2.1 Investigation into existing solutions in Industry 11](#_Toc69305393)

[2.1.1 Mask R-CNN 11](#_Toc69305394)

[2.1.2 U-Net Architecture 12](#_Toc69305395)

[2.1.3 Activation Functions 13](#_Toc69305396)

[2.2 Researching the development of Neural Networks 14](#_Toc69305397)

[2.3 Appropriate Libraries 15](#_Toc69305398)

[2.3.1 NumPy 15](#_Toc69305399)

[2.3.2 Scikit-Image 16](#_Toc69305400)

[2.4 Researching common structural defects in concrete 16](#_Toc69305401)

[3. Methods 18](#_Toc69305402)

[3.1 Research Method 18](#_Toc69305403)

[3.1.2 Establish a test plan for the neural network 18](#_Toc69305404)

[3.1.3 Stage 1 Prototype: Gather data from pre-existing datasets 18](#_Toc69305405)

[3.1.4 Stage 2 Prototype: Initial neural network created using TensorFlow 19](#_Toc69305406)

[3.1.5 Stage 3 Prototype: Neural network requires testing before full training 19](#_Toc69305407)

[3.1.6 Stage 4 Prototype: Fully training the neural network 19](#_Toc69305408)

[3.1.7 Conduct testing 19](#_Toc69305409)

[3.1.8 Analyse test results and draw a conclusion 19](#_Toc69305410)

[3.2 Data Preparation 19](#_Toc69305411)

[3.2.1 Mask Conversion 20](#_Toc69305412)

[3.2.2 Standardisation 21](#_Toc69305413)

[3.2.3 Image Augmentation 21](#_Toc69305414)

[4. Implementation 23](#_Toc69305415)

[4.1 Neural Network Foundations 23](#_Toc69305416)

[4.2 Dataset Conversion 23](#_Toc69305417)

[4.3 Data Adjustment & NPY Creation 25](#_Toc69305418)

[4.4 Neural Network Models 27](#_Toc69305419)

[4.5 Custom Activation Functions 30](#_Toc69305420)

[5. Testing 33](#_Toc69305421)

[6. Evaluation 37](#_Toc69305426)

[6.1 Activation Functions Evaluation 37](#_Toc69305427)

[6.2 Prediction Results Evaluation 38](#_Toc69305428)

[7. Conclusion 40](#_Toc69305429)

[7.1 Project Resume 40](#_Toc69305430)

[7.2 Results Discussion 40](#_Toc69305431)

[Activation Functions Conclusion 40](#_Toc69305432)

[Prediction Results Conclusion 41](#_Toc69305433)

[Overall Conclusion 41](#_Toc69305434)

[7.3 Limitations and Future Improvements 41](#_Toc69305435)

[Limitations 41](#_Toc69305436)

[Future Improvements 41](#_Toc69305437)

[8. References 42](#_Toc69305438)

[9. Appendices 46](#_Toc69305439)

[Appendix A – Code Listings 46](#_Toc69305440)

[GitHub Repository 46](#_Toc69305441)

[Full Project Upload with Prediction Results & NPY files 46](#_Toc69305442)

[Appendix B – Main.py of Neural Network 46](#_Toc69305443)

[Appendix C – Data.py of Neural Network 47](#_Toc69305444)

[Appendix D – Activation Function Logs 49](#_Toc69305445)

# Table of Figures

[Figure 1 - Demonstration of Image segmentation mask 7](#_Toc69019593)

[Figure 2 - Segmentation Comparison 11](#_Toc69019594)

[Figure 3 - An example of misalignment producing inaccurate mask but accuracy results remain high 12](#_Toc69019595)

[Figure 4 - U-Net Architecture. Each blue box represents a multi-channel feature map. The number of channels is shown at the top of the box and the white boxes represent copied feature maps. The arrows denote the different operations as seen in the legend. 13](#_Toc69019596)

[Figure 5 - ReLu Activation Function 14](#_Toc69019597)

[Figure 6 - Swish Activation Function 14](#_Toc69019598)

[Figure 7 - Example of Newly Converted Mask 21](#_Toc69019599)

[Figure 8 - Conversion of JPG to PNG 24](#_Toc69019600)

[Figure 9 - MAT Conversion 25](#_Toc69019601)

[Figure 10 - npyCreation.py image processing methods 26](#_Toc69019602)

[Figure 11 - adjustData method 27](#_Toc69019603)

[Figure 12 - U-Net Code Example 28](#_Toc69019604)

[Figure 13 - Simple\_U-Net example 29](#_Toc69019605)

[Figure 14 - Final U-Net iteration 30](#_Toc69019606)

[Figure 15 - LeakyReLu Model 31](#_Toc69019607)

[Figure 16 - Swish Implementation 32](#_Toc69019608)

[Figure 17 - Network Image Result 1 38](#_Toc69019609)

[Figure 18 - Network Image Results 2 38](#_Toc69019610)

[Figure 19 - Network Image Results 3 38](#_Toc69019611)

[Figure 20 - Network Image Results 4 39](#_Toc69019612)

# Table of Tables

[Table 1 - Activation Functions Testing 1 34](#_Toc68988955)

[Table 2 - Activation Functions Testing 2 34](#_Toc68988956)

[Table 3 - Activation Functions Testing 3 35](#_Toc68988957)

[Table 4 - Activation Functions Testing 4 35](#_Toc68988958)

[Table 5 - Activation Functions Testing 5 36](#_Toc68988959)

# 1. Introduction

This section contains a short background of both neural networks and machine learning and the defects that plague modern concrete structures such as railway tunnels, buildings and roads to highlight to the reader the importance of maintenance and the advantages that this project can provide to the problem area. Further into this section it will inform the reader why machine learning is becoming more commonly used in industry and how it can provide an advantage to systems that use it. Also included in this section will be a project overview outlining the objectives that this project aims to achieve concluding in a hypothesis.

## 1.1 Project Background

The biggest issue that structures such as bridges, dams and railway tunnels face is the degradation of the building material over their lifetime, this is when defects form. Cracks are the most common and important diseases of concrete bridges (Liang D. et al, 2019), cracks are a common issue with most structures, forming both horizontally and vertically which develop due to repeated use and/or age of the material, these cracks can present risk to the structural integrity (Konig, J. et al. 2019).

These types of defects are prominent in not just concrete infrastructure but also structures such as wind turbines, according to a survey of wind turbines in Sweden, structural damage to rotor blades contribute to 13.4% of failures (Reddy, A. 2019). An analysis of collapsed bridges revealed that 46% of these bridges had existing structural defects before their collapse (Dung C.V. 2019).

It’s also important to consider that some cracks that form don’t present a risk to the structure, some of these defects might just be surface level cosmetic damage or in the instance of older railway tunnels, they might have been designed to form an archway using brickwork and due to either poor workmanship or natural settlement, cracks could form early on in the structures lifetime. In particular with modern concrete railway tunnels, forms are created using timber which then has concrete poured over it to provide the shape, after the concrete has cured form marks that resemble defects might appear, while these marks might resemble cracks they don’t present a structural risk to the structure.

Projects such as this one are important to the safety of infrastructures, infrastructure with serious issues such as cracks can cause significant economic losses and poses a threat not only industry but also to civilian security (Feng et al, 2019). With the maintenance of road systems resulting in costs of an estimated $25 billion every year (M-Mahdi Naddaf-Sh. 2019), the importance of tools that can accurately identify damage in materials such as concrete is high as having an effective monitoring system in place could reduce maintenance costs as any damage that is caught earlier in its development can be repaired at a cheaper cost while also providing the benefit of reducing the risk the defect poses to the structure.

To help both reduce the amount of maintenance required and protect maintenance workers from the hazardous conditions of roads, railways etc (Zakeri H. 2017). The use of a neural network that can recognize these defects in structures is vital.

The use of neural networks in industry has become a large area of investment and increased use over the last years evolving from biological based architectures to more advanced computational models which allows for faster approaches to pattern and image recognition (Schumann J. 2010).

In particular, convolutional neural networks being used for image segmentation have shown excellent results, some of these networks have managed to achieve an incredibly high accuracy rate of 99.71% at correctly identifying the defect in the image (Chen K. et al, 2019).

A convolutional neural network is made up of several layers that can range from low level functions that can define changes in the image such as curves to higher level functions that can detect more challenging aspects of an image by using the multiple layers (Ma, J. et al, 2017). Unlike traditional methods, convolutional neural networks don’t need to convert the format of the image they are trying to segment and instead can automatically differentiate the features of the defect reducing the overall workload therefore increasing performance and accuracy compared to traditional neural networks (Li, S. & Zhao, X. 2019).

Image segmentation is the process of assigning each pixel in an image with predefined class labels, by using this process you can split the output into two classes and create a mask of the defect to allow for further analysis (Jenkins, MD. 2018). By creating the image mask, the neural network can focus on the area that has been identified as the defect. The figure below demonstrates an instance of how a mask can be visually displayed (Jenkins, MD. 2017).

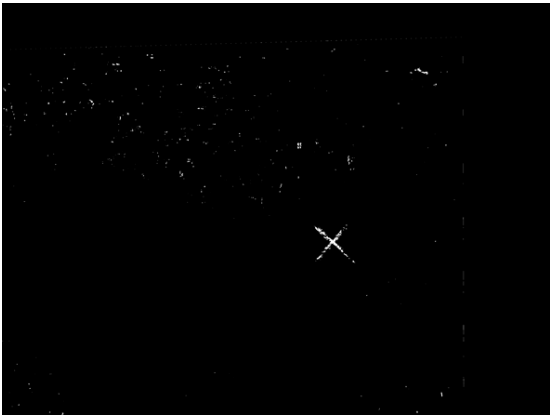


Figure - Demonstration of Image segmentation mask

While the imaging conditions are of a slightly different nature compared to this project use, it demonstrates the change that defects can cause which is displayed using the white pixels in the image with unchanged areas being displayed as black pixels (Figure 1). The use of image segmentation with convolutional neural networks has become increasingly more common in industry although current systems have issues with certain material types and environments such as current image detection methods used to detect cracks in nuclear reactor plants, the issue was the cracks that formed were very small and the presence of image noise on the surface (Zhang, L. 2019).

To combat this issue, the use of new algorithms to allow for greater accuracy when identifying defects is vital. Convolutional neural networks for image segmentation that have new architectures designed for their specific purpose such as analysing MRI images have seen a remarkable increase in accuracy compared to traditional methods (Ding, Y. 2020).

With the threat of the loss of structural integrity of structures that have been in use past their expected life span such as the majority of bridges and tunnels in use around the world, it is vital that a tool is produced to monitor defects that can arise in these structures to prevent the likelihood of incidents occurring and reduce maintenance costs.

## 1.2 Project Overview

This section will try to provide the reader with a clear explanation of the project outline and the aims and objectives of the project.

### 1.2.1 Project Outline

There is a steady reliance on the use of concrete in everyday life with most buildings and structures such as railway tunnels, roads etc., being built from concrete. With this reliance there is also an issue of maintenance, to keep these structures safe and usable constant maintenance is required. The increasing use of machine learning for different areas of industry and the developments made in machine learning frameworks providing higher levels of performance in both accuracy and speed in multiple applications of machine learning.

The aforementioned reasons in the statement above give validity to the creation of a neural network that can handle the analysis of defects in concrete structures which could spot issues earlier than inspection teams and provide an increased level of safety for these structures. In respect to these reasons, the research question for this project is:

**Can a neural network be used to identify and analyse cracks in concrete structures to therefore increase the level of safety and prevent degradation and do different activation functions effect the performance of that neural network?**

### 1.2.2 Project Aims and Objectives

The aim of this project is to create a neural network that can be used to analyse images of concrete structures to identify defects such as cracks while researching the effect on performance that different activation functions provide.

To identify whether the project is a success and to ensure that the project is a success, objectives have to be met, two types of objectives consisting of Secondary Objectives that will be met during the research phase of the project and Primary Objectives which are achieved during the development phase of the project.

**Secondary Objectives**

* **Investigate existing solutions in industry**

Research existing applications of neural networks that are currently being used in a similar industrial setting. This research will provide valuable information such as the frameworks being used, efficient algorithms for image segmentation and how the system can be implemented practically. This also provides information about why using a neural network for this project is both viable and advantageous.

* **Research development of neural networks**

Investigating the current methodology used when developing neural networks with a specific focus on how that will apply to the project’s problem area is very beneficial to the project as it will improve an understanding of how the project functions and helps to provide knowledge of what logic to follow when developing the project.

* **Research common structural defects**

Researching the common structural defects that concrete structures face throughout their lifetime will provide valuable information on which defects pose major risks and minor risks. This allows proper categorization of any data and helps to identify functional and non-functional requirements.

**Primary Objectives**

* **Establish a test plan for the neural network**

To properly evaluate the results and analyse the performance of the neural network, a test plan will be created with the images and labels from the chosen dataset to determine the effectiveness of the neural network.

* **Stage 1 Prototype: Gather data from pre-existing datasets**

In order to train the neural network, a dataset must be acquired. For this project, the CrackForest dataset provides a lot of useful and appropriate data pertaining to the study of cracks in roads and other concrete structures.

* **Stage 2 Prototype: Initial neural network created using TensorFlow**

Using the information gained from the secondary objectives and the research from the literature review, the neural network will be developed using the TensorFlow framework in Python then the next stage will be training the network.

* **Stage 3 Prototype: Neural network requires testing before full training**

To ensure that during final training the neural network will be trained without mistakes and using the most efficient activation function, multiple tests will be performed. Once testing has been carried out to a sufficient level, the neural network will move on to the next stage and be fully trained.

* **Stage 4 Prototype: Fully training the neural network**

In the last stage of development, the neural network will be running through multiple epochs to maximise the level of accuracy attainable for the neural network. The best model of all the epochs will be saved as a checkpoint.

* **Conduct testing**

Testing will be conducted on the best model of the neural network at the end of development to evaluate performance using the test plan created at the start of the primary objectives.

* **Analyse test results and draw a conclusion**

Taking the results from testing and properly analysing the outcomes of each test will provide information about the accuracy, loss and effectiveness of the neural network, this information will then be used to draw a conclusion to the effectiveness and viability of this project.

## 1.3 Hypothesis

To draw a hypothesis for this project, conditions must be met. Multiple activation functions must be tested properly to evaluate each function’s effect on performance and due to the nature of the data, it must be properly handled and converted to the correct file format.

If these conditions are met then it can be assumed that the neural network using the most efficient activation function for the network’s purpose of image segmentation will provide an efficient network that is able to detect cracks in concrete structures with a high degree of accuracy and minimal loss.

This hypothesis can be properly tested during the final stage of development of the neural network and the results of the testing in the final stage of development will provide an evaluation of both the neural network and whether the hypothesis was correct.

# 2. Literature Review

This section will focus on gathering the information required to properly develop the neural network by researching past research papers and projects by other researchers working in the field of image segmentation. The goal of the literature review is to achieve the secondary objectives mentioned in the project aims and objectives section.

These objectives are:

* Investigate existing solutions in industry
* Research the development of neural networks
* Research common structural defects in concrete

# 2.1 Investigation into existing solutions in Industry

The image segmentation of cracks is both stressful on time and computation to perform which is why multiple solutions are being explored to discover the most efficient network for this task (T. A. Carr et al, 2018). To find the most efficient architecture and beneficial activation functions for image segmentation tasks, research on two popular architectures and activation functions suited to the task of image segmentation has been carried out with a further investigation into their specific applications within image segmentation and their efficiency.

## 2.1.1 Mask R-CNN

Mask R-CNN is a possible alternative to the U-Net architecture as it was specifically designed for use as an instance segmentation network. Unlike U-Net which utilizes semantic segmentation, Mask R-CNN utilizes instance segmentation which attempts to detect multiple instances of a class within an image and differentiates them in the output mask whereas semantic segmentation finds all the instances of a class and creates an output mask showing the areas with the same class with no differentiation (H, Kaiming, 2017). To explain this clearly a diagram has been provided in the Figure 2 – Segmentation Comparison.

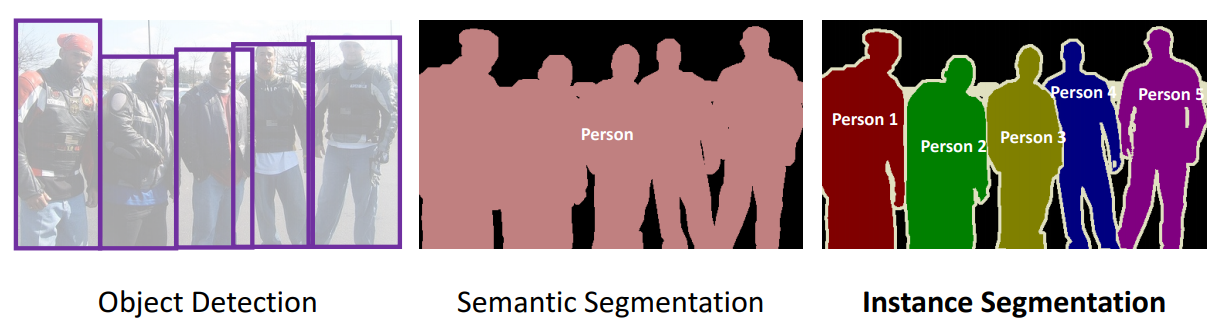


Figure - Segmentation Comparison

As shown in Figure 5, the network detects the objects that belong to the person class and then in the semantic segmentation example outputs the mask as one instance of the class Person and in the instance segmentation example, the output mask is shown as multiple instances of the Person class, differentiating between each instance as 1,2,3 etc.

Mask R-CNN works by using two outputs for each object, a class label and a bounding-box offset adding an additional branch in parallel to output a binary mask, the binary mask provides the object’s spatial layout allowing for refinement of the output mask using boundary boxes (H, Kaiming, 2017).

However, there are some issues with instance segmentation networks that utilise the same technologies of Mask R-CNN, there can be a misalignment between the classification confidence and the final mask output quality. The confidence may be high due to accurate box-level results and a high classification store but there may be a misalignment, producing an inaccurate mask, this can be seen in Figure 6 (Z. Huang et al, 2019).



Figure - An example of misalignment producing inaccurate mask but accuracy results remain high

## 2.1.2 U-Net Architecture

When researching existing solutions, the U-Net architecture appeared in multiple research papers. The original purpose of the U-Net architecture was intended for was biomedical image segmentation which commonly has very small amounts of data available to train a network similar to the issue of crack segmentation has with a small amount of data available therefore it was designed to provide accurate segmentation and to train using smaller amounts of training data (O. Ronneberger et al, 2015).

To supplement the lack of training data, data augmentation is performed adding deformations to the training data which allows the network to learn which properties stay unchanged throughout the augmentation process which is useful for evaluating areas such as cracks and in the case of biomedical imaging discerning tissue and realistic deformations such as tumours or muscles (O. Ronneberger et al, 2015).

The following diagram in Figure 2 represents the inner workings of the U-Net architecture, the network is improved by adding successive layers, replacing pooling operators with upsampling operators therefore increasing the output resolution which allows further layers to be able to create accurate output based on the higher quality information (O. Ronneberger et al, 2015).

Chart, box and whisker chart

Description automatically generated

Figure - U-Net Architecture. Each blue box represents a multi-channel feature map. The number of channels is shown at the top of the box and the white boxes represent copied feature maps. The arrows denote the different operations as seen in the legend.

The last layer in the U-Net architecture depicted in Figure 2 applies a Sigmoid activation function to split the output into a range of [0,1] depending on the values passed from previous layers.

## 2.1.3 Activation Functions

Layers within the network architecture for U-Net are using the ReLu activation function which is very effective for the task of image segmentation as gradients are able to flow when the input is positive for a ReLu function and due to its simple nature and efficiency along with its ability to be more easily optimised compared to Sigmoid or Tanh has led to its widespread use within machine learning development (P. Ramachandran, 2017).

Development in the last few years however has proposed a possibly more efficient activation function for the task of image classification/segmentation called Swish. Swish can be mathematically defined as x · σ(βx), where β represents either a constant parameter or a trainable parameter and σ(z) = (1 + exp(−z))−1 is the sigmoid function (P. Ramachandran, 2017).

|  |  |
| --- | --- |
| A Gentle Introduction to the Rectified Linear Unit (ReLU)  Figure - ReLu Activation Function | Deep Learning: The Swish Activation Function  Figure - Swish Activation Function |

As seen in Figure 5, the ReLu function for inputs of lower than 0 allows the neuron to be deactivated which increases efficiency as not all neurons are active simultaneously and any input above 0 will be activated. Due to the linearity of the ReLu function there is little risk of missing gradients allow the gradient descent to stay proportional to the activations (Wang, Y et al, 2020).

Gradient Descent can be defined as an optimization algorithm used to find the coefficients of a function to minimize the cost function, usually the coefficient starts small and then the derivative of the cost is calculated by inputting them into the function and then to get smaller values for the coefficients to further calculate cost, the derivative is used to determine the slope of the function at a given point. This algorithm is the reason why the linearity of the ReLu function is efficient for this operation.

However, research performed by researchers from the Google Brain time has proposed an alternative to ReLu, Swish. Swish looks similar to ReLu as seen in Figure 4 but the difference between them is that smooth slope, as Swish is a smooth activation function it bends slight at 0 towards values < 0 then curves upwards which means that Swish unlike ReLu is non-monotonic (P. Ramachandran, 2017). Monotonic is defined as a function that never decrease or never increases, it only increases or decreases which can be seen in ReLu with the linear path starting at 0, the advantage of being a non-monotonic function is the output will be smooth allowing for better optimization towards minimising loss.

# 2.2 Researching the development of Neural Networks

In the previous section different neural network architectures for image segmentation were investigated and some references were made to common components of neural networks such as activation functions and gradient descent. This section will provide an in-depth look into the common components that go into the development of neural networks.

Artificial neural networks have existed as a concept since the 1940’s with researchers McCulloch and Pitts studying the potential and capabilities of the interconnection between several components based on the model of the human brain’s neurons (Y. Huang, 2009). Several experiments creating training algorithms and neural network architectures such as Perceptron, an architecture created by Frank Rosenblatt which was a patch panel that used different combinations of patches as inputs and used potentiometers as adaptive weights (R. Seising, 2018).

Further research and advancements in the field of machine learning have led to a typical template for a neural network, the basic components of an artificial neural network are an input layer, hidden layers that are responsible for computation and the output layer. In addition to these basic components, learning happens through two steps, Forward-Propagation and Back-Propagation.

Forward-Propagation works by randomly guessing the answer to the input that the network has been given, it does this by randomly initializing weights which is then multiplied by the input layer to create a hidden layer and the output of the hidden layer will be through an activation function to form the guessed output.

Backward-Propagation attempts to minimize the loss of accuracy between the guess from the network and the actual answer of the problem, to accomplish this first loss is calculated by passing the expected value from the training set and the value from forward-propagation through a cost function then calculating the derivatives of each weight which will be multiplied by the learning rate, the results from this are subtracted from the weights (M. Malik, 2018).

# 2.3 Appropriate Libraries

Before development of the neural network begins, several libraries that could aid the effectiveness and efficiency of the neural network should be investigated as it increases functionality and the chance of developing a successful neural network.

## 2.3.1 NumPy

NumPy was created in 2005 as an open-source project facilitating the use of numerical computing within Python (Oliphant, T, 2021).

NumPy has been included in numerous scientific projects such as the first imaging of a black hole, confirming the existence of Einstein’s Gravitational Waves and is widely used within the machine learning community as it provides libraries for computer vision and others to accommodate different machine learning tasks (Oliphant, T, 2021).

By implementing NumPy arrays within a project, it enables efficient implementations of numerical calculations which is extremely important in a machine learning project due to the amount of data that must be calculated and any increase in efficiency in regard to computation leads to less time required for training the network (Stéfan, V.W et al., 2011).

## 2.3.2 Scikit-Image

Scikit-Image provides a large collection of useful functions for the task of image processing and just like NumPy, Scikit-Image was created as an open-source project (van der Walt, S, 2021).

The benefits of applying image processing operations such as the ones provided by Scikit-Image for the task of image segmentation are exponential in improving the performance of the neural network and the results outputted from the network after training (Exeter Data Analytics, 2021).

The reasoning behind using a library such as Scikit-Image is that it includes functions that are vital to an image segmentation task that would otherwise require a custom implementation to perform the same task, examples of this include the transform feature that allows for multiple image manipulation techniques such as resize, rotate, and rescale which are commonly used during image segmentation tasks.

Including Scikit-Image in this project is justified due to the nature of the task being image segmentation of cracks in concrete structures and as previously mentioned, Scikit-image contains a lot of image processing operations that enable rapid development and experimentation enabling research to be conducted at a faster rate rather than implementing new functions to accomplish the same operations that Scikit-image provide (Stéfan, V.W et al., 2014).

# 2.4 Researching common structural defects in concrete

The amount of reliance in society on concrete is enormous and it increases each year with the construction of new buildings, structures like bridges, tunnels and the arteries of the modern world, roads.

Governments around the world are very aware of the importance of their infrastructure and in particular their road networks and any form of defect could possibly reduce the safety and performance of the road therefore having inspection and maintenance methods is vital in maintaining high quality infrastructure (Y. Shi, 2016).

Some defects found in concrete and more in particular, reinforced concrete, include creep, spalling, delamination, cracks and seepage, all these defects are common in concrete structures. The importance of identifying these defects is immense due to the reliance on reinforced concrete by so many structures such as railway tunnels, bridges and dams.

Creep is a slow deformation usually exhibited by concrete under sustained stress and proceeds at a decreasing rate each year, this is related to shrinkage during the pouring of cement to form concrete, severe cases of creep deformation can lead to significant displacement within the structure and possibly could cause catastrophic failure to the structure (C.E. Reynolds, 1981).

Delamination is a defect which results in the separation between the layers of reinforcement, it can be caused by a variety of reasons such as incompatible blending materials and low melt temperature within the cavity. Delamination can occur when there is a fracture within the adhesive bond between the layers of reinforcement, a resin fracture within the reinforcement itself or the resin debonding from the reinforcement. In situations where a roof of a tunnel starts to become delaminated, the results can be disastrous in terms of safety and structural integrity (J. Liangbao, 2020).

Spalling and Seepage are related to each other as it involves some kind of liquid penetrating through the concrete due to concretes natural porous surface, in each case discoloration and flaking on the surface can occur and if nothing is done to prevent further damage, it can lead to corrosion of the steel reinforcement reducing the structural integrity (S. Anupoju, 2020).

Cracks can form naturally in concrete after years of stress being put on the surface or they could occur due various other reasons such as poor construction, environmental variables such as heat or even just due to the design of the structure and the concrete settling. As the main defect this project is attempting to automatically identify using the neural network, it is important to mention that cracks can either be purely cosmetic or are signs of a weakness within the structure, this is why it is crucial to find and identify cracks (S. Anupoju, 2020).

# 3. Methods

This section will detail what must be accomplished to be able to finish this project and have a successful result which answers the question of what effects do different activation functions have on the performance of the neural network.

# 3.1 Research Method

The aim of this project is to produce a convolutional neural network that is capable of analysing the cracks in concrete structures with an investigation into the effects that different activation functions have on the performance of the neural network.

To evaluate this aim correctly, the research method for this project will rely on the agile development methodology, this method allows for an iterative project structure which allows for different stages of the project to be completed at separate times unlike linear methods such as the waterfall methodology that plan stages from beginning to end with little to no iteration (Y. Lindsjørna et al, 2016).

The planning stage of the project has been carried out by performing research in the form of the literature review that has been carried out completing the secondary objectives of the project, the results of this research for the creation of planning documents such as test plans and development of the neural network can begin.

Once both the planning and development stages of the project are carried out, to evaluate the performance of the network, testing will be performed and in the case of this project, different functions will have to be recorded and compared against each other to properly evaluate the performance of the network with each activation function.

Taking the results from testing and properly analysing the outcomes of each test will provide information about the accuracy, loss and effectiveness of the neural network, this information will then be used to draw a conclusion on the effectiveness and viability of this project.

# 3.1.2 Establish a test plan for the neural network

Using the knowledge gathered from the literature review and initial planning regarding functionality of the neural network, a test plan will be created for the neural network comprised of the different activation functions which the performance of each will then later be charted into table. This will allow for simple comparison between each activation function regarding statistics such as loss and accuracy.

# 3.1.3 Stage 1 Prototype: Gather data from pre-existing datasets

In the first stage of development, finding a valid dataset for the purpose of crack analysis is vital. During the research stage, CrackForest, a dataset created for automatic road detection of cracks became an ideal candidate. The dataset will need to be prepared for use with the neural network by creating masks using the included MAT files and converting the files into PNG images so they can be properly read by the neural network.

# 3.1.4 Stage 2 Prototype: Initial neural network created using TensorFlow

Using a Keras based neural network using the U-Net architecture as a starting point for the neural network, adjustments will have to be made to have the network work with the dataset and further additions such as functions to stop training when it reaches a certain level of consistency during training and saving results from the data augmentation as NPY files to be used to continue training the model.

# 3.1.5 Stage 3 Prototype: Neural network requires testing before full training

During development of the network, testing should be performed to confirm that the network is able to perform image segmentation on the dataset without errors, at this stage it is not vital to achieve a high level of efficiency as testing is being used to reveal errors within the code not to judge performance.

# 3.1.6 Stage 4 Prototype: Fully training the neural network

When all the errors have been resolved, the model will undergo constant training to achieve peak efficiency with the current activation function which during development will be ReLu. This stage will provide a base level of performance and provide results for the first activation function for testing.

# 3.1.7 Conduct testing

Once the development of the neural network has been finalised then each activation function will be tested by changing the activation functions that each layer in U-Net utilizes apart from the final Sigmoid function. Once training has completed for each activation function, the performance will be recorded.

# 3.1.8 Analyse test results and draw a conclusion

Comparing the results of the testing stage by looking at each activation functions performance within the network regarding accuracy and loss of each function. These results will aid the development of a conclusion by demonstrating the efficiency of each activation function regarding the purpose of image segmentation for cracks.

# 3.2 Data Preparation

During the planning stage of this project, large amounts of data preparation took place to ensure that training can be performed with consistent and accurate data to ensure the best chance for this project’s success.

Data preparation is one of the most important elements in a neural network’s ability to learn on a given dataset as the quality of the dataset provided greatly effects the effectiveness and potential for the neural network to learn.

To improve the quality of the dataset, image processing techniques must be performed on the original images and masks so that the network can reach peak performance during training and validation, changes such as changing the colour of images from RGB to grayscale images reduces the complexity of the image and reducing the number of pixels that will need to be calculated by the network therefore increasing performance.

Network architectures such as U-Net take advantage of image augmentation heavily to circumvent the size of small datasets, by augmenting the original images and masks by applying transformations such as rotations, flipping the images, changing the scale of the image and other techniques that enable the network to create multiple versions of the same images with different transformations applied to reduce the amount of time required to train the network and increase the network’s performance.

## 3.2.1 Mask Conversion

The first challenge presented by the chosen dataset of CrackForest, a collection of images that show cracks in concrete structures such as roads with image masks provided to allow our network to change, was the formatting of the image masks.

CrackForest have formatted their image masks using an array format designed for use with MATLAB, for this project these masks would need to be converted to PNG images rather than attempting to natively input the MAT file format into the network as it would be easier to use the same image processing techniques on both the masks and images rather than writing different functions for the images and masks.

The conversions of the MAT files were performed using a Python script that changed the values within the area into a corresponding colour value e.g., 2 becomes 255 and 1 becomes 0. This allowed the MAT files to be given a graphical representation in the format of a PNG image with 1’s being represented as white and 0’s being represented as black.

Using a Python script for this task rather than using MATLAB to perform the conversions allowed for greater control of the masks produced by the conversion, as there were issues with some of the masks having areas of the mask being represented by 3 and 4 that represented areas of the images between the cracks, as this was not the areas this project is looking for the Python script allowed changing those values to 0.

An example of the masks produced by this process can be found below in Figure 7 – Example of Newly Converted Mask.



Figure - Example of Newly Converted Mask

## 3.2.2 Standardisation

Some datasets include images of varying image resolutions and could also have different image formatting such as having different colour spaces such as BGR, CMYK and RGB. Many neural network architectures require the images to follow a certain size to create square images to be used as inputs.

By eliminating the variances of resolution that the images have from each other, data can be changed to be consistent and fit the requirements required by the network’s architecture. This can be accomplished using various methods but for this project, use of the Scikit-image library to resize the images and mask images would be the most suitable method due to the consistent use of Scikit-images functions.

Another method of image standardisation was included in the data python script using Image and Mask generators which are methods from the Keras framework that allow the use of real time data augmentation by looping through the images in batches, this allows for the images to be resized and have additional augmentations applied such as rotations and changing the brightness range.

## 3.2.3 Image Augmentation

One of the many issues presented when working with image-based machine learning tasks is needing to adjust the original image to allow for better performance and more accurate results.

There are two methods that can be possibly used within the project for training the model, model.fit and model.fit\_generator. When using the generator method, random image augmentations are applied ranging from flipping the image vertically and horizontally, rotating the image, changing the zoom of the image and changing the height and width of the image using a data dictionary that is passed through to the ImageDataGenerator method in the data class.

Once these images are generated, they can then be used to create a npy file that contains the original training image and the newly generated images to provide more data and reduce training time, this npy file can be passed into the model.fit method but during development observations pointed to passing only the original training data in a npy file to model.fit and running for more epochs than the generator method led to better results from the dataset.

Due to this observation, model.fit was used as the main method for training the model and image augmentations were applied to the data before being added to a NumPy array. The image augmentations used were CLAHE and adjusting the gamma of each image in the dataset.

Gamma correction is important in dealing with the brightness of an image as gamma affects how light and luminosity is perceived in an image and by extension how a neural network would see that image, by adjusting the gamma of the image darker areas of the image can be changed to be lighter and more visible and lighter areas can be darkened to improve visibility.

CLAHE or Contrast-limited adaptive histogram equalization can be described as an adaptive histogram equalization technique used to increase the contrast of an image that is in grayscale, CLAHE accomplishes by changing the image values in regions called tiles rather than performing the operation on the entire image at once. The benefit of using CLAHE is that the adaptive nature of its operation allows for greater image clarity after equalization compared to a standard histogram equalization that is applied to the entire image at once as each region can be changed based on its values.

Using a Python library called OpenCV which includes a function for CLAHE, the size of the regions can be set using a variable called tileGridSize and a threshold for contrast limiting can be set using the variable clipLimit, for this project tileGridSize was set to 8x8 which is 8 rows by 8 columns and clipLimit was set to a threshold of 2.0.

# 

# 4. Implementation

This section will detail the development process of creating the neural network and the implementation of specific features that are important to achieving the task of image segmentation of concrete structures to find deformations within the image.

Features of the implementation will be discussed such as the programming language of choice, machine learning framework and the libraries required to develop the project, this will be the focus of this section.

## 4.1 Neural Network Foundations

The first challenge during the implementation stage was finding an existing code base that could accomplish similar tasks to the planned image segmentation task of this project.

The U-Net Keras implementation created by Zhixuhao was chosen to be the starting code base for this project as it already had an implementation of the chosen architecture and was created using the Keras api using the Tensorflow machine learning framework as the backend of the api.

To accommodate the required Python version and required libraries, Anaconda was used as an environment manager which allowed for an isolated Python environment to be created and used exclusively with the neural network.

Using an environment opposed to installing the required Python version and libraries directly to Windows provides benefits such as complete independence from system libraries, minimises the risk of making mistakes that require a complete reinstall system wide and the ability to use and verify that specific versions of the libraries and Python implementation are used.

Once the preliminary setup had been completed, the next challenge was converting the dataset to a usable medium, adapting the network to use the chosen dataset and applying different image standardisation techniques to ensure that the neural network is training at peak efficiency.

## 4.2 Dataset Conversion

CrackForest, the chosen dataset for this project, uses 3 different file types in the original dataset. These file types include JPG for the images of the cracks, MAT files included as the masks or otherwise known as the ground truth in the case of the CrackForest dataset which are files created as a measurement of the targeted element of the images using observation, in this case these measurements are of the cracks in the concrete images. The CrackForest dataset also includes SEG files usually used to contain seismic data but for the purpose of this project, these files are not required.

To begin with the JPG files for the images were converted to the PNG image format as the original code base used PNG files most likely for the lossless compression and high contrast images that PNG can provide especially when augmenting the images with image processing techniques, this conversion required a simple implementation to perform the necessary operations as seen in Figure 8 – Conversion of JPG to PNG.

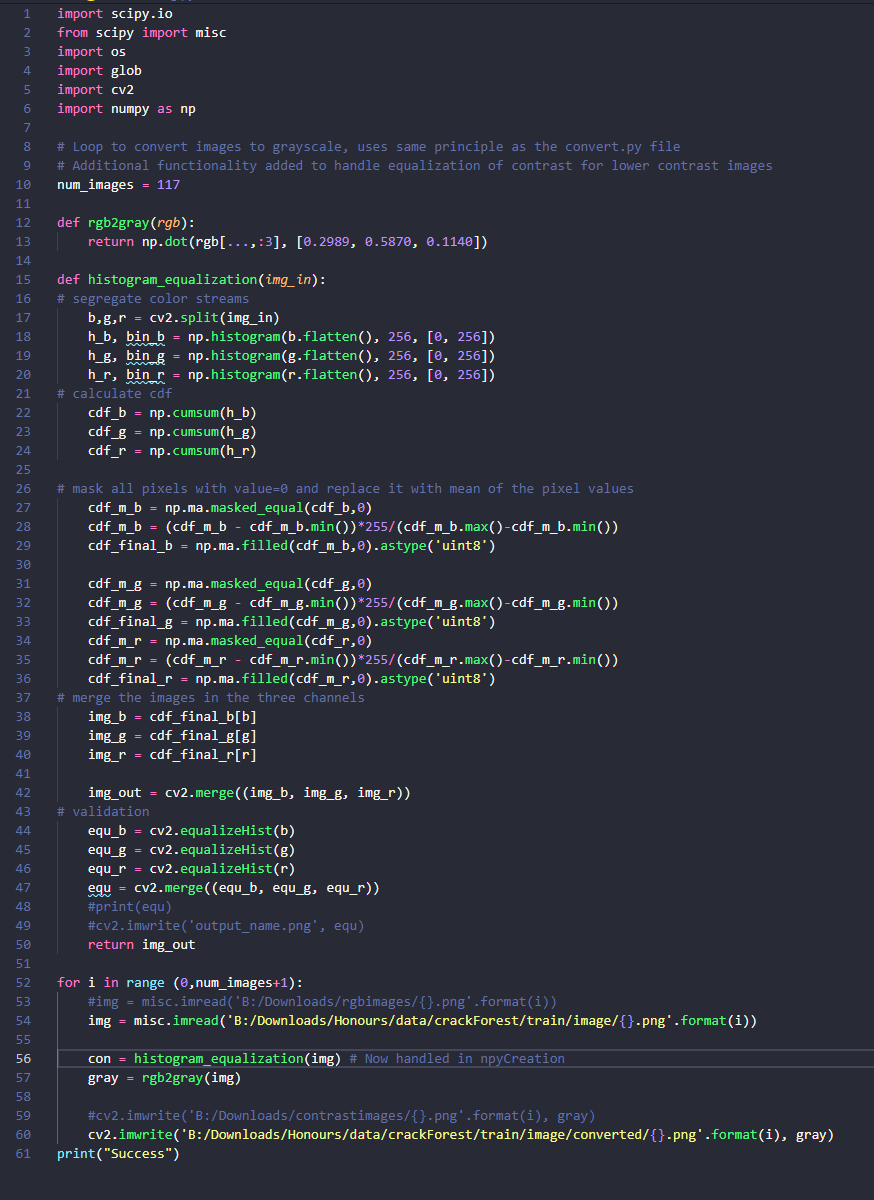


Figure - Conversion of JPG to PNG

In Figure 8, multiple methods can be seen in the finished Python class of concreteAug.py which can be used to apply image processing techniques such as histogram equalization and converting the RGB image to grayscale.

From this class, it was decided that only converting the images to grayscale then saving the images to PNG was required as the process of histogram equalization was replaced with a better alternative with CLAHE in the NPYCreation.py class while creating the NPY files required for training as CLAHE is an adaptive histogram equalization that works using sections of the image rather than the standard histogram equalization.

The more complicated task with the dataset conversion lies with the MAT files used as the masks due to MAT being a proprietary file type of Matlab which is a computing environment and programming language that makes use of algorithms to accomplish numerical computations such as creating MAT arrays. Python with the use of the NumPy library can achieve much more complex computations with a higher level of efficiency so a Python class was created to handle the conversion from MAT to a PNG image format.

Conversion was accomplished by specifying numbers in the MAT array as cracks and concrete by assigning a divisible value that will either equal 1 or 0, this was achieved by setting any number representing a crack with 2 as white in the grayscale image colour space as 255 which will equal 1 when divided by 255 and setting any other number in the array as 0 for black.

The code for this conversion can be seen below in the Figure 9 – MAT Conversion. In the figure, a for loop is initialised with a maximum value of 117 as it will loop 118 times from 0 which is the number of images within the CrackForest dataset, from there it will find the values used to identify the cracks by specifying the field of “groundTruth” from within the MAT array and will then change the value and save the image as a PNG image.

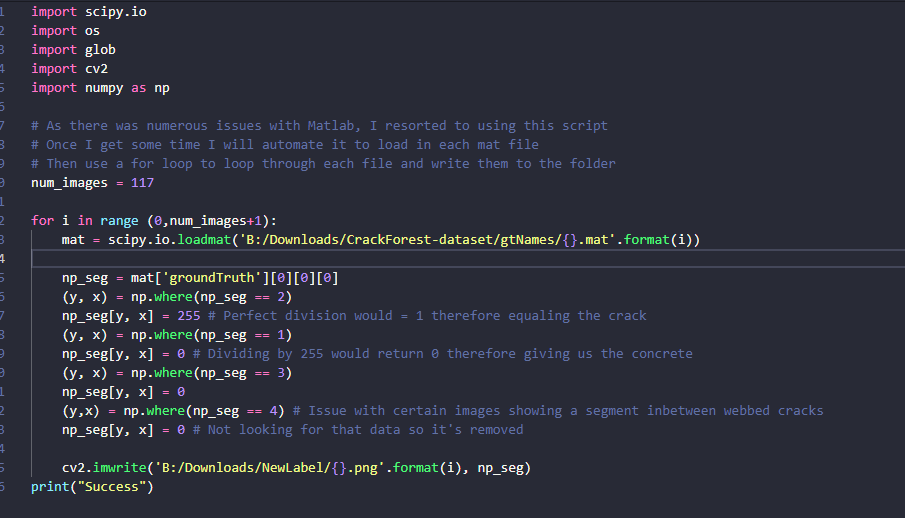


Figure - MAT Conversion

## 4.3 Data Adjustment & NPY Creation

As previously mentioned in the data preparation section, a lot of data preparation methods need to be created to allow for the standardisation and manipulation of images which will then allow for greater efficiency and reliability during training.

The original code base from Zhixuhao included multiple data preparation methods in the data.py class but due to deviations taken during development, the data.py class is used rarely and is now mainly responsible for loading in the generated NPY files created from a new Python class called NPYCreation.py.

The new class was created as a way of modifying the original data.py class methods without affecting the original class, this was carried out as most of the original methods from data.py would not be used by model.fit and would only be used with model.fit\_generator.

The adjust method from the original data.py class was modified to call newly created standardisation methods and instead of dividing the image values by the original 255 it now uses a more robust system of finding the minimum and maximum of the image value and dividing it by the maximum. Additional methods are called from the adjustData method to apply image processing techniques to the original images, these include CLAHE and gamma correction methods which as previously explained CLAHE applies histogram equalization to sections of the image therefore balancing contrast and gamma correction balances the brightness levels of the image.

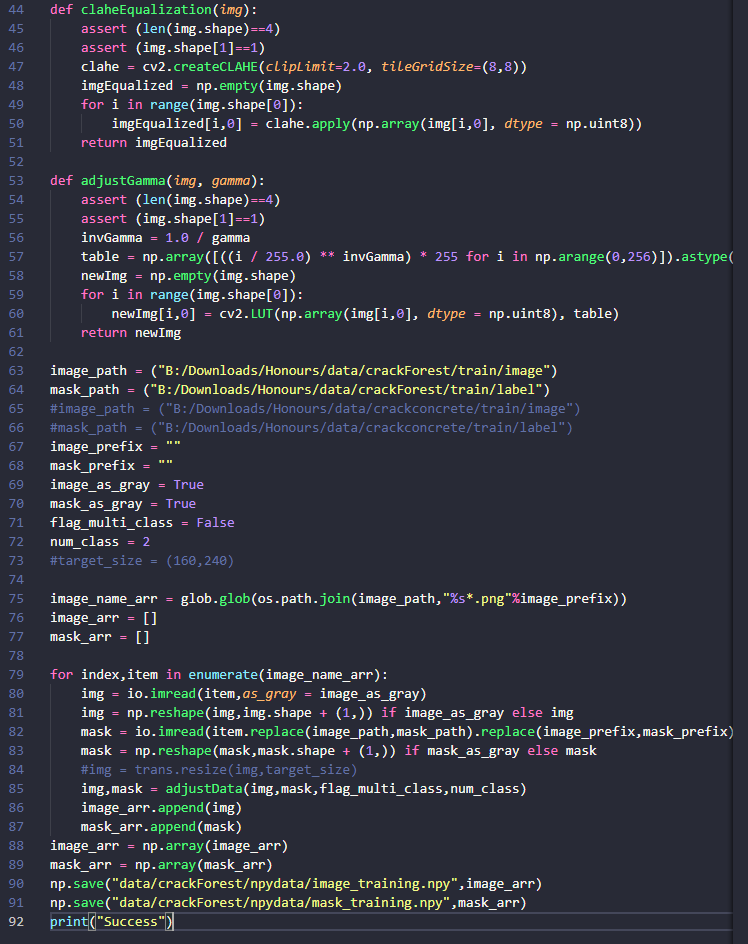


Figure - npyCreation.py image processing methods

In Figure 10 – npyCreation.py image processing methods, as seen above, it shows the implementation of the For loop that cycles throughout the entire dataset of 118 images and 118 masks by reading each file as a grayscale image then calling the adjustData method and finally adding the newly augmented file to the array and saving that array as a NPY array file.

When claheEqualization is called it utilises the cv2 library implementation of CLAHE equalization with the input parameters of a tile size set to 8 x 8 so that CLAHE can only be applied to an area that is 8 pixels by 8 pixels and a clip limit of 2 to limit how much CLAHE can change the image contrast in that section.

The adjustGamma method is much simpler than the CLAHE implementation seen in the claheEqualization method, a gamma value of 1.2 has been set then an inverted gamma variable will be created using invGamma calculating the inverse of the gamma value provided, from there a lookup table is created to facilitate the LUT method from the cv2 library which will perform transformations based on the values found in the look-up table therefore performing gamma correction to any image passed through the method.

The two image processing methods of claheEqualization and adjustGamma are called inside of the adjustData method seen in Figure 11 – adjustData method below.

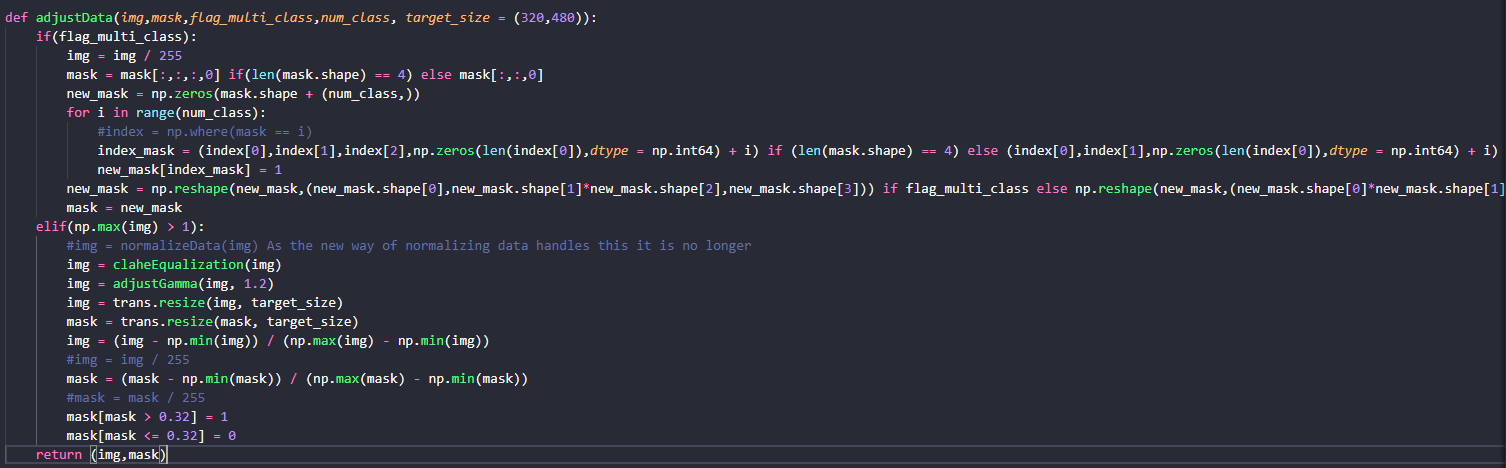


Figure - adjustData method

In Figure 11, a commented call to a method known as normalizeData can be seen, this was later changed to performing the normalization within the adjustData method as it became clear that normalizing the data with one line per image set was more efficient than an entire method.

## 4.4 Neural Network Models

The essential part of any neural network is the models as this is how neural networks form relationships with mathematical expressions. The models for this project were based around the U-Net architecture which works on the principal of convolutional layers that forms two parts that are symmetrical to each other; the contracting path and the expansive path.

The contracting path of U-Net focuses on convolutional layers that increase the depth of the input and then reduce the size with each max pooling function, this type of path can be seen in all of the models created for this project but as an example of a standard U-Net implementation, code has been included in Figure 12 – U-Net Code Example below.

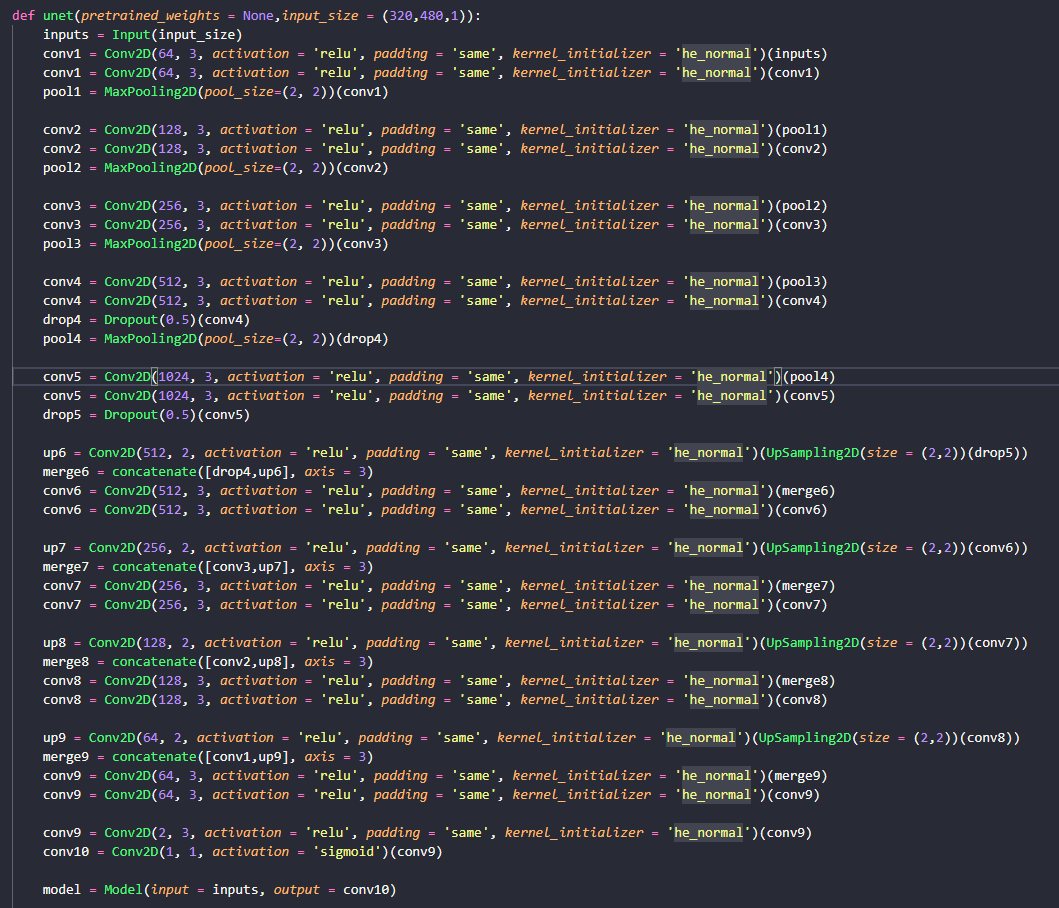


Figure - U-Net Code Example

This figure demonstrates a standard implementation of the U-Net Architecture and the contracting path can be seen in conv1 to conv5 following the pattern of convolutional layer leading into another convolutional layer then into a max pooling function and optionally a dropout function can be added to the end of that pattern which can help to regulate the neural network by randomly dropping layer outputs based on the concept that dropout can prevent or stop situations where the network can learn to correct mistakes and therefore make the network more reliable.

Further down in the Figure, the expansive path can be seen. The expansive path focuses on upscaling the image using either the UpSampling2D method provided by Keras or using transpositional convolution layers, the difference between these two methods is that transpositional convolution layers do not have a defined tuple for upscaling the image like UpSampling2D in this case 2 x 2 but instead uses learnable parameters that learn the optimal interpolation method for that specific image to upscale.

During development, this model was refactored many times to test the efficiency when upscaling in the expanding path and the performance difference exhibited when changing the model between transpositional convolution layers and UpSampling2D was minimal if not identical so in most models it was decided that UpSampling2D would be used for the upscaling function.

The issues faced with the standard U-Net implementation during the development of this project were with very little detail being discovered in the final results produced by the neural network so a hypothesis was derived on the basis that detail must be getting lost in both the contracting and expansive paths therefore a model would have to be created that simplified the standard implementation of U-Net into a smaller version.

Below in Figure 13 – Simple\_U-Net example, a simpler iteration of the standard U-Net architecture had been created and worked on the initial idea of reducing the number of channels compared to the standard implementation with the highest channel amount being 1024 in the standard implementation and the newer simplified version having a maximum of 512.

This difference in channels did lead to better performance but results were still not producing the expected results and therefore further iteration had to be done on simplifying U-Net for this project task.

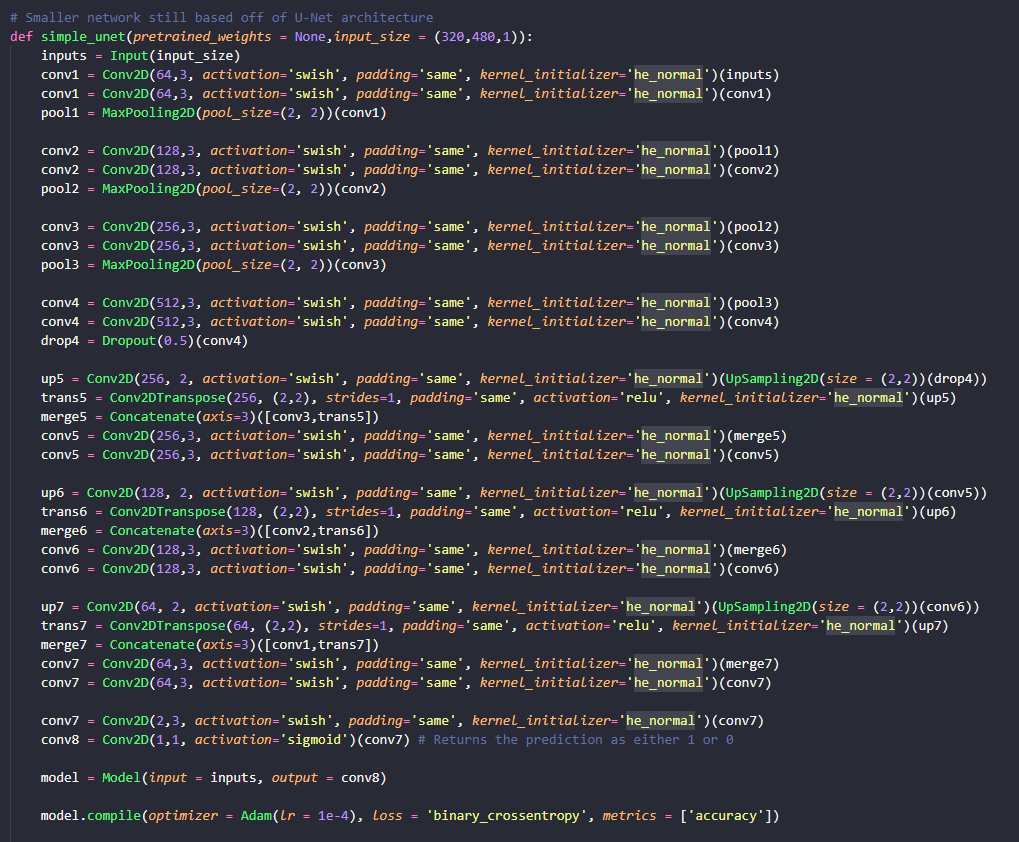


Figure - Simple\_U-Net example

The final iteration of the simplified U-Net model reduces not only the number of channels per layer but also reduces the overall total number of layers used in the model and these adjustments lead to finally getting more detailed results back from the output of the neural network.

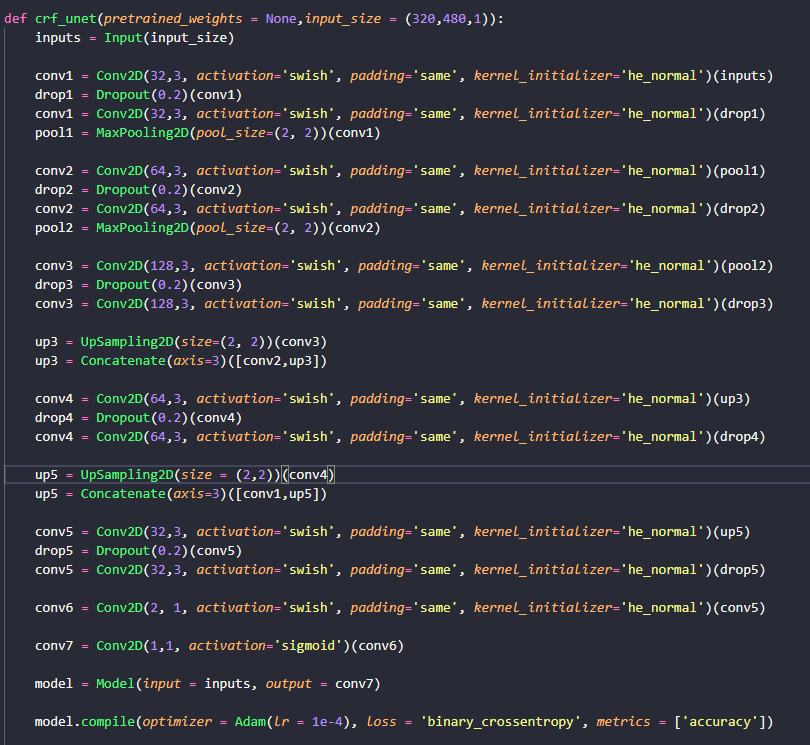


Figure - Final U-Net iteration

The final iteration of the simplified U-Net model, seen in Figure 14 – Final U-Net iteration, reduces the number of total layers and reduces the number of channels, this simplification of the original U-Net architecture helped to achieve the desired results from the CrackForest dataset.

As seen in the model, the activation functions have been set to swish but usually these models are used with the ReLu activation function and were changed to Swish for the purposes of testing the performance differences that different activation functions offer the neural network. The last layer of the models is always set to the sigmoid activation function as it will only return one of two values, 1 or 0, providing a definite result.

## 4.5 Custom Activation Functions

To allow for testing multiple activation functions and investigate the effect on performance that different activation functions would provide for this project. The chosen activation functions to investigate for this project are ReLu, Swish and LeakyReLu as all three of these activation functions have been used extensively within the machine learning community for computer vision projects.

ReLu is provided as a standard activation function within the Keras framework so to test and include ReLu as an activation function within the model, only the parameter within the activation field will need to be changed to ‘ReLu’ and this was the main activation function used during development of the project.

The main challenge when investigating these different activation functions was that Swish does not have an implementation yet in the Keras framework and while LeakyReLu is included within a library known as advanced activations it required a separate class otherwise it would cause an error during runtime.

LeakyReLu was implemented easily after creating a separate class that could be called into the models.py class to be used by making a call within the model and stripping the convolutional layer of its activation parameter, to demonstrate this an example of this can be seen in Figure 15 – LeakyReLu Model below.

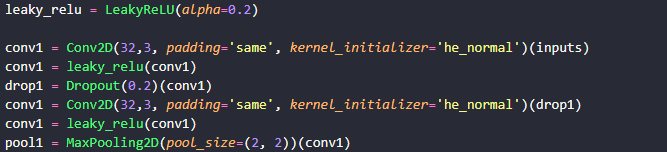


Figure - LeakyReLu Model

The class that initialises the LeakyReLu activation function was created due to an error during runtime called AttributeError that prevented the model from training as the LeakyReLu function would not be recognised within the model when it was declared from within model.py but by abstracting the initialisation into a separate class file and initialising LeakyReLu in that class file prevented this error from reoccurring.

Implementing Swish was slightly more complicated than implementing LeakyReLu as Swish has not been officially added to the Keras framework, due to this a custom activation function would need to be created to facilitate testing the Swish activation function. The formula for Swish can be simply defined as *f(x) = x \* Sigmoid(x),* as the formula is already known and can be defined simply the next challenge was to define the formula as a custom activation function.

To begin, a new class was created called swish.py and the standard imports to the Keras backend and the layers library within Keras were made with the new addition to the imports being generic\_utils which allows the use of a function called get\_custom\_objects that will allow the creation of the custom activation function. At this point the class was declared and the formula could be defined within a method called swish using the sigmoid function from the Keras backend and the parameter ‘x’ used to represent the input neurons defining the formula as (K.sigmoid(x) \* x).

The last step in creating the custom activation was to use the get\_custom\_objects function to update the global dictionary of custom objects to allow the model.py to use ‘swish’ as a parameter to call the newly created Swish activation function, the implementation of the activation function can be seen below in Figure 16 – Swish Implementation.

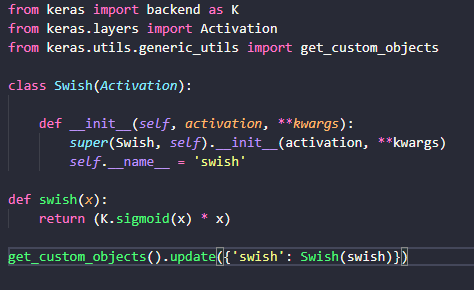


Figure - Swish Implementation

# 5. Testing

This section will discuss the investigation into the performance differences that different activation functions can give the network and the results from training the network on the task of finding cracks within concrete structures.

To evaluate the different activation functions, a test plan was created that would give each activation function an equal start as to keep results reliable and accurate. Each activation function would be run for a total of 50 epochs with a batch size of 4 and a validation split of 0.4 which would use 60% of the dataset for training and 40% for validation, each test would be carried out three times and the final test would be recorded within the test table.

Below the results from the testing can be seen in the multiple tables below starting from Table 1 – Activation Functions Testing 1, displaying the values for accuracy, loss, validation accuracy and validation loss which will be used to compare the performance of the function and will determine which function was most suited to the projects task.

|  |  |
| --- | --- |
| Table - Activation Functions Testing 1 | Table - Activation Functions Testing 2 |

|  |  |
| --- | --- |
| Table - Activation Functions Testing 3 | Table - Activation Functions Testing 4 |



Table - Activation Functions Testing 5

# 6. Evaluation

This section will draw on the results from the testing section and the overall development cycle to evaluate the project against the stated research question and determine if the project was successful based on this information.

The main focuses of this evaluation and for determining the success of this project will be the performance differences of each activation function and the results from the network regarding image predictions.

## 6.1 Activation Functions Evaluation

In the previous testing section, multiple tables can be seen displaying the results from testing the performance of the different activation functions, these results will be the basis of this evaluation of activation functions and will determine the best performing activation function for this projects task.

At the beginning of the test results between epoch 1 and 5, Swish had a strong start with higher accuracy and lower loss compared to the initial epochs for ReLu and LeakyReLu but later results saw Swish plateau and underperform compared to the other two activation functions.

Swish had multiple moments where its benefits could be seen over the other two activation functions, but it failed to maintain a high level of accuracy in both training accuracy and validation accuracy over the course of the 50 epochs, there could be limitations based on the implementation of Swish that was created for this project but considering this, it seems that for this project Swish is not the optimal activation function.

LeakyReLu showed steady improvement throughout the course of the testing and given more training time it is reasonable to believe that it could be used to fully train the network to a high degree of accuracy but the issue with LeakyReLu lies with its learning rate, LeakyReLu while showing gradual improvement also demonstrated a slow rate of learning compared to the other functions and while it would be possible to reach either the same level of accuracy or a higher level of accuracy, the time it would take to reach that level would be significantly higher than the other functions.

ReLu at the beginning was lagging behind the other activation functions but soon after epoch 12, accuracy and overall performance started to increase and the rate of learning rapidly increased towards the end of testing, managing to achieve a higher level of performance the other activation functions.

Overall ReLu demonstrated a high level of accuracy and loss compared to the other activation functions and demonstrated why ReLu is the activation function of choice within the computer vision community. The results from testing the activation functions and the evaluations of each activation function have shown that each function is promising for the task of image segmentation but for this project the activation function of choice is ReLu due to its overall performance.

## 6.2 Prediction Results Evaluation

The goal of this network was to produce a neural network that can identify cracks within concrete structures such as roads and pavements using image segmentation with the CrackForest dataset. The network was created during development and adapted to work with the CrackForest dataset which managed to produce successful results.

These results can be seen below compared to the original image and the mask in Figure 17 – Network Image Result 1.

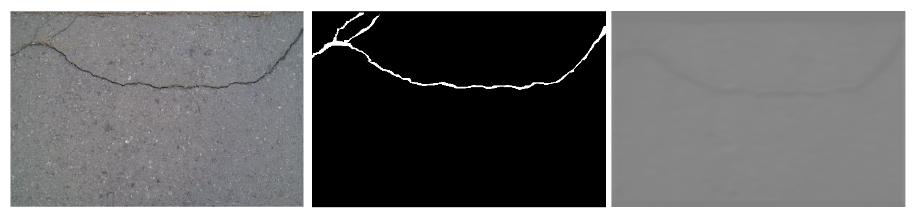


Figure - Network Image Result 1

From left to right, the first image is the original photograph of the defect, the second image is the image mask that will allow the network to make a prediction based on the white area of the image and the last image is the predicted crack outputted by the network, while the colours are faint you can clearly make out the crack being predicted by the network.

Other results provide a clearer image of the predicted crack and demonstrate the potential this network has for the task of image segmentation for finding cracks, these results can be found below in Figure 18 and Figure 19.

A picture containing text

Description automatically generated

Figure - Network Image Results 2

A picture containing text

Description automatically generated

Figure - Network Image Results 3

Figure 18 and Figure 19 provide much clearer examples of the predictions that prove that the network can make accurate predictions based on the images found within the CrackForest dataset but also show the potential for improvement that can be done to the network to help improve prediction clarity and overall accuracy.

However, it was observed in the results that images that featured very thin hairline cracks within the concrete were less likely to show up clearly or be predicted accurately, an example of this can be found in Figure 20 – Network Image Results 4.

A picture containing text, metalware, chain

Description automatically generated

Figure - Network Image Results 4

While the prediction did find the deeper crack at the bottom of the image, it failed to find the small hairline cracks surrounding the larger cracks. This result further enforces the idea that the potential for improvement to the network and refinement is exponential.

# 7. Conclusion

## 7.1 Project Resume

The problem with aging structures in the modern world has led to a decrease in safety and quality of infrastructure which can lead to further costs invested into inspections and overall maintenance. Many different solutions have been attempted to resolve these issues over recent years but with the growing capabilities of neural networks and computer vision-based machine learning, it was clear that a neural network could be used to resolve these issues and could potentially perform better than methods that still require human interaction.

To investigate the existing solutions in industry and feasibility of a neural network for this task, a literature review was performed to gather research and provide a basis to justify this project.

The main aim of this project was to achieve the goal set by the research question of this project:

**Can a neural network be used to identify and analyse cracks in concrete structures to therefore increase the level of safety and prevent degradation and do different activation functions effect the performance of that neural network?**

To achieve this goal, the projects development focused on producing a neural network that was capable of image segmentation that can be adapted to the CrackForest dataset to train the network on the task of finding cracks within concrete structures. The project would follow an agile development lifecycle involving the creation of requirements, implementation of the network and testing the performance of the network’s predictions and the performance of different activation functions.

The evaluation stage of this project led to the following conclusion as to whether this project was successful in achieving its goal:

A neural network can be produced to identify cracks in concrete structures and different activation functions provide benefits and disadvantages to performance based on the machine learning task.

## 7.2 Results Discussion

The results of both the network’s predictions and the performance of different activation functions were previously discussed in the evaluation stage of this project and from those evaluations, conclusions can be drawn.

### Activation Functions Conclusion

The results from testing different activation functions provided interesting results that led to ReLu performing higher than both Swish and LeakyReLu. This demonstrated that for the task of image segmentation for this network, ReLu was the best overall performing activation function and showed that both Swish and LeakyReLu still had potential in the task of image segmentation.

### Prediction Results Conclusion

The results from the development of the network and training the network to work with the CrackForest dataset provided accurate predictions that demonstrated that it was possible to use a neural network to identify cracks within concrete structures and as mentioned in the evaluation stage, the potential for improvement is exponential.

### Overall Conclusion

Overall, the project successfully demonstrated that it was entirely possible to use a neural network to find and predict cracks in concrete structures and satisfied the goal of investigating the performance effects of different activation functions. While there remains room for improvement in the project, it was successful in producing a neural network that is capable of image segmentation and achieved both the research question and the aim of this project.

## 7.3 Limitations and Future Improvements

### Limitations

Despite the success of answering the research question and achieving the aim of this project it is important to reflect on the limitations faced during development. The project was limited by the lack of data available in regard to defects in concrete, with CrackForest being one of the largest examples available to the public which led to the use of U-Net as the model architecture due to its ability to work on smaller datasets and hardware also limited this project as training a neural network is an intensive task that requires a large amount of computing resources as the computer hardware available was unable to train at the desired parameters which increased the time needed for training.

### Future Improvements

To improve this project further, increasing the quality of the computer hardware available could lead to better training times and possibly increase efficiency and gaining access to a larger dataset for the subject matter would improve the accuracy and reduce the training time required for achieving a similar level of accuracy.

The network may also benefit from including different machine learning techniques such as Gated Scale Pooling which can be seen in similar networks used for image segmentation and a revision on the image processing techniques used on the dataset may be beneficial at improving the accuracy of the predictions.

# 

# 8. References

Shi, X.B. (2014), "Study of Image Segmentation and Image Restoration Based on Variational Method", Applied Mechanics and Materials, vol. 644-650, pp. 4319-4321.

Feng, C., Zhang, H., Wang, S., Li, Y., Wang, H. & Yan, F. (2019), "Structural Damage Detection using Deep Convolutional Neural Network and Transfer Learning", KSCE Journal of Civil Engineering, pp. 1-10.

M-Mahdi Naddaf-Sh, Hosseini, S., Zhang, J., Brake, N.A. & Zargarzadeh, H. (2019) "Real-Time Road Crack Mapping Using an Optimized Convolutional Neural Network", Complexity, vol. 2019, pp. 17.

Chen, K., Yadav, A., Khan, A., Meng, Y. & Zhu, K. 2019, "Improved Crack Detection and Recognition Based on Convolutional Neural Network", Modelling and Simulation in Engineering, vol. 2019, pp. 8.

Li, S. & Zhao, X. (2019), "Image-Based Concrete Crack Detection Using Convolutional Neural Network and Exhaustive Search Technique", Advances in Civil Engineering, vol. 2019, pp. 12.

Reddy, A., Indragandhi, V., Ravi, L. & Subramaniyaswamy, V. (2019), "Detection of Cracks and damage in wind turbine blades using artificial intelligence-based image analytics", Measurement, vol. 147, pp. 1.

Liang, D., Xue-Feng, Z., Wang, S. & Chen-Jing, L. (2019), "Research on Concrete Cracks Recognition based on Dual Convolutional Neural Network", KSCE Journal of Civil Engineering, , pp. 1-9.

Ding, Y., Acosta, R., Enguix, V., Suffren, S., Ortmann, J., Luck, D., Dolz, J. & Lodygensky, G.A. (2020), "Using Deep Convolutional Neural Networks for Neonatal Brain Image Segmentation", Frontiers in Neuroscience.

Laigang, Z., Zhou, S., Li, Y., Sun, Q., Zhao, Y. & Deying, F. (2020), "Image object detection and semantic segmentation based on convolutional neural network", Neural Computing & Applications, vol. 32, no. 7, pp. 1949-1958.

Ma, J., Wu, F., Tian’an Jiang, Zhao, Q. & Kong, D. (2017), "Ultrasound image-based thyroid nodule automatic segmentation using convolutional neural networks", International Journal of Computer Assisted Radiology and Surgery, vol. 12, no. 11, pp. 1895-1910.

Zakeri, H., Nejad, F.M. & Fahimifar, A. (2017), "Image Based Techniques for Crack Detection, Classification and Quantification in Asphalt Pavement: A Review", Archives of Computational Methods in Engineering, vol. 24, no. 4, pp. 935-977.

Dung, C.V. & Anh, L.D. (2019), "Autonomous concrete crack detection using deep fully convolutional neural network", Automation in Construction, vol. 99, pp. 52.

Konig, J, Jenkins, MD, Barrie, P, Mannion, M & Morison, G (2019), A convolutional neural network for pavement surface crack segmentation using residual connections and attention gating. in 2019 IEEE International Conference on Image Processing (ICIP). IEEE, pp. 1460-1464. https://doi.org/10.1109/ICIP.2019.8803060

Jenkins, MD, Buggy, T & Morison, G (2017), An imaging system for visual inspection and structural condition monitoring of railway tunnels. in 2017 IEEE Workshop on Environmental, Energy, and Structural Monitoring Systems (EESMS). IEEE, pp. 1-6. https://doi.org/10.1109/EESMS.2017.8052679

Jenkins, MD, Carr, TA, Iglesias , MI, Buggy, T & Morison, G (2018), A deep convolutional neural network for semantic pixel-wise segmentation of road and pavement surface cracks. in 2018 26th European Signal Processing Conference (EUSIPCO). IEEE, pp. 2120-2124. <https://doi.org/10.23919/EUSIPCO.2018.8553280>

Schumann J., Gupta P., Liu Y. (2010) Application of Neural Networks in High Assurance Systems: A Survey. In: Schumann J., Liu Y. (eds) Applications of Neural Networks in High Assurance Systems. Studies in Computational Intelligence, vol 268. Springer, Berlin, Heidelberg. <https://doi.org/10.1007/978-3-642-10690-3_1>

T. A. Carr, M. D. Jenkins, M. I. Iglesias, T. Buggy and G. Morison. (2018) Road crack detection using a single stage detector based deep neural network. 2018 IEEE Workshop on Environmental, Energy, and Structural Monitoring Systems (EESMS), Salerno, 2018, pp. 1-5. doi: 10.1109/EESMS.2018.8405819.

O. Ronneberger, P. Fischer, T. Brox (2015) *U-Net: Convolutional Networks for Biomedical Image Segmentation. (*Cornell University Library) <https://arxiv.org/abs/1505.04597>

P. Ramachandran, B. Zoph, Q. V. Le. (2017) Searching for Activation Functions (Google Brain) <https://arxiv.org/abs/1710.05941>

Wang, Y., Li, Y., Song, Y. & Rong, X. (2020) "The Influence of the Activation Function in a Convolution Neural Network Model of Facial Expression Recognition” (*Applied Sciences)*vol. 10, no. 5, pp. 1897. <http://dx.doi.org/10.3390/app10051897>

[Kaiming He](https://arxiv.org/search/cs?searchtype=author&query=He%2C+K), [Georgia Gkioxari](https://arxiv.org/search/cs?searchtype=author&query=Gkioxari%2C+G), [Piotr Dollár](https://arxiv.org/search/cs?searchtype=author&query=Doll%C3%A1r%2C+P), [Ross Girshick](https://arxiv.org/search/cs?searchtype=author&query=Girshick%2C+R). (2017) Mask R-CNN (Facebook AI Research) <https://arxiv.org/abs/1703.06870>

Yngve Lindsjørna, Dag I.K. Sjøberga,b , Torgeir Dingsøyr , Gunnar R. Bergersena , Tore Dybåa. (2016) “Teamwork quality and project success in software development: A survey of agile development teams” (University of Oslo) <https://www-sciencedirect-com.gcu.idm.oclc.org/science/article/pii/S016412121630187X?via%3Dihub>

Z. Huang, L. Huang, Y. Gong, C. Huang and X. Wang. (2019) "Mask Scoring R-CNN,", *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, Long Beach, CA, USA, 2019, pp. 6402-6411, doi: 10.1109/CVPR.2019.00657.

Yanbo Huang (2009) “Advances in Artificial Neural Networks – Methodological Development and Application” (United States Department of Argriculture) <http://dx.doi.org/10.3390/algor2030973>

Seising, R. (2018) "The Emergence of Fuzzy Sets in the Decade of the Perceptron—Lotfi A. Zadeh’s and Frank Rosenblatt’s Research Work on Pattern Classification" *Mathematics,*vol. 6, no. 7.

Mukul Malik. (2018) “Basics of Neural Network” (BecomingHuman) Available online at: <https://becominghuman.ai/basics-of-neural-network-bef2ba97d2cf>

Yong Shi, Limeng Cui, Zhiquan Qi, Fan Meng, and Zhensong Chen (2016) “Automatic Road Crack Detection Using Random Structured Forests” (IEEE) IEEE Transactions on Intelligent Transportation Systems, Vol 17, No 12

Sadanandam Anupoju (2020) “Defects in Concrete Structures – Types, Causes and Prevention” (The Constructor) Available online at: <https://theconstructor.org/concrete/concrete-defects-types-causes-prevention/8581/>

Charles E Reynolds, James C Steedman (1981) “Reinforced Concrete Designer’s Handbook” (Viewpoint) Ninth Edition, Section 4, Page 53

Liangbao Jiao, Qiling Ye, Xuehong Cao, Dryver Huston, Tian Xia (2020) “Identifying concrete structure defects in GPR image” (Nanjing Institute of Technology) <https://doi.org/10.1016/j.measurement.2020.107839>

Oliphant, T (2021) “*NumPy - About Us*.” (Numpy.org) Available online at: <https://numpy.org/about/>

van der Walt, S (2021) “scikit-image: image processing in Python — skimage v0.18.0 docs” (Scikit-image.org) Available online at: <https://scikit-image.org/docs/stable/overview.html>

Exeter Data Analytics. (2021) “Scikit-Image : Image Processing with Python” (Exeter Data Analytics) Available online at: <https://exeter-data-analytics.github.io/python-data/skimage.html>

Stéfan, V.W., Schönberger, J.,L., Nunez-Iglesias, J., Boulogne, F., Warner, J.D., Yager, N., Gouillart, E., Yu, T. & the scikit-image contributors (2014), "scikit-image: image processing in Python" (PeerJ)

Van Der Walt, S., Colbert, S.C. & Varoquaux, G. (2011) “The NumPy array: a structure for efficient numerical computation”, Cornell University Library, arXiv.org, Ithaca.

# 9. Appendices

## Appendix A – Code Listings

The original source code and GitHub Repository can be found at:

### GitHub Repository

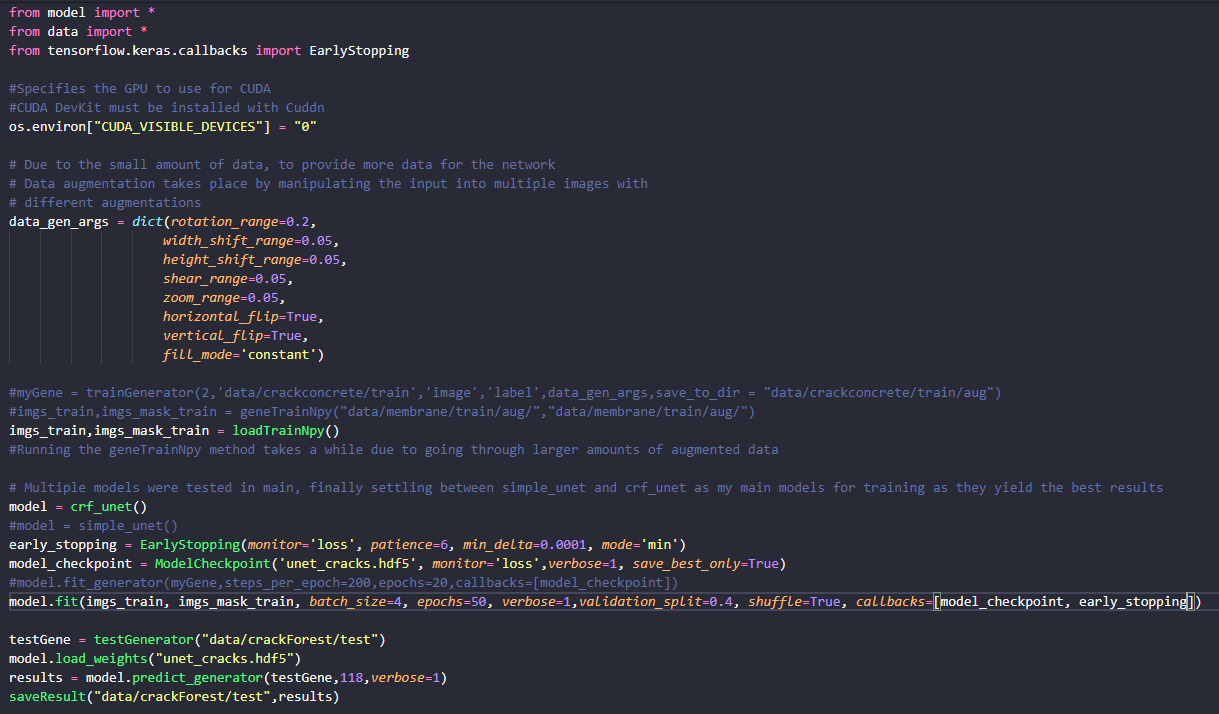
<https://github.com/BenMaxGCU/Honours>

### Full Project Upload with Prediction Results & NPY files

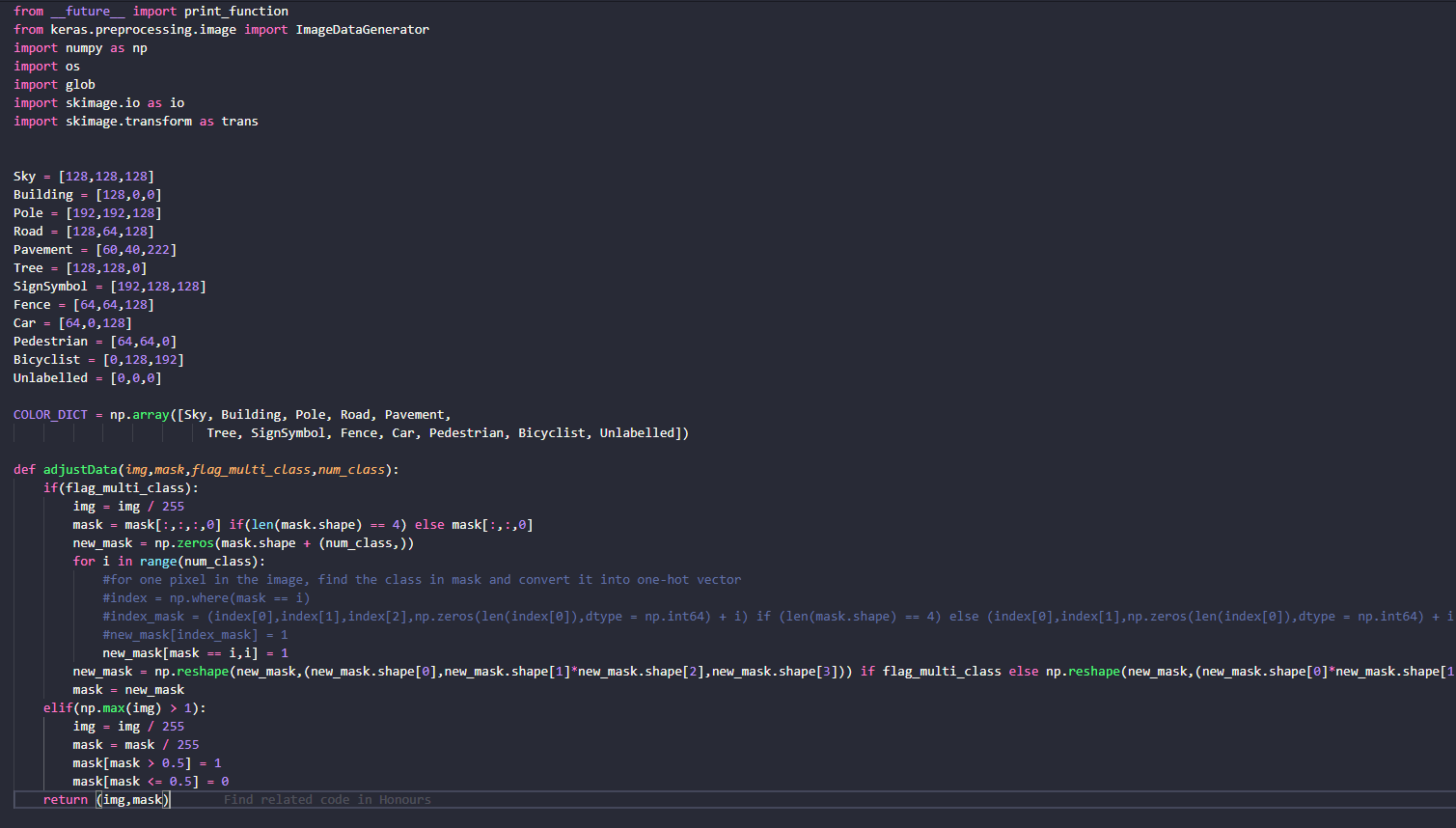
Due to the size of this project, the full project with documentation needed to be hosted on an external file sharing site:

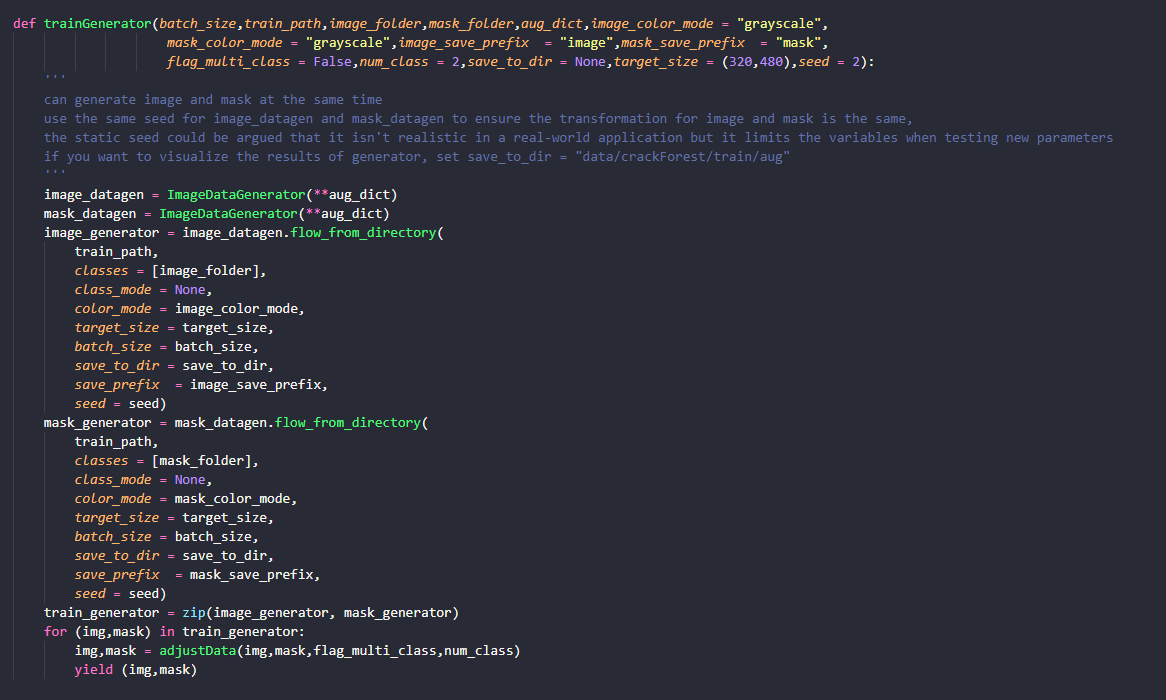
<https://drive.google.com/drive/folders/1I_nSDfLkYBuIE0qs-RBuFZGOGL4dDCs_?usp=sharing>

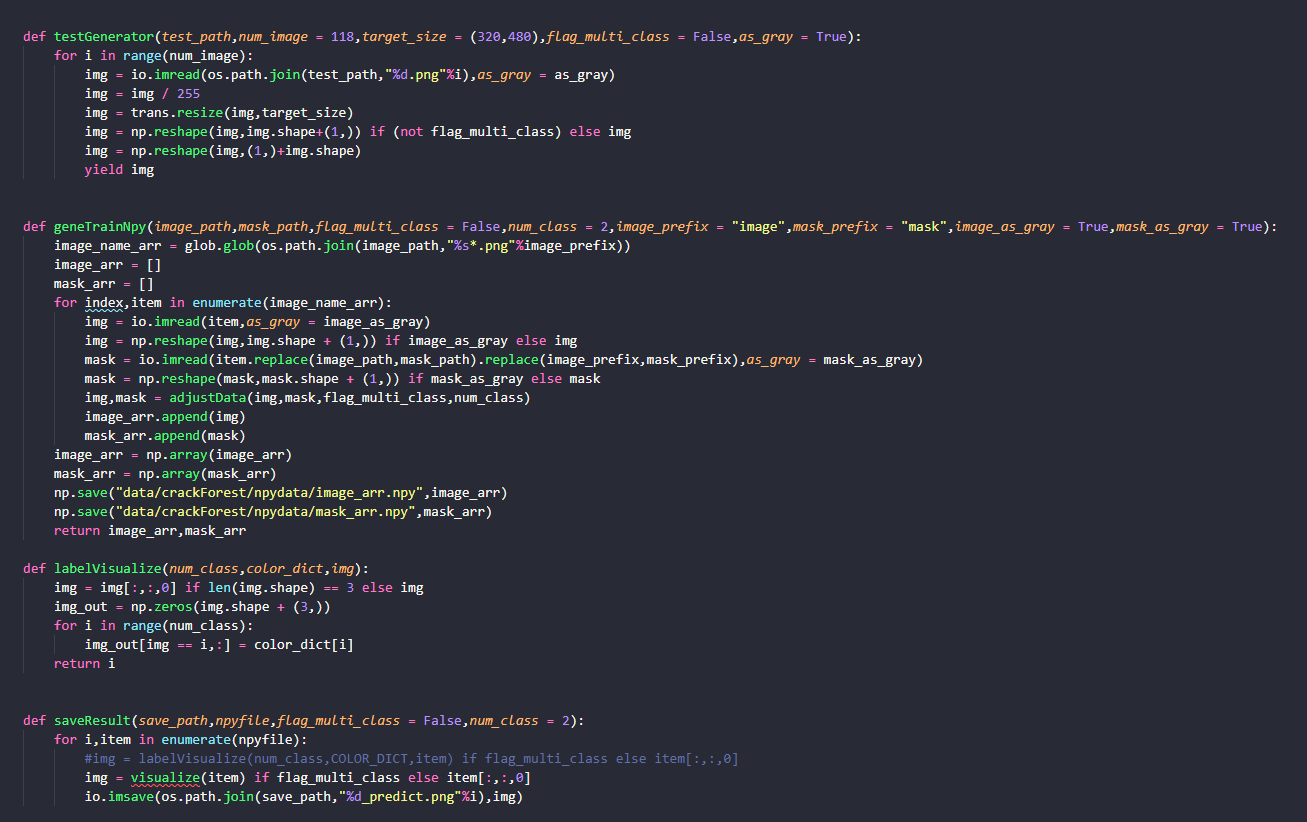
## Appendix B – Main.py of Neural Network

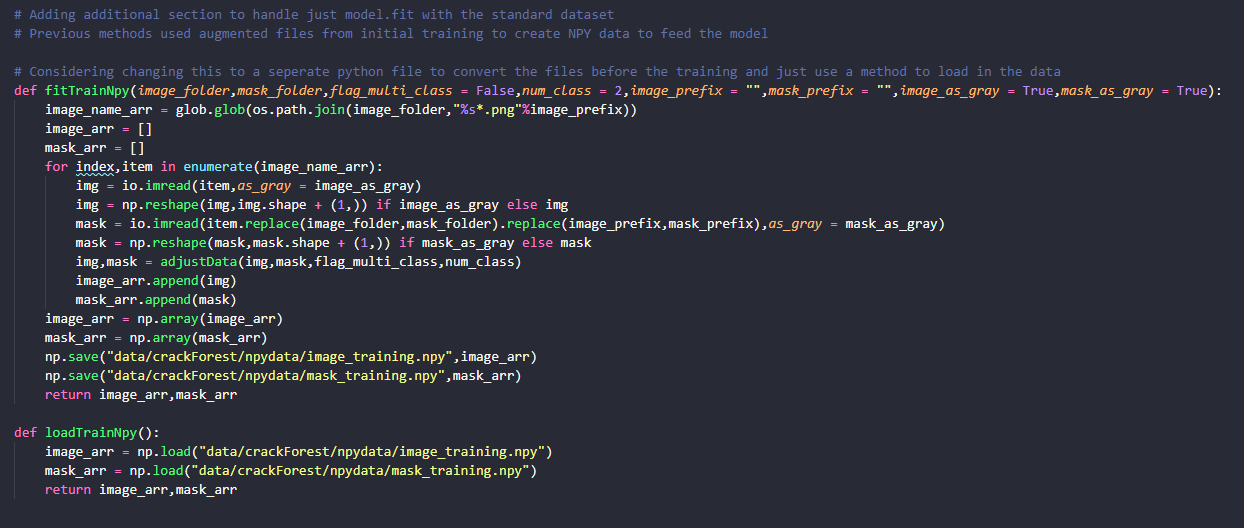


## Appendix C – Data.py of Neural Network









## Appendix D – Activation Function Logs

Smaller sections have been included in this section; the full logs can be found at the Full Project Upload link hosted on Google Drive. 