# Ryan Madigan – Final Year Project

# Using image recognition techniques to

# simulate attacks in Clash of Clans

# Abstract

Image recognition, also known as computer vision, is a field of study in computer science. The aim of image recognition is to determine if the image data contains some sort of object or feature within. The first thing that may come to mind when implementing image recognition would most likely be A.I. However, image recognition has been around for quite some time. Before the use of A.I, algorithms that detected shapes and other things were implemented to carry it out. Image recognition has been used in many things such as facial recognition, visual searching and many more. Image recognition could also be implemented through the use of A.I. For example, through using a convolutional neural network, the A.I could learn to recognise each of the different kind of images.

The purpose of this study is to implement some form of image recognition in the mobile game ‘Clash of Clans’. The idea is to have software recognise the different buildings and structures on a player’s base. Recreate that base and then simulate an attack on the base. The first hurdle is finding a way to implement the image recognition so that it can distinguish between the different buildings.

Table of Contents

[Ryan Madigan – Final Year Project 1](#_Toc57466748)

[Using image recognition techniques to 1](#_Toc57466749)

[simulate attacks in Clash of Clans 1](#_Toc57466750)

[Abstract 2](#_Toc57466751)

[Chapter 1: Introduction 4](#_Toc57466752)

[Chapter 2: Image Recognition 5](#_Toc57466753)

[2.1 Introduction 5](#_Toc57466754)

[2.2 Image Recognition Process 5](#_Toc57466755)

[2.3 Image Recognition with traditional techniques 6](#_Toc57466756)

[Chapter 3: Convolutional Neural Network and Digital Image Processing 11](#_Toc57466757)

[3.1 Introduction 11](#_Toc57466758)

[3.2 Convolutional Neural Networks 11](#_Toc57466759)

[3.3 Digital Image Processing 11](#_Toc57466760)

[Chapter 4: Methodology & Design 13](#_Toc57466761)

[4.1 Key Research Findings 13](#_Toc57466762)

[4.4 Design 13](#_Toc57466763)

[4.5 Prototype 14](#_Toc57466764)

[Chapter 5: Implementation 18](#_Toc57466765)

[Chapter 6: Data Analysis/Synthesis 19](#_Toc57466766)

[Chapter 7: Findings & Conclusions 20](#_Toc57466767)

[References 21](#_Toc57466768)

# Chapter 1: Introduction

# Overview

In modern times, the use of image recognition technology can be found everywhere. Examples of it uses are on phones that may use facial recognition to unlock the phone, visual searches, translating and many more. In short, image recognition looks at the image data and attempts to determine if there are any sort of features or objects within the image. Image recognition was originally implemented through different kinds of algorithms. In modern day, it is more common to use an A.I to implement image recognition, convolutional neural networks are one of the most popular ways of implementing a form of image recognition.

The research chapter can be split into three parts. First, an in depth look at image recognition itself, its origins, and the ways it can be implemented. The following chapter looks at convolutional neural networks and the final research chapter looks at digital image processing.

The methodology chapter gives a view of the sort of technology and approaches that can be taken in the project.

The implementation section of the document gives a detailed report of how all of the work was carried out in each of the sprints, it includes screenshots of some of the code as well as pictures of some of the results that were discovered along the way.

Finally, the last few sections, the conclusion, findings, and analysis give a synopsis of the work that was carried out during the project and gives a summary of the analysis and the findings taken from the work carried out.

# Problem Statement

The research question for the project is:

*Can image recognition be used to simulate attacks in Clash of Clans?*

# 1.3 Aims and Objectives.

**Aim:** Implement a form of image recognition that will take an image of a base from the game Clash of Clans, scan through the image recognising the different components, recreate the base and simulate an attack on it.

**Objectives:**

* Implement some form of image recognition that will be able to recognise the different buildings on a Clash of Clans base.
* Recreate a base after looking at the image of the base.
* Simulate an attack on the base.

# Chapter 2: Image Recognition

## 2.1 Introduction

“Image recognition, in the context of machine vision, is the ability of software to identify objects, places, people, writing and actions in images”. (Rouse, 2017) Image recognition also known as computer vision first appeared back in the 1960s when the field of study began to take shape. The aim of this was to mimic what humans could see and ask the computer what they could see, automating the process of analysing an image. Computers look at the pixels in images and discern different shapes and such based on the shade of the colour. As computers evolved over time, they became much better at doing this. (Pulsar Platform, 2018)It could be considered a subfield of artificial intelligence and machine learning. Refer to Fig2.1 for an overview on the relationship between the three.

Diagram

Description automatically generated

***Fig2.1 Overview of Relationship***(Brownless, 2019)

## 2.2 Image Recognition Process

Image recognition can be used to teach a computer the ability to differentiate between images. In order for a computer to be able to recognise an image it needs to follow a number of steps. First, it extracts the number of characteristics from the image. Images are made up of pixels and each pixel has a RGB value to determine its colour. Next, once each image is converted into its features and characteristics, it needs the known label of the image to train them. Refer to Fig2.2

A picture containing photo, food, different, cat

Description automatically generated

***Fig2.2 Images with the correct label for training***(Dataman, 2018)

The more images used, the better trained the model is. After, the model is trained so that it can categorise the different images. The idea of this is so that an image with features in common from the input will be correctly labelled. Once the model has been trained it can be used to recognise or predict other unknown images. (Dataman, 2018) Refer to Fig2.3 for a diagram showing the recognition process

Chart, diagram

Description automatically generated

***Fig2.3 Image recognition, input to output***(Dataman, 2018)

## 2.3 Image Recognition with traditional techniques

Prior to the image recognition process there are considerations that need to be addressed. The first consideration is the size of the image. The simplest solution here is to simply resize images so that they are the same size. Another possible issue is that the image could be the exact same, but it could be rotated or for example there may be a black border around the image. There is also the possibility of the image having different lighting even if it is the same in every other way.

The first approach would be to take the image and simply go through it comparing it pixel by pixel. Each pixel has its own value, take one from the other and if the value is zero then it is the same otherwise change the similarity variable. Once done, the similarity variable can tell you the percent of similarity. This method is most likely the simplest to implement but may be very slow when comparing many images.

The next method would be keypoint matching. In this approach instead of picking 100 random points it picks 100 important points. As certain parts of an image contain more information than others, such as the edges and corners. What’s great about this approach is that it is possible to avoid problems regarding the image size, rotation or lighting. The cons of this approach are that it does require some knowledge on how to implement it and that it can be quite slow in comparison to others. (Simek, 2009)

**2.2.1 Histogram method:**

In this approach a histogram is built for each image and choose the image with the histogram that is closest to the input images histogram. Each image can have multiple histograms, for example splitting it up by the RGB values, one for each colour and maybe another for the scale or direction of the image. The advantage of this approach is that it can be faster than the previous approach and that it can be much more straightforward to implement. The disadvantage of this is that it can fail on images that are rotated, scaled or discoloured. (Simek, 2009)

**2.2.2 Key points and Decision Trees**

Another approach is done through the use of keypoints and decision trees. This means taking points from an image and then using a decision tree to compare the image. This method is similar to the keypoint matching but is much faster. The pros of using this is that it is not affected by rotation, scale or lighting of the image. However, it is considered the hardest to implement. (Simek, 2009)

Below are some screenshots of some code trying out three different methods for image comparison.

Text, application

Description automatically generated

***Fig2.4 Code Snippet Method 1***(Rosebrock, 2014)

Text

Description automatically generated

***Fig2.5 Code Snippet Method 2***(Rosebrock, 2014)

Graphical user interface, text, application

Description automatically generated

***Fig2.6 Code Snippet Method 2***(Rosebrock, 2014)

Text, letter

Description automatically generated

***Fig2.7 Code Snippet Method 2***(Rosebrock, 2014)

Graphical user interface, text

Description automatically generated

***Fig2.8 Code Snippet Method 3***(Jenn, 2019)

Of course, these are not the only methods that exist, there are many more and these are merely a few examples.

# Chapter 3: Convolutional Neural Network and Digital Image Processing

## 3.1 Introduction

Convolutional neural networks (CNN) were originally introduced in the 1980s by Yann LeCun, a computer science researcher. The early versions of CNNs, called LeNet after LeCun, were able to recognise handwritten digits. The early versions had serious problems as they required large amounts of data and would only work on images of low resolution. (Dickson, 2020) Digital image processing deals with the manipulation of images through the use of a computer. (TutorialsPoint, 2020) It can be used to change the image in many ways without sacrificing the quality of the original image. One of the first applications of digital image processing appeared back in the 1920s when a digital picture was created from a coded tape by a telegraph printer. (Ian M. Smith. *et al.*, 2001)

## 3.2 Convolutional Neural Networks

CNNs are a category of neural networks that have proven to be very useful in areas like image recognition and classification. (Ujjwalkarn, 2016) CNNs have been successful in being able to identify things like faces, objects and many more things apart. CNN’s have three core layers, convolutional layer, pooling layer and the fully connected layer. The convolutional layer is the foundation of a CNN and where a majority of the work is done. It requires three parts, input data, a filter and a feature map. The filter also called a feature detector is moved across the different parts of the image scanning to see if a feature is found. (IBM Cloud Education, 2020)

Table

Description automatically generated

***Fig3.1 A filter being applied to the input image.*** (IBM Cloud Education, 2020)

To summarise, the role of the convolutional layer is to convert the image into numerical values that allow the neural network to interpret and extract relevant patterns. (IBM Cloud Education, 2020)

The next layer is the pooling layer, similar to how the convolutional layer works, it applies a filter to the input. This time however, the filter applies an aggregation function to the values in the area of the image that is being scanned, which populates the output array. There are two types of pooling, max pooling and average pooling. In max pooling, the pixel with the highest value is sent to output array. In average pooling, the filter calculates the average value in the area of the image and sends that value to the output array. (IBM Cloud Education, 2020)

The final main layer of a CNN is the fully connected layer. This layer classifies the features that were extracted in the previous layers along with their filters. Each node in the output layer is directly connected to a node in the previous layer. (IBM Cloud Education, 2020)

## 3.3 Digital Image Processing

Digital image processing deals with image manipulation. The original image is given as an input. The image is then processed using an algorithm and the new image is given as an output. Refer to the image below for an example.

Diagram

Description automatically generated

***Fig3.1 Digital Image Processing Example*** (TutorialsPoint, 2020)

Images can be transformed with filters. Filters can used to do things such as changing brightness, scale, rotation etc. Filtering can be done with two methods. Firstly, through convolution with the use of kernels in the spatial domain. The second method is done through masking specific frequency regions in the frequency domain. Filters can be found in everyday life; filters are used in social media apps such as Instagram and Snapchat which allow users to edit their photos before saving or sending them. The use of digital image processing is not limited to this and has also been used in other sectors such as the medical sector in things like gamma-ray imaging or x-ray imaging. Digital image processing can split up into smaller sections such as image acquisition, image enhancement, image restoration, colour image processing etc. Image enhancement and image restoration are somewhat similar as you are trying to improve the quality of an image in some way. Image acquisition is simply the process of retrieving an image from a source so that it can be passed on for processing. Colour image processing is the analysis, transformation and interpretation of data based on colour. It can have many results such as grayscale conversion which turns an image black and white. (Ian M. Smith. *et al.*, 2001)

# Chapter 4: Methodology & Design

## 4.1 Key Research Findings

Research undertaken into Image recognition indicated that there are many ways to go about it. There are many algorithms in existence that do something along these lines, from recognising shapes in images or things like buildings. In addition, image recognition can be done through the use of A.I or the traditional algorithms like stated above.

Additionally, research completed in CNN’s discovered that the process of how a CNN works. Following this research, it is evident that there are two ways in which to simulate attacks in a gaming environment. One way is to use an Artificial Intelligence, the other option is to use various algorithms in much more straight forward but also brute force sort of method. For the A.I approach, a dataset of all the images would be required to train the CNN on. In Clash of Clans, buildings and such can be upgraded, when they are upgraded, they cannot be considered the same. For example, a level 1 Canon is completely different to a level 2 canon. This means that the CNN would not only need to be able to differ between buildings but also the different levels between them. This means, getting an image of each building at every level available. The process of training a CNN can be quite lengthy depending on your own computer. In the second approach, using various algorithms, this would be more of a trial and error. There are many sorts of algorithms in existence, it is all about to discovering the one that works the best and tweaking it to suit your needs. This approach would most likely take longer due to its factor of trial and error.

**4.2 Research Question**

Can image recognition be used to simulate attacks in the mobile game Clash of Clans?

**4.3 Project Proposal**

There are two approaches that can be taken for this project. Using an A.I or using traditional algorithms and such. For the moment the intent is to move forward with the second method. Knowing this, next is to investigate how to go about it. The first thing to be noted is that each building in the Clash of Clans has an irregular shape so that rules out the option of detecting buildings based on their shape. Upon further inspection it was discovered that the map in Clash of Clans has a chequer pattern on it, with the colour of the tiles varying. By discovering the size of a tile on the map, this information could hopefully be used to pull out specific parts of the map i.e. the different buildings in a player’s base. This idea is currently being trialled in the hopes of getting something working. If this does not work out like hoped, then either it may be necessary to try out the first approach of using an A.I or looking for some other alternative.

## 4.4 Design

This is a formal design which includes best practices for your selected software development methodology. You might cover design documentation including:

**4.5** **Prototype**

|  |  |  |
| --- | --- | --- |
| **Prototype** | **Start Date** | **Finish Date** |
| 1 | 09/11/2020 | ??/12/2020 |

|  |  |  |
| --- | --- | --- |
| **Task Number** | **Details** | **Status** |
| 1 | Take a screenshot in Clash of Clans map without any objects in the way and crop it. | Complete |
| 2 | Create a method to set the colour of the tiles to either green or black to easily differentiate between colours.  Colours are set based on whether they are above or below an average. | Complete |
| 3 | Since some tiles have patches of the opposite colour in them, try out different kinds of filters to try and patch out areas with mixed colours. | Complete |
| 4 | The median blur filter had the best effect on the image, apply it multiple times to get a satisfactory result. | Complete |
| 5 | With the new filtered image, open in paint, draw a line from one point to another. Get the start pixel coordinates and end pixel coordinates. | Complete |
| 6 | Import the image in Jupyter notebooks, before the points can be put through the loop getting the colour value, they need to be put through a line algorithm. Breshenham’s line algorithm is used to ready the points to be used afterwards. | Complete |
| 7 | Create the Breshenham line algorithm in python and test that it works as intended. | Complete |
| 8 | Takes the points gotten from paint, put them into the Breshenham algorithm. This result will be put into the next algorithm which will take the rgb value from each pixel, we only need the g value, and add it into an array. | Complete |
| 9 | Plot the values on a graph. | Complete |
| 10 | Take a few more lines from the image and apply the same process to verify if it works. | Complete |
| 11 | Automate the process of scanning through the lines on the image, there are two lines going in different directions so there will need to be two methods, one for each line. | Complete |
| 12 | The method should only take the best values, calculate the standard deviation to filter which are good values. The lower the standard deviation, the better, also remove any outliers from the best values. | Complete |
| 13 | Filter out any outliers, the mean value of the remaining values will be considered an accurate number for the size of a tile. | Complete |

In the prototype, a cropped image of the map in Clash of Clans is taken. It is put through a method that sets the colour of a pixel to either green or black based on whether it is above or below a calculated average. The processed image is saved for the next part. Next is to apply the median blur filter multiple times to remove some of the patches in tiles that don’t match. Take a start point and an end point, loop through it taking the colour value of each pixel and saving it to an array. Plot the points of the array, to hopefully see a pattern in the values.

Prototype is a work in progress and not complete.

Below are screenshots of code and the image as it changes.

A close up of a green field

Description automatically generated

***Fig4.1 Starting Image of map***

Text

Description automatically generated

***Fig4.2 Method that sets the colour of pixel to either black or green***

A picture containing person, holding, player, green

Description automatically generated

***Fig4.3 Image after applying the above method***

Text

Description automatically generated

***Fig4.4 Code that applies the median blur filter to the image***

***A picture containing light

Description automatically generated***

***Fig4.5 Image after applying the median blur filter***

***Text

Description automatically generated***

***Fig4.6 Breshenham Line Algorithm***

***Graphical user interface, text, application

Description automatically generated***

***Fig4.7 Breshenham Line Algorithm***

***Graphical user interface, text, application

Description automatically generated***

***Fig4.8 Code that calls Breshenham and then loops through result getting the g value***

***Chart, histogram

Description automatically generated***

***Fig4.9 Graph 1***

***Chart, histogram

Description automatically generated***

***Fig4.10 Graph 2***

***Chart, histogram

Description automatically generated***

***Fig4.11 Graph 3***

***Chart, histogram

Description automatically generated***

***Fig4.12 Graph 4***

***Chart, histogram

Description automatically generated***

***Fig4.13 Graph 5***

***Chart, histogram

Description automatically generated***

***Fig4.14 Graph 6***

***Chart, histogram

Description automatically generated***

***Fig4.15 Graph 7***

Text

Description automatically generated

***Fig4.16 Line Scan 1***

***Text

Description automatically generated***

***Fig4.17 Line Scan 2***

***Chart, histogram

Description automatically generated***

***Fig4.18 Example of a “bad graph” because there is no consistency***

***Chart, histogram

Description automatically generated***

***Fig4.19 Example of a “good graph” because the distance between the jumps is much even and consistent***

# Chapter 5: Implementation

The implementation of the project will build upon what was done in the prototype chapter. In the prototype of the project, it was discovered that the rough size of a tile on the map is 30 x 30. It was decided to next look at the borders of the tiles. By knowing the position of the tiles, it is believed that specific tiles could then be then ‘pulled out’ from the map.

### 5.1 Sprint 1 – Tile Border

|  |  |  |  |
| --- | --- | --- | --- |
| Sprint Number | Sprint Name | Start Date | Finish Date |
| 1 | Looking at Tile Border | 25/01/2021 | 08/02/2021 |

|  |  |  |
| --- | --- | --- |
| Task Number | Details | Status |
| 1 | Discuss the work and math’s needed to work with the tile borders on the image with project supervisor. | Complete |
| 2 | Code what was discussed with supervisor | Complete |
| 3 | Look at results and see if it was successful. | Complete |

**Task 1**

The first thing to do was to have a discussion with the project supervisor and where to go from after the results of the prototype. It was decided to begin looking into the tile borders on the map. Once this was decided, there was then further discussion on how to go about this. After this discussion, the next part was to actually go and code what was discussed.

**Diagram, text

Description automatically generated**

***Fig5.1 Summary of work to be done after speaking with supervisor.***

**Task 2**

The objective of this task was to take the information from the previous discussion and put it into code that would accomplish the goal. There were not that many lines of code needed but it did take some time to get it to run without issue. The code would take an x,y coordinate, a slope value and the width/length of the tile. Those three values are added and multiplied then normalised. There is then a loop to go through the line of values and change the pixel colour on the image in the hope of it marking out the tile borders. Below is a screenshot of what the code currently is but may need to be modified in the future.

Graphical user interface, text, application

Description automatically generated

***Fig5.2 Code used in sprint.***

Text

Description automatically generated

***Fig5.3 Code used in sprint.***

**Task 3**

The last thing to complete was looking at the results of the code and seeing if they reflect what was discussed in task 1. After the code was finished and run, it was time to view the results. The code was supposed to put a dot at the corners of the tiles on the image. Below is a picture of the result.

A picture containing light

Description automatically generated

***Fig5.4 Results of the sprint.***

It may be difficult to see due to the size of the dots but there are dots plotted along the image. However, they are not on the corners of the tiles like hoped. This means that the initial calculation of the tile width and length is inaccurate. Knowing this, we will need to go back and make another attempt at calculating the size of the tiles on the game map.

### 5.2 Sprint 2 – Recalculating Tile Size

|  |  |  |  |
| --- | --- | --- | --- |
| Sprint Number | Sprint Name | Start Date | Finish Date |
| 2 | Recalculating the size of the tiles. | 14/02/2021 | 28/02/2021 |

|  |  |  |
| --- | --- | --- |
| Task Number | Details | Status |
| 1 | Discuss results of previous sprint with supervisor and decide where to go next. | Complete |
| 2 | Begin working on the next method for calculating the width of the tile. | Complete |

**Task 1**

After viewing the results of the previous sprint, the next thing to do was to have a discussion with the project supervisor to decide how to proceed based on this newfound information. The project supervisor proposed another method of calculating the tile size.

Chart, histogram

Description automatically generated

***Fig5.5 New way of calculating tile width.***

The first time when calculating the size of the tile, the width of one of the ‘jumps’ like in the graph above was used. However, this time, the entire graph will be looked at. This time, the “length” of the entire graph will be grabbed and that will be divided be the number of “jumps”. So, in the image above it would be divided by 7 (the number of jumps on the graph) to get a, hopefully, more accurate size for the tile for proceeding forward. There was also another way for calculating the tile size. Based on the images above the original value of 30 is not too far off from what the actual value is. This method would be trial and error approach where different values would be tested out to see which would lines up with the tiles. In addition, there is a third approach, which involves storing the pixel coordinates when a “jump” in the value occurs, the “jump” suggests that it is at the edge of tile and using these values in a distance calculation.

**Task 2**

Following on from what was discussed in the previous task, the trial-and-error approach was chosen to be done first as it is easier to carry out. The values were increased, the code was run and then the image was viewed to see the results. After a few tries, the image below came about.

A picture containing light

Description automatically generated

***Fig5.6 Result of trial-and-error approach changing the values in the code.***

As shown above, while not perfectly align, the dots line up well with the corners of the squares on the image. This image came from setting the tile width and tile length variables to 38. After seeing this, the same thing was tried on a larger map. A picture containing grass

Description automatically generated

***Fig5.7 Large map with dots***

As you can see, the dots line up very well on the map. The next step was to try and get a value for tile size through the means of an actual calculation. It was decided to use one of the methods mentioned above, storing the “jump” coordinates. There was a method in the code that already looked at this, so it just needed to be modified so that it would store the coordinates of the jump points.

Text

Description automatically generated

***Fig5.7 Method has been modified to store the jump points.***

The code was run on one of the slope values that was gotten earlier and when the method is called, it displays results like in the image below.

A picture containing text

Description automatically generated

***Fig5.8 Method results.***

Using these values like shown above, the 1st point and the last point were taken and put into the distance formula which can calculate the distance between two points. The formula is as follows: ***√ (Y2 – Y1)2 + (X2 – X1)2***

The values are put into the formula and that answer is then divided by the number of jumps in the graph, in this case it was seven. The final value was 39.58820127241215 which was not too far off the trial value of 38. This value was put into code and the image below was the result.

A picture containing grass

Description automatically generated

***Fig5.8 Results of calculation being used.***

Unfortunately, the ellipses are a bit off the corners of the tiles. The same calculation was performed on a different slope and the value calculated from those points was approximately 49 which is even more off than the first one. After discussing the results with the supervisor, it was decided it would be best to move on for the sake of time.

### 5.3 Sprint 3 – Finding appropriate tile value through trial and error.

|  |  |  |  |
| --- | --- | --- | --- |
| Sprint Number | Sprint Name | Start Date | Finish Date |
| 3 | Finding appropriate tile value through trial and error | 1/03/2021 | 15/03/2021 |

|  |  |  |
| --- | --- | --- |
| Task Number | Details | Status |
| 1 | Continue with trying to calculate an accurate size for the tiles on the map. | Complete |
| 2 | Discuss the results of task one and decide what should be done next to implement image recognition. | Complete |

**Task 1**

As mentioned before, a value of 38 for the tiles worked well when drawing in the corners of the tiles. However, after increasing the range of which the ellipses were drawn on the map, they gradually started to move away from the corners which is not good. Below is a screenshot showing it.

A picture containing grass

Description automatically generated

***Fig5.9 Result using a tile value of 38 and increasing the size of the loop***

Examining the ellipses towards the end, they start move off a bit so the next step is to trial and error different values that may work better and increase the size of the loop so that it covers the entire map. After a bit of trial and error, values that covered the entire map and good size for the tiles resulted in the image below.

A picture containing grass

Description automatically generated

***Fig5.10 Loop that covers map and that somewhat line up with tiles.***

**Task 2**

With the previous task complete, the next task was the figure out where to go next. After speaking with the project supervisor, it was decided to now try and ‘pull out’ specific parts of the map. Below is an example of what is intended to be done.



***Fig5.11 Highlighting a specific part of the map and storing the information.***

### 5.4 Sprint 4 – Trying to ‘pull out’ parts of the map.

|  |  |  |  |
| --- | --- | --- | --- |
| Sprint Number | Sprint Name | Start Date | Finish Date |
| 4 | Pulling out parts of the map | 17/03/2021 | 31/03/2021 |

|  |  |  |
| --- | --- | --- |
| Task Number | Details | Status |
| 1 | Begin working on a way to take out specific parts of the map | Complete |
| 2 | Research ways to compare images. | Complete |

**Task 1**

The first thing to do is to chance the image that is being worked on. An image that contains different buildings and such on the map was brought into the project to use for this part. After discussing the situation with the project supervisor, the image below was the initial idea for attempting this.

Text

Description automatically generated

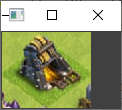
***Fig5.12 Possible way to pull out specific parts of the map.***

After spending some time working on the work in the image above, unfortunately it was not yielding any results. While looking for a way to improve upon the work that was being done. Another rather simple approach was discovered. It works by cropping part of the image, you give it the coordinates of the section you want, and it pulls it out. Below is the few lines of code that do this as well the result.

Text

Description automatically generated

***Fig5.13 The snippet of code that crops a selection of the image.***



***Fig5.14 The result of cropping the image.***

**Task 2**

After finding a way to take out specific parts of the image, the next step was to try some image comparison. Back at the beginning of the project, research was done on way to compare two images. Some of the following approaches were using mean squared error, the lower the value returned the more similar the images are. The next method is structural similarity index, a value returned that is less than one means that the images are different. The third method is the histogram method, it returns the root mean square value of the two images histograms. The final approach is a pixel-by-pixel comparison of the two images, it will return a value, the closer it is to 0 the more alike the images are. The next task would be to attempt to implement each of these methods and see if they are a viable option.

### 5.5 Sprint 5 – Testing out various methods of image comparison.

|  |  |  |  |
| --- | --- | --- | --- |
| Sprint Number | Sprint Name | Start Date | Finish Date |
| 5 | Pulling out parts of the map | 02/04/2021 | 16/04/2021 |

|  |  |  |
| --- | --- | --- |
| Task Number | Details | Status |
| 1 | Attempt to implement and test the mean squared error method. | Complete |
| 2 | Attempt to implement and test the structured similarity index method. | Complete |
| 3 | Attempt to implement and test the histogram method. | Complete |
| 4 | Attempt to implement and test pixel by pixel comparison method. | Complete |

**Task 1**

The first method to test out was the mean squared error. This method takes the images and will return the sum of the squared difference between the two. The images must be the same dimension in order for it to work. Below are the two images that were used for testing it out as well as the code.

Graphical user interface

Description automatically generated with low confidence

***Fig5.15 This is the cropped image pulled from a larger image of base.***

***A picture containing dark

Description automatically generated***

***Fig5.16 This is the template image to compare against.***

**Text, letter

Description automatically generated**

***Fig5.17 Code for Mean Square Error and result*** (Rosebrock, 2014)

The two images used in the code above were the same but the result it gave was 3727.933 is way off. The result should be a lot smaller for the images to be considered similar. Unfortunately, it does not appear that this method will be a viable solution.

**Task 2**

The next method to test was the structural similarity index (SSIM). For this method both images are converted to grayscale and the structural similarity is computed. If the value returned is less than 1 then the images are considered different. For testing it, the same two images from before were used. Below is the code that was used for this and the result.

Text, letter

Description automatically generated

***Fig5.18 Code for Structural Similarity Index and result*** (Rosebrock, 2014)

The result it gave was -0.027. Considering that both images were the same this does not look promising.

**Task 3**

The third method to test was the histogram method. It compares the images histograms and calculates the root mean square value. The lower the value returned the more similar the images are. Below is the code for the method. Unfortunately, the code did not work on the images being used and were unable to get a result. This method is not suitable for what is needed.

A picture containing graphical user interface

Description automatically generated

***Fig5.19 Code for histogram method*** (AutomatedTester, 2009)

**Task 4**

The final method found was a pixel-by-pixel comparison to see if the images were similar. The images are read using the Image library and are compared through the use of a loop and a method from the Image library. It returns a numeric value at the end, the lower the value is the more similar the images are. Below is a screenshot of the code and the result.

Text

Description automatically generated

***Fig5.20 Code for pixel-by-pixel comparison and result*** (AutomatedTester, 2009)

The result from this 2,402,819. Considering that the two images were of the same object this does not look promising either. The test was run again, using a completely different image to compare against. Below is the result.

Graphical user interface, application

Description automatically generated

The result is a bit higher than comparing with images that are the same, but the difference is not significant enough.

# Chapter 6: Data Analysis/Synthesis

Explain the data that you’ve collected.

If your data is not what was predicted, then conduct another literature review and see if other researchers obtained results similar to yours.

Obtaining an unexpected result does not lower your grade! Simply stand-over your findings.

Discuss other approaches with your supervisors.

(not required prior to Christmas)

# Chapter 7: Findings & Conclusions

# References

AutomatedTester (2009) *Compare two images the python/linux way - Stack Overflow*. Available at: https://stackoverflow.com/questions/1927660/compare-two-images-the-python-linux-way (Accessed: 6 April 2021).

Brownless, J. (2019) *A Gentle Introduction to Computer Vision*. Available at: https://machinelearningmastery.com/what-is-computer-vision/ (Accessed: 25 October 2020).

Dataman, D. (2018) *What Is Image Recognition?. The first question you may have is what… | by Dr. Dataman | Towards Data Science*. Available at: https://towardsdatascience.com/module-6-image-recognition-for-insurance-claim-handling-part-i-a338d16c9de0 (Accessed: 25 October 2020).

Dickson, B. (2020) *What are convolutional neural networks (CNN)? – TechTalks*. Available at: https://bdtechtalks.com/2020/01/06/convolutional-neural-networks-cnn-convnets/ (Accessed: 28 October 2020).

Ian M. Smith. *et al.* (2001) *Digital Image Processing*, *Dairy Science & Technology, CRC Taylor & Francis Group*.

IBM Cloud Education (2020) *What are Convolutional Neural Networks? | IBM*. Available at: https://www.ibm.com/cloud/learn/convolutional-neural-networks (Accessed: 15 December 2020).

Jenn, J. (2019) *How to compare two images and display the differences using Python (Hint: Pillow) - YouTube*. Available at: https://www.youtube.com/watch?v=fUfvBnREBFc (Accessed: 5 November 2020).

Pulsar Platform (2018) *A history of computer vision & how it lead to ‘vertical AI’ image recognition*. Available at: https://www.pulsarplatform.com/blog/2018/brief-history-computer-vision-vertical-ai-image-recognition/ (Accessed: 1 December 2020).

Rosebrock, A. (2014) *How-To: Python Compare Two Images - PyImageSearch*. Available at: https://www.pyimagesearch.com/2014/09/15/python-compare-two-images/ (Accessed: 5 November 2020).

Rouse, M. (2017) *What is image recognition? - Definition from WhatIs.com*. Available at: https://searchenterpriseai.techtarget.com/definition/image-recognition (Accessed: 25 October 2020).

Simek, K. (2009) *Image comparison - fast algorithm - Stack Overflow*. Available at: https://stackoverflow.com/questions/843972/image-comparison-fast-algorithm (Accessed: 5 November 2020).

TutorialsPoint (2020) *Digital Image Processing - Tutorialspoint*. Available at: https://www.tutorialspoint.com/dip/index.htm (Accessed: 12 November 2020).

Ujjwalkarn (2016) *An Intuitive Explanation of Convolutional Neural Networks – the data science blog*. Available at: https://ujjwalkarn.me/2016/08/11/intuitive-explanation-convnets/ (Accessed: 28 October 2020).