Autonomous and Real Time Rock Image Classification using Convolutional Neural Networks

Alexis David Pascual
The University of Western Ontario

Supervisor
McIsaac, Ken
The University of Western Ontario Co-Supervisor
Osinski, Gordon
The University of Western Ontario

Graduate Program in Electrical and Computer Engineering
A thesis submitted in partial fulfillment of the requirements for the degree in Master of Engineering Science
© Alexis David Pascual 2019

Follow this and additional works at: https://ir.lib.uwo.ca/etd

Recommended Citation
https://ir.lib.uwo.ca/etd/6059

This Dissertation/Thesis is brought to you for free and open access by Scholarship@Western. It has been accepted for inclusion in Electronic Thesis and Dissertation Repository by an authorized administrator of Scholarship@Western. For more information, please contact wlswadmin@uwo.ca.
Abstract

Autonomous image recognition has numerous potential applications in the field of planetary science and geology. For instance, having the ability to classify images of rocks would allow geologists to have immediate feedback without having to bring back samples to the laboratory. Also, planetary rovers could classify rocks in remote places and even in other planets without needing human intervention. In 2017, Shu et. al. used a Support Vector Machine (SVM) classification algorithm to classify 9 different types of rock images using a with the image features extracted autonomously. Through this method, they achieved a test accuracy of 96.71%. Within the last few years, Convolutional Neural Networks (CNNs) have been shown to be perform better than other algorithms in classifying images of everyday objects. In light of this development, this thesis demonstrates the use of CNNs to classify the same set of rock images. With the addition of dataset augmentation, a 3-layer CNN is shown to have a significant improvement over Shu et. al.’s results, achieving an average accuracy of 99.60% across 10 trials on the test set. Multiple CNN operations with similar output shapes have been designed and appended to an existing architecture to expand hyperparameter considerations. These Combinational Fully Connected Neural Networks achieves an accuracy of 99.36% on the test set. The resulting models are also shown to be lightweight enough that they can be deployed on a mobile device. To tackle a more interesting and practical problem, CNNs have also been designed to classify natural scene images of rocks, an inherently more complex dataset. The task has been simplified into a binary classification problem where the images are classified into breccia and non-breccia. This thesis shows that a Combinational Fully Connected Neural Network achieves an accuracy of 93.50%, better than a 5-layer CNN, which achieves 89.43%.

Keywords: Computer Vision, Machine Learning, Convolutional Neural Networks, Geology, Planetary Science, Planetary Exploration
# Contents

Abstract ........................................... i

List of Figures ..................................... vi

List of Tables ....................................... ix

List of Appendices .................................. x

1 Introduction ....................................... 1
   1.1 Motivation ..................................... 1
   1.2 Tasks and Thesis Contribution ............... 3
   1.3 Thesis Outline ................................ 3

2 Review of Related Literature .................... 4
   2.1 Images and Image Classification .............. 4
      2.1.1 An introduction to images .............. 4
      2.1.2 Image Feature Representations ......... 7
         First-Order Statistics ...................... 8
         Second-Order Statistics from
         Gray Level Co-occurrence Matrices .... 8
   2.2 Training a Classifier ........................ 11
   2.3 Classification Algorithms .................... 12
      2.3.1 Unsupervised learning .................. 12
         K-means ................................... 12
         Anomaly Detection ......................... 13
      2.3.2 Supervised learning ...................... 14
         Logistic Regression ....................... 15
Support Vector Machines ............................................. 17
K-Nearest Neighbours .................................................... 18
Decision Trees ............................................................... 18
Artificial Neural Networks .............................................. 19
2.3.3 Image Classification Metrics .................................... 23
2.4 Rock Image Classification ........................................... 25
2.5 Convolutional Neural Networks .................................... 27
  2.5.1 AlexNet and the Rise of ConvNets .............................. 27
  2.5.2 Defining Convolutional Neural Networks ...................... 29
    Convolutional Layer ................................................. 29
    Pooling Layer ....................................................... 32
    Fully Connected Layers .......................................... 33
2.6 ConvNet Architectures .............................................. 34
  AlexNet ................................................................. 34
  VGG ................................................................. 35
  ResNet .............................................................. 37
  Inception, InceptionV2, and InceptionV3 ......................... 38
  DenseNet ............................................................ 43
  MobileNet .................................................................. 44
2.7 Transfer Learning ..................................................... 44
2.8 Implementation ....................................................... 46
3 Rock Image Classification with Convolutional Neural Networks 48
  3.1 Related Work ......................................................... 48
    Manual Feature Selection .......................................... 49
    Unsupervised Feature Selection ................................... 50
    Shu et. al.’s Results ............................................... 51
  3.2 Applying a Convolutional Neural Network on the Dataset .... 51
    3.2.1 Data Augmentation ............................................. 52
    3.2.2 Transfer Learning .............................................. 53
List of Figures

2.1 Image Recognition is being able to tell that this is an image of a dog and Object Detection is pointing out where in the image is the dog. Image courtesy of Tamu and Dr. Hanif Ladak ........................................... 5

2.2 Color Map. Image reproduced from [8].......................................................... 5

2.3 Image broken down to individual RGB channels............................................ 6

2.4 Histogram of a sample image................................................................. 7

2.5 Image example for GLCM creation. Image referenced from [15] ............. 9

2.6 GLCM reference table. (i,j) is the number of times pixel values i and j have been neighbours with each other in one particular direction.................... 9

2.7 GLCM values for a given image in Figure 2.5........................................... 10

2.8 K-means algorithm visualization. a) Unclassified data points. b) Initialization of two centroids denoted by the red and blue ”x”. c) The data points are labelled according to their proximity with the centroid. d) The centroids are adjusted to reflect the average of all points belonging to its own class label. The data points are then re-labelled, based on proximity with the new centroids. e,f) The process is repeated until no new data points are re-labelled to a different class. Image taken from [16]............................................... 13

2.9 Point anomaly illustration ........................................................................ 14

2.10 Sigmoid Function Graph. Image taken from [19]...................................... 17

2.11 SVM visualization.................................................................................. 17

2.12 KNN Visualization............................................................................... 18

2.13 Decision Tree Visualization taken from [20].......................................... 19

2.14 ANN Neuron illustration...................................................................... 19
2.15 Artificial Neural Network architecture that attempts to classify an input with 4 features to one of three possible classes. ........................................ 21
2.16 Top 5 Error Rate illustration taken with permission from [25]. Actual class labels are shown in larger text, and model predictions are shown with the colored bars indicating confidence. ........................................ 24
2.17 LeNet architecture taken from [38]. ........................................ 28
2.18 AlexNet architecture taken with permission from [25]. .............. 28
2.19 Convolution Demonstration. ........................................ 30
2.20 Edge Detection with Sobel Kernels. Top: horizontal edge detection kernel. Bottom: vertical edge detection kernel ........................................ 32
2.21 Rectified Linear Units activation function graph. .................... 32
2.22 Max Pooling. ........................................ 33
2.23 AlexNet Simplified. ........................................ 34
2.24 AlexNet feature map visualization taken with permission from [43]. ........ 36
2.25 VGG Architecture taken with permission from [48]. .............. 37
2.26 Residual Networks Building Blocks taken with permission from [50]. ........ 38
2.27 34-layer Residual Network Architecture taken with permission from [50]. ........ 39
2.28 Inception Module taken with permission from [52]. .............. 41
2.29 GoogleNet architecture taken with permission from [52]. .......... 42
2.30 Inception V3 Module taken with permission from [56]. .......... 43
2.31 Inception-Resnet Module A taken with permission from [55]. .......... 43
2.32 DenseNet Module taken with permission from [57]. .............. 44
2.33 MobileNet splitting convolution layer into depth-wise convolution and point-wise convolutions. Image taken from [58]. .............. 45

3.1 Sample image for each class in Shu et. al.’s dataset [7]. From top to bottom, left to right: Andesite, Dolostone, Granite, Limestone, Oolitic Limestone, Peridotite, Red Granite, Rhyolite, Volcanic Breccia. ......................... 49
3.2 Unsupervised Feature Learning Diagram. Image taken with permission from [7]. 51
3.3 Data Augmentation Operations. ......................................... 53
## List of Tables

<table>
<thead>
<tr>
<th>Table</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1</td>
<td>Confusion Matrix.</td>
<td>24</td>
</tr>
<tr>
<td>2.2</td>
<td>Hardware used for training.</td>
<td>47</td>
</tr>
<tr>
<td>3.1</td>
<td>Feature groupings used in [7].</td>
<td>50</td>
</tr>
<tr>
<td>3.2</td>
<td>Results from [7].</td>
<td>52</td>
</tr>
<tr>
<td>3.3</td>
<td>Pre-trained networks number of trainable parameters.</td>
<td>55</td>
</tr>
<tr>
<td>3.4</td>
<td>Transfer learning results for each network.</td>
<td>56</td>
</tr>
<tr>
<td>3.5</td>
<td>Custom Networks number of trainable parameters.</td>
<td>58</td>
</tr>
<tr>
<td>3.6</td>
<td>Custom Networks number of trainable parameters.</td>
<td>58</td>
</tr>
<tr>
<td>3.7</td>
<td>Custom Networks Results.</td>
<td>59</td>
</tr>
<tr>
<td>3.8</td>
<td>Custom Networks Results.</td>
<td>60</td>
</tr>
<tr>
<td>3.9</td>
<td>Custom Networks Average Accuracy and Loss across 10 trials.</td>
<td>61</td>
</tr>
<tr>
<td>3.10</td>
<td>Output shape for each layer in the 3-layer network.</td>
<td>62</td>
</tr>
<tr>
<td>3.11</td>
<td>Output shape for each layer in another 3-layer network that follows the same &quot;path&quot; as the previously created 3-layer network.</td>
<td>63</td>
</tr>
<tr>
<td>3.12</td>
<td>Output shape for each layer in another network that follows the same &quot;path&quot; as the previously created 3-layer network.</td>
<td>63</td>
</tr>
<tr>
<td>3.13</td>
<td>Combinational Fully Connected Network Results.</td>
<td>66</td>
</tr>
<tr>
<td>3.14</td>
<td>iPad deployment results.</td>
<td>67</td>
</tr>
<tr>
<td>4.1</td>
<td>Hyperparameters used for Transfer Learning as well as training the custom networks.</td>
<td>78</td>
</tr>
<tr>
<td>4.2</td>
<td>Transfer learning results for each network.</td>
<td>79</td>
</tr>
<tr>
<td>4.4</td>
<td>Custom Networks Results.</td>
<td>79</td>
</tr>
<tr>
<td>4.3</td>
<td>Custom linear network architecture with 5 convolutional layers.</td>
<td>80</td>
</tr>
</tbody>
</table>
List of Appendices

Appendix A Training and Validation Graphs for Chapter 3: 9-Class Rock Dataset . . . . 86
Appendix B Training and Validation Graphs for Chapter 4: Breccia vs Non-Breccia . . . 93
Chapter 1

Introduction

1.1 Motivation

Traditionally, geological exploration and mapping is a tedious and manual process: a geologist goes out in the field, and brings with them a plethora of equipment (compass, GPS, clinometer, camera, protractors, mapping pens and erasers, etc.) and painstakingly records all of these information in a notebook [1]. Subsequently, when there is an interesting rock that needs to be looked at, the geologists takes a sample, and brings it back to the lab for examination. Although nothing could replace the nostalgia of writing with pen and paper, a lot of the grunt work of field work could be automated. This leaves more time for the geologist to do more scientific work.

Seeing the need to simplify the field exploration process, a research group under the supervision of Dr. Gordon Osinski (Dept. Earth Sciences) at the University of Western Ontario is developing an iPad application that combines all of the tasks into one. Geologists could now simply bring just one device to the field and record data as opposed to having to worry about taking with them other equipment. At the end of the exploration, the geologists could then upload all of their data into a database for safekeeping, where they could retrieve data and review their trip on a website.

Among the data that is being uploaded into this database are images of outcrops and samples alongside the geologist’s notes. Often, the notes describe the outcrop and in the image, i.e. its lithology (rock type), orientation, composition. In other words, the geologists are uploading
labelled images of rocks into the database. A question now arises: would it be possible to build a model that would classify images of rocks based on the uploaded images?

Identifying outcrop lithology is paramount in geological mapping and exploration because it gives valuable insight about the area under examination, i.e. it’s geologic history, origin, and nature. Also, making accurate geologic maps has a profound impact on numerous other fields of study. For instance, mining and resource exploration rely on accurate maps to know where a valuable mineral could be feasibly extracted; civil and structural engineering require solid geological information in building dams, roads and buildings; and environmental geosciences depend upon geological maps to predict hazards [1]. Therefore, a system that could automatically classify outcrop lithology would have multiple benefits to the geologist:

- Unknown rocks could be identified without bringing back samples to the laboratory;
- When linked back to the field notebook application, geologists would be able to know where else a particular kind of rock could be found;
- Accurate geological maps could be made with significant ease;
- Geologists could have access to the classifier as learning tool in identifying rocks;

Applications outside of this planet could also be foreseen. Planetary rovers like the Mars Science Laboratory Rover are equipped with a suite of instruments that are collecting a plethora of scientific data [2]. However, communication with the rovers from Earth remains an issue. For one, as the rover travels farther distances in each mission, the amount of data that can be sent to Earth is reduced which potentially results in missed opportunities in terrain in long traverses [3]. As well, many tasks require several steps of human intervention to perform such as approaching a rock outcrop and placing an instrument against it [4]. So with the long, arduous process it takes to send commands and receive data from the rover, a lot of time and data is wasted. For this reason, autonomous targeting systems like the OASIS (Onboard Autonomous Science Investigation System) and AEGIS (Autonomous Exploration for Gathering Increased Science) have been developed to cut back on the data turnaround time, to afford the rover some amount of autonomy and, to allow the rover to collect and send more data [3, 5, 6]. Therefore,
a vision system that could identify interesting rock types without human intervention could further aid in increasing the quality and the quantity of data sent by the rovers.

1.2 Tasks and Thesis Contribution

This research explores the use of Convolutional Neural Networks (CNNs/ConvNets) in classifying rock images and takes a step towards identifying rocks in natural scene images. First, we establish the effectiveness of CNNs in identifying rock images. Then, leveraging the amount of labelled images in the aforementioned database, we use CNNs to create a "Breccia" (a rock that containing angular fragments that are cemented together) and a "Non-Breccia" classifier in natural scene images. Although very challenging, classifying natural scene images is important because the idea is that geologists would bring this system to the field where it would be very difficult to take clean and uniform images of rocks. Therefore, to prove that a system could be created to classify natural scene images of rocks, a simple binary classifier of Breccia/Non-Breccia is first developed. Finally, the rock image classifier would then be deployed in a lightweight and portable device, such as an iPad to create a system that geologists could take into the field.

This system has potential to automatically and reliably classify a large number of images for a short amount of time and energy. However, it is not meant to replace geologists altogether as there is no foreseeable way that a computer could be smarter than trained geologists in classifying rocks. This is merely a tool that could help point geologists in the right direction.

1.3 Thesis Outline

The rest of this thesis is laid out as follows: Chapter 2 provides a literature review on common techniques used in classifying images of everyday objects as well as images of rocks. Also, a background on CNNs will be provided in this chapter. Chapter 3 discusses the work of Shu et. al. [7] and applying a CNN on a dataset of clean images of rocks. Chapter 4 presents the development of the Breccia/Non-Breccia classifier in natural scene images. Finally, chapter 5 concludes the thesis and discusses future work in improving the classifier.
Chapter 2

Review of Related Literature

2.1 Images and Image Classification

2.1.1 An introduction to images

When human beings look at images, the identification of what is in the image and where it is in the image almost comes naturally. For instance, if shown an image of a dog (Figure 2.1), anyone would be able to point out that it is an image of a dog (image classification) and where it exactly is in the image (object detection). However, this process is more complicated for a computer because for one, a CPU is no match for the human brain and two, to a computer an image is simply a 2-dimensional matrix of numbers.

Formally defined, an image is a matrix of numbers containing visual information. Each picture element (pixel) of this matrix represents a screen brightness level on that position of the matrix. A color map defines how to map a pixel value with a particular shade of gray onto the screen. On a gray scale image, this pixel value is commonly referred to as gray level. Figure 2.2 shows a sample image matrix mapped to a screen, in reference to a color map.
2.1. Images and Image Classification

Figure 2.1: Image Recognition is being able to tell that this is an image of a dog and Object Detection is pointing out where in the image is the dog. Image courtesy of Tamu and Dr. Hanif Ladak

Figure 2.2: Color Map. Image reproduced from [8].

In the case of color images, each pixel in the matrix is an array of 3 values each corresponding to color or brightness depending on the color map. The most commonly used color map is the RGB space, where each value in the array represents the brightness level of the red, green, and blue channels respectively.
 Plenty of other color spaces exist, one of which is the Hue, Saturation, and Value (HSV) space where the colors in the image are modelled to be similar to how humans perceive color. Another commonly used color space in digital videos is the YCbCr, where visual information is stored as values of Luminance (Y) and Chrominance (Cr and Cb). The pattern persists, however, in that pixels in color images are usually composed of an array of 3 values.

Even though computers ”see” images as merely matrices, important visual information about the image can be inferred from these matrices [8, 9]. For example, in natural scene images, pixel values in a neighbourhood are highly correlated. Looking at the green channel image in Figure 2.3, a random pixel in the middle of the onion on the top right part of the image is guaranteed to have an adjacent pixel that has a value very close if not identical to it. This maintains a consistent color gradient pointing towards the existence of an object within a localized area in the image.

Another interesting property of images is that edges of objects could be deduced with a sharp difference between adjacent pixels [8, 9]. As an example, in Figure 2.2, a vertical edge could be inferred from a sudden change of 0 to 127 and 0 to 255 in the pixel values in the first and second columns of the image. The same is true in deducing horizontal edges in that there is a sharp difference between the pixel values of the 1st and 2nd rows. Another useful tool in
gathering information about an image is by creating a histogram, which is a plot that shows the number of pixels for each pixel value (Figure 2.4). From this plot, information like pixel distribution, skewness, kurtosis, and many others could be inferred. By gathering and utilizing enough information like these, or features contained within the pixel values in the image, a computer could have an educated guess of what is depicted by the image.

![Histogram of a sample image.](image)

Traditionally, the pipeline for image classification is as follows: first, \( n \) features from the image is extracted, then these \( n \) features are combined to create an \( n \)-dimensional feature vector which represents each image. This \( n \) dimensional feature vector could be thought of as a point in an \( n \) dimensional feature space. Therefore, each image in the dataset exists as a point in this \( n \) dimensional feature space.

Features with which to represent images are not only limited to edges and neighbourhood pixel values, however, and research is being actively done on figuring out the best features to use in representing an image [10–13]. Some of the most popular feature representations used in computer vision are highlighted below.

### 2.1.2 Image Feature Representations

One of the more intuitive ways to represent an image is by looking at the raw pixel values in the images themselves. After all, the pixels themselves contain information about color or brightness. Information garnered from raw pixel values has been dubbed first-order statistics and these features have been known to describe the texture of an image well [14].
First-Order Statistics

- Mean (average gray level intensity within the image)
  \[ \bar{I} = \frac{1}{N} \sum_{i=1}^{N} I(i) \]  

- Median (middle gray level value when the gray levels are sorted)
  \[ I_{\text{median}} = \frac{n + 1}{2} \text{th} \]  

- Standard Deviation (amount of variation between the gray levels)
  \[ I_{sd} = \sqrt{\frac{1}{N-1} \sum_{i=1}^{N} (I(i) - \bar{I})^2} \]  

- Skewness (measures of the asymmetry of the distribution of the histogram)
  \[ \text{skewness} = \frac{\frac{1}{N} \sum_{i=1}^{N} (I(i) - \bar{I})^3}{\left( \sqrt{\frac{1}{N} \sum_{i=1}^{N} (I(i) - \bar{I})^2} \right)^3} \]  

- Kurtosis (measures the sharpness of the peak of the distribution of the histogram)
  \[ \text{kurtosis} = \frac{\frac{1}{N} \sum_{i=1}^{N} (I(i) - \bar{I})^4}{\left( \sqrt{\frac{1}{N} \sum_{i=1}^{N} (I(i) - \bar{I})^2} \right)^4} \]  

where:
- \( I \) = Image matrix
- \( N \) = Total number of pixels in the image

Second-Order Statistics from Gray Level Co-occurrence Matrices

On the other hand, Haralick et al. [15] developed another way of looking at the texture of images by first creating a Gray Level Co-occurrence Matrix (GLCM) and then extracting information from this matrix. The concept is based on the assumption that the texture of the image can be gleaned from the relationship of the gray levels have with each other within the image. This again stems from the fact that neighboring pixels are highly correlated with each other.
2.1. Images and Image Classification

Take for example a 4x4 image in Figure 2.5 with gray levels $\in [0, 1, 2, 3]$. Each non-border pixel would have 8 neighbors over all and 2 neighbours each in the following directions: $0^\circ$, $45^\circ$, $90^\circ$, and $135^\circ$ (see Figure 2.5). The GLCM is therefore a matrix which displays the number of times a pixel pair have been neighbours along a direction all through out the image. And since we are counting along four different directions in considering the pixel neighbourhood, there would also be four different GLCM’s. To illustrate further, Figure 2.6 shows how a typical GLCM would look like for the image shown in Figure 2.5. $(0, 0)$ would be the number of times the gray level value 0 had a neighbouring pixel whose value is also 0. $(0, 1)$ would then be the number of times gray level values 0 and 1 were neighbours with each other and so on. Keep in mind that the definition of ”neighbour” is a function of the direction to which the adjacent pixel is located, as well as the distance $d$ which defines the scope of the neighbourhood.

![Figure 2.5: Image example for GLCM creation. Image referenced from [15].](image)

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>0</th>
<th>1</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td></td>
</tr>
</tbody>
</table>

![Figure 2.6: GLCM reference table. (i,j) is the number of times pixel values i and j have been neighbours with each other in one particular direction.](image)
With this, Figure 2.7 shows the actual values of the four matrices given Figure 2.5.

\[
\begin{bmatrix}
4 & 2 & 1 & 0 \\
2 & 4 & 0 & 0 \\
1 & 0 & 6 & 1 \\
0 & 0 & 1 & 2
\end{bmatrix}
\begin{bmatrix}
2 & 1 & 3 & 0 \\
1 & 2 & 1 & 0 \\
3 & 1 & 0 & 2 \\
0 & 0 & 2 & 0
\end{bmatrix}
\]

(a) GLCM for \(0^\circ\). (b) GLCM for \(45^\circ\).

\[
\begin{bmatrix}
6 & 0 & 2 & 0 \\
0 & 4 & 2 & 0 \\
2 & 2 & 2 & 2 \\
0 & 0 & 2 & 2
\end{bmatrix}
\begin{bmatrix}
4 & 1 & 0 & 0 \\
1 & 2 & 2 & 0 \\
0 & 2 & 4 & 1 \\
0 & 0 & 1 & 0
\end{bmatrix}
\]

(c) GLCM for \(90^\circ\). (d) GLCM for \(135^\circ\).

Figure 2.7: GLCM values for a given image in Figure 2.5.

From these matrices, Haralick et. al. defined 14 textural features that can be used to describe the texture of an image. Some of the textural features are outlined below. The complete list can be found at [15].

- Angular Second Momentum (measures the smoothness or uniformity of the image)
  \[
  ASM = \sum_{I_1, I_2} P(I_1, I_2)^2
  \]  
  (2.6)

- Contrast (measures local level variations)
  \[
  Contrast = - \sum_{I_1, I_2} |I_1 - I_2|^2 \log P(I_1, I_2)
  \]  
  (2.7)

- Correlation (measures pixel correlation in two different directions)
  \[
  Correlation = \sum_{I_1, I_2} \frac{(I_1 - \mu_1)(I_2 - \mu_2)P(I_1, I_2)}{\delta_1 \delta_2}
  \]  
  (2.8)

- Inverse Difference Moment (measures the homogeniety of the image)
  \[
  Homogeniety = \sum_{I_1, I_2} \frac{P(I_1, I_2)}{1 + |I_1 - I_2|^2}
  \]  
  (2.9)
2.2. Training a Classifier

The selection of features to represent an image is very important because the classifiers depend on these features to make a decision on whether or not an image belongs to a particular class. Usually, features are chosen such that there is a clear and distinguishable difference between images belonging in different classes. If for example the features chosen do not represent the image well (i.e., there is little variation between features of images belonging in different classes), then the classifiers would fail to be accurate.

After careful selection of the features, the next step is to train a classification algorithm. Generally, each classification algorithm takes an input feature vector, performs a number of operations on them, and then outputs a prediction for which class the input feature belongs. Training a classifier then entails that the algorithm learns to adjust its parameters so that it could output better predictions and differentiate between classes effectively. Some of the commonly used classification algorithms will be discussed in the next section.

With a huge enough database, a collection of the chosen feature vectors is used to create a training set, a validation set, and a test set. Typically, a dataset is split into around 70% training set, and 15% for both the validation and the test set. From the names themselves, the training set is used to train the classifier. The validation set is used to check how the classifier performs during training, which would then indicate if training is sufficient or not. If so, then the model is saved for further testing. Otherwise, training will continue. Finally, the test set is used to see how well the model generalizes into never before seen data.

During the selection of features, it is also important to consider how many features to use. If too many features were selected, this not only increases the computational resources to train a classifier, but it may also lead to overfitting. Overfitting occurs when the classifier achieves
excellent accuracy on the training set but fails to generalize on the validation or test set. In other words, the classifier ”memorized” the distinguishing features on the training set but fails on never before seen data.

2.3 Classification Algorithms

The following section outlines the different classification algorithms commonly used in image classification. Take note however, that these algorithms are common across classification problems, not just image classification.

As was alluded to in previous sections, the training set is fed into a classification algorithm that takes in an input feature vector and outputs a prediction for which class that particular feature vector belongs to. Often, this output is a probability that each input belongs to class \( y_i \).

There are 2 types of classification algorithms which will be described below.

2.3.1 Unsupervised learning

Unsupervised learning is an umbrella term for algorithms which make predictions without looking at the class labels of each feature vector. Because of this, unsupervised learning algorithms do not have prior knowledge of how many classes there are in each dataset. These types of algorithms are more powerful, however, in that they can usually generalize better with completely new input. At the same time, these algorithms can point towards similarities and differences among objects beyond their class labels. Although these algorithms would not be explicitly used in this research, a brief spiel for unsupervised learning algorithm is given for a potential future application.

K-means

One of the most common unsupervised learning algorithms is the the K-means algorithm. This algorithm attempts to cluster the dataset into \( K \) predetermined classes regardless of whether there are actually \( K \) classes in the dataset or not. This is done by first randomly selecting \( K \) centroids in the feature space and then one by one assigning each training sample to class \( k_i \).
2.3. **Classification Algorithms**

whichever centroid is the closest. Centroid $k_i$ is then recalculated, taking the average of all the training samples belonging in that centroid. As more and more training samples are assigned to each centroid, the more precise each centroid would be in approximating each cluster. Figure 2.8 illustrates how the K-means algorithm works.

![K-means algorithm visualization](image)

Figure 2.8: K-means algorithm visualization. a) Unclassified data points. b) Initialization of two centroids denoted by the red and blue ”x”. c) The data points are labelled according to their proximity with the centroid. d) The centroids are adjusted to reflect the average of all points belonging to its own class label. The data points are then re-labelled, based on proximity with the new centroids. e,f) The process is repeated until no new data points are re-labelled to a different class. Image taken from [16].

**Anomaly Detection**

An anomaly can be defined as an instance or an object that does not conform to what is deemed ”normal” [17]. In terms of finance and credit cards, detecting anomalies could be seen as detecting fraudulent purchases. In the same context, a point anomaly [18] is one instance of fraud where a purchase has been made in a country different from where the owner is currently at. A contextual anomaly [18] on the other hand is when the user goes on a vacation to a different country and uses the credit card there. In this case, the context invalidates that instance as being anomalous.
2.3.2 Supervised learning

In training a supervised classification algorithm, the class labels for each feature vector is known to the classifier and it uses these class labels to compare its prediction to the actual class label of each feature vector. Keep in mind that since we are dealing with classification problems, the output of the algorithms takes on discrete values which pertain to the object’s class label. And in some cases, the output of the algorithms is the probability that an object belongs to a particular class. The difference between the output of the algorithm and the actual class label is usually represented using a Cost Function.

In learning how to predict correctly, the goal of training classification algorithms is to minimize this cost function. Since the output of the classifiers is often a probability, minimizing the cost function entails that the classifiers output a probability as close to 1 as possible for the correct class and as close to 0 as possible for incorrect classes. Say for example we have a dataset with 5 classes, \( y \in \{\text{cat, dog, goat, sheep, fish}\} \). When given an image of a dog, we want the output predictions of the algorithm be \([0, 1, 0, 0, 0]\) where the algorithm is absolutely sure that the image is that of a dog, and not of anything else. Otherwise, a predictions of say \([0.4, 0.3, 0.2, 0, 0.1]\) means that the algorithm is unsure about the class label of the image.

In general, the process is that upon making a prediction, the cost function is calculated to see how far off the prediction is from the true value. The classifier then adjusts its parameters in such a way that wrong predictions are penalized and correct predictions are reinforced.

To elaborate further, the most common supervised classification algorithms are outlined below:
2.3. Classification Algorithms

Logistic Regression

Say for example we have a binary classification problem where class labels \( y \in [0, 1] \). Also for this problem, say our feature vector \( \vec{x} \) only has 2 dimensions \( x_1 \) and \( x_2 \). We can define our hypothesis as:

\[
h_\theta(\vec{x}) = g(\theta^T \vec{x}) = \frac{1}{1 + e^{-\theta^T \vec{x}}} \tag{2.11}
\]

\[
h_\theta(\vec{x}) = g(\theta^T \vec{x}) = \frac{1}{1 + e^{-(\theta_1 x_1 + \theta_2 x_2)}} \tag{2.12}
\]

\[
g(z) = \frac{1}{1 + e^{-z}} \tag{2.13}
\]

where: \( g(z) = \) sigmoid function

\( \theta_1, \theta_2 = \) parameters to optimize

A plot of the sigmoid function is shown in Figure 2.10. Notice that \( g(z) \) approaches 1 as \( z \to \infty \), \( g(z) \) approaches 0 as \( z \to -\infty \). Therefore, we can approximate that \( g(z) \) outputs the probability that a feature vector \( \vec{x} \) belongs to class \( y = 1 \). Conversely, the probability that the same feature vector \( \vec{x} \) belongs to class \( y = 0 \) is \( 1 - g(z) \)

Formally:

\[
P(y = 1|\vec{x}; \theta) = h_\theta(\vec{x}) = g(z) \tag{2.14}
\]

\[
P(y = 0|\vec{x}; \theta) = 1 - h_\theta(\vec{x}) = 1 - g(z) \tag{2.15}
\]

\[
\therefore P(y|\vec{x}; \theta) = [h_\theta(\vec{x})]^y [1 - h_\theta(\vec{x})]^{1-y} \tag{2.16}
\]

For all training samples \( m \), the product of all the probabilities shown in equation 2.16 can be thought of as the Likelihood Function where it measures how likely that the input feature vector \( \vec{x} \) belongs to class \( y = 1 \):
\[ L(\theta) = \prod_{i=1}^{m} \left[ h_\theta(x_i) \right]^{y(i)} \left[ 1 - h_\theta(x_i) \right]^{1-y(i)} \] (2.17)

The goal of training this particular classifier therefore is to maximize the likelihood of an input belonging in class \( y = 1 \). Therefore, we need to find the values of \( \theta^T \) such that

\[ g(z) \to 1 \text{ for } y = 1 \text{ and } g(z) \to 0 \text{ for } y = 0 \]

This basically indicates that we want the classifier to be 100% sure in determining whether an input belongs to class \( y = 1 \) or not. One of the ways that this could be done is via Stochastic Gradient Ascent:

\[ \theta := \theta + \alpha \cdot \nabla_\theta J(\theta; \tilde{x}; y) \] (2.18)

where: \( \alpha = \text{learning rate} \)
\[ \nabla_\theta = \text{gradient of the loss function with respect to parameter } \theta_i \]
\[ L(\theta; \tilde{x}; y) = \text{likelihood function parametrized by } \theta \]

This gives a parameter update rule as follows:

\[ \theta_j := \theta_j + \alpha ((y(i) - h_\theta(x^{(i)})))x_j^{(i)} \] (2.19)

The learning rate determines how fast the algorithm learns by scaling the change in the parameters up or down. A lower learning rate means that the parameters would update slower, therefore increasing the time it takes to train the algorithm. However, if the learning rate is too high, the global optimum of the likelihood function may not be reached.
2.3. **Classification Algorithms**

![Sigmoid Function Graph](image)

Figure 2.10: Sigmoid Function Graph. Image taken from [19].

**Support Vector Machines**

When \( m \) samples with \( n \) dimensions are plotted into \( n \) dimensional feature space, the aim of **Support Vector Machines** (SVM) is to find an optimal hyperplane that separates one class from another. The separating hyperplane is optimal in such a way that the closest points to the boundary are at a maximum distance from said boundary. To visualize, five samples of class *cross* and class *circle* with features \( x_1 \) and \( x_2 \) were plotted in Figure 2.11. Even though both of the red lines separates the two classes, it does not do so very well as opposed to the solid line in the middle as this line has the maximum distance between the boundary and the closest points from each class.

![SVM visualization](image)

Figure 2.11: SVM visualization.
K-Nearest Neighbours

K-Nearest Neighbours is a supervised classification algorithm where a training sample is assigned a class via majority vote of the $K$ nearest points in its neighbourhood. In Figure 2.12, say we have an unknown point which is represented by the $X$ in the middle. If we set $K = 3$, meaning we will only consider the 3 nearest neighbours of point $x$, we will assign class green triangle to point $x$. However, if we set $K = 7$, we will assign class red circle to $x$.

![Figure 2.12: KNN Visualization.](image)

A KNN algorithm with $K = 1$ is simply known as the Nearest Neighbour algorithm (NN).

Decision Trees

Finally, a Decision Tree is a classification algorithm where each internal node represents a discriminative decision between features and whose branches represent the outcome of the decision. A class label is then assigned to the leaf node, which is result of the final decisions as illustrated in Figure 2.13. As an example, if we have a classifier for fruits and our features consist of color, taste, and shape, the starting query could be "is color == green", and if so, ask the next question "is taste == sweet". If the answer to the second query is yes, assign label "kiwi" to it. Otherwise if the taste is sour, assign the label green apple to it.
2.3. **Classification Algorithms**

![Decision Tree Visualization](image)

Figure 2.13: Decision Tree Visualization taken from [20].

**Artificial Neural Networks**

An Artificial Neural Network (ANN) is a machine learning algorithm that attempts to mimic biological neural networks. The network is composed of an input layer, followed by hidden or intermediate layers, and finally an output layer. Each layer further consists of *neurons*, or *nodes*.

In the input layer, the nodes are simply each feature in the feature vector. Therefore, the number of nodes on the input layer corresponds to the number of dimensions on the feature vector. Each node on the hidden layer then takes the weighted sum of the output of the previous layer and goes through a *non-linear activation*. Figure 2.14 shows the internal mechanics of a neuron.

![ANN Neuron Illustration](image)

Figure 2.14: ANN Neuron illustration.

Among others, the most common non-linear activation function is the sigmoid function
shown in equation 2.11. Keep in mind that since the previous layer is fully connected to the
current layer, each of the connections from one node to another would have its own corre-
sponding weight. The output of the sigmoid function would then be the input to the next layer,
with the same process repeated again. In addition to the weights of each of the connections, a
bias parameter is added per layer.

Finally, the output layer takes in the output from the last hidden layer and then performs a
different activation depending on the problem at hand. In multi-class classification problems,
this activation function is often the softmax function, a generalization of the binary logistic
regression classifier.

\[
f_j(\theta^T \vec{x}) = \frac{e^{\theta^T \vec{x}_j}}{\sum_k e^{\theta^T \vec{x}_k}}
\]  

(2.20)

where: \( k = \) number of classes

\( \vec{x} = \) input feature vector

The softmax function takes in an input vector \( x \) and outputs a value between 0 and 1 for
each class \( j \). When concatenated, a vector of predictions \( \vec{f} \) is produced for each input \( \vec{x} \).
The values in the vector of predictions sum up to 1. The output of the softmax function can
then be interpreted as the normalized probability that the input feature vector \( \vec{x} \) belongs to a
particular class. Therefore, the nodes on the output layer matches the number of classes in the
classification problem. An illustration is provided in Figure 2.15

The algorithm learns by updating the weights \( \theta \) and biases \( b_i \) which essentially dictates the
contribution of each input in the firing of the neuron. To adjust the weights, we first define a
new loss function suitable for multi-class classification called the Categorical Cross Entropy:

\[
J_i = -f_{y_i} + \log \sum_j e^{f_j}
\]  

(2.21)

where: \( f_{y_i} = \) true class label

\( f_j = \) prediction of the algorithm for class \( j \)
2.3. Classification Algorithms

2.3.1. Artificial Neural Networks

Figure 2.15: Artificial Neural Network architecture that attempts to classify an input with 4 features to one of three possible classes.

From equations 2.21 and 2.20 we get a cost function parametrized by the weights $\theta$:

$$J_i = -f_{y_i} + \log \sum_j e^{\theta^T \vec{x}_j}$$

(2.22)

Take note that this follows the intuition for the loss function explained in the Logistic Regression section where the loss is essentially the difference between the prediction of the algorithm and the true class label of each input. Since the aim is to minimize the cost function, or to make the predictions as close to the true labels as possible, the weights are then adjusted via a gradient descent method or variations thereof. To recap:

$$\theta := \theta - \alpha \cdot \nabla_{\theta} J(\theta; \vec{x}, y)$$

(2.23)

where: $\alpha =$ learning rate

$\nabla_{\theta} =$ gradient of the loss function with respect to parameter $\theta_i$

$J(\theta) =$ loss function parametrized by $\theta$

For the first forward pass, the weights $\theta$ are initialized randomly and then updated using equation 2.23. There are a few options available for when to update the weights. For instance, one can choose to update the weights after one training sample has been fed forward to the network (stochastic gradient descent). On the other hand, one can also choose to update the
weights after a mini-batch of samples have been fed forward through the network \((\text{batch gradient descent})\). Although stochastic gradient descent allows the weights to be updated right away after a single sample, this often leads to huge fluctuations in the weights. Thus, batch gradient descent is preferred by researchers because it takes into account more samples in updating the weights. But at the same time, opting for mini-batches reduces computational complexity as opposed to calculating the loss over the whole training set \([21]\). Once the entirety of the training set has had a chance to adjust the parameters, this is known as an \(\text{epoch}\). Often times, one epoch is not enough to sufficiently train a network. Thus, multiple passes through the training set is required to successfully create an accurate classifier.

There are better alternatives in updating the parameters however, one of which is the Adaptive Subgradient Method (Adagrad) \([22]\). The Adagrad optimizer introduces more variables which adjusts the learning rates for each parameter depending on their past values. Larger updates are done for infrequently appearing parameters and smaller updates are done for frequent parameters. The update rule then becomes:

\[
\theta := \theta - \frac{\alpha}{\sqrt{G_t} + \epsilon} \odot g_t
\]  

(2.24)

where: \(G_t = \text{diagonal matrix where each diagonal element is the sum of the squares of the past gradients with respect to parameter } \theta_i\)

\(g_t = \text{gradient of the loss function with respect to the parameter } \theta_i\)

\(\epsilon = \text{hyperparameter}\)

\(\odot = \text{element-wise matrix multiplication operation}\)

The advantage of using Adagrad is that manually tuning the learning rates for each parameters is no longer needed. Instead, the adjustments of the weights are dictated by their previous values, which is contained in \(G_t\).

A variation of Adagrad called Adadelta also computes adaptive learning rates for each parameter \([23]\). However, instead of taking the sum of the squares of the past gradients, Adadelta computes an exponentially decaying average of past gradients. This counteracts the tendency
of Adagrad to aggressively decrease the learning rate. A further variation of Adadelta has been developed by Kingma et. al. [24] called Adam, where the same concept of adaptive learning rate is kept, but instead the mean and the uncentered variance of past gradients are taken into consideration.

\[ m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t \]

\[ v_t = \beta_2 v_{t-1} + (1 - \beta_1) g_t^2 \]  \hspace{1cm} (2.25)

where: \( m_t \) = estimate of the mean of past gradients (first moment)

\( v_t \) = uncentered variance of past gradients (second moment)

\( \beta_1, \beta_2 \) = decay rates (hyperparameter)

However, \( m_t \) and \( v_t \) are initialized to 0 and the first few forward passes tend to bias their values close to 0 as well especially with low decay rates. To solve this problem, a bias corrected first and second moments are used instead:

\[ \hat{m}_t = \frac{m_t}{1 - \beta_1} \]

\[ \hat{v}_t = \frac{v_t}{1 - \beta_2} \]  \hspace{1cm} (2.26)

Finally, the update rule is computed as follows:

\[ \theta := \theta - \frac{\alpha}{\sqrt{\hat{v}_t} + \epsilon} \hat{m}_t \]  \hspace{1cm} (2.27)

With enough training, \( \theta \) would converge to a point where the loss function is at its minimum, signifying that the model achieves its best predictions with the obtained values for \( \theta \).

### 2.3.3 Image Classification Metrics

One of the most intuitive ways to check the accuracy of the model is to measure how many correct guesses it obtained on the test set. However, a dataset having a huge number of classes calls for a more forgiving metric. For this reason, the top-5 error rate is used to consider the top-5 predictions of the model in determining correct guesses. The top-5 error rate therefore
Figure 2.16: Top 5 Error Rate illustration taken with permission from [25]. Actual class labels are shown in larger text, and model predictions are shown with the colored bars indicating confidence.

is the number of times where the true class label did not appear in the top 5 guesses of the classifier. An illustration is shown in Figure 2.16.

For binary classification, the **Confusion Matrix**, **F-Score**, **Recall**, and **Precision** are all useful metrics in determining model performance. The confusion matrix is a table showing the number of true positives, true negatives, false positives, and false negatives predicted by the model.

<table>
<thead>
<tr>
<th>Predicted Class</th>
<th>Actual class</th>
</tr>
</thead>
<tbody>
<tr>
<td>Positive</td>
<td>Positive</td>
</tr>
<tr>
<td></td>
<td>True Positives (TP)</td>
</tr>
<tr>
<td>Negative</td>
<td>False Negatives (FN)</td>
</tr>
</tbody>
</table>

Table 2.1: Confusion Matrix.

Precision is the ratio of true positive predictions to the total positive predictions outputted by the model. A high precision indicates that the model could correctly predict positive classes and avoids mislabelling negative classes.
2.4 Rock Image Classification

\[
\text{Precision} = \frac{TP}{TP + FP} \tag{2.28}
\]

Recall, on the other hand, is the ratio of true positive predictions to the number of actual positive classes in the dataset. This measures how well the model could capture all positive classes.

\[
\text{Recall} = \frac{TP}{TP + FN} \tag{2.29}
\]

F-Score is a combination of the two:

\[
\text{F-score} = 2 \cdot \frac{\text{precision} \times \text{recall}}{\text{precision} + \text{recall}} \tag{2.30}
\]

F-score is a useful metric in cases where the dataset is severely unbalanced. For example, say we are given a dataset that contains 90% positive cases and 10% negative cases. An ineffective classifier that simply guesses everything as positive would have a ”good” accuracy of 90%. However, the true performance of the classifier would be reflected on the precision and hence, the f-score of the classifier.

2.4 Rock Image Classification

In the field of geology, image classification is not unheard of and has tremendous potential applications. For example, autonomous hyperspectral image classification has huge applications in mapping unexplored surfaces [26, 27]. Also, classifying rock images garner significant attention from researchers. For instance, Harinie et. al tested image classification algorithms on four types of rocks: intrinsic igneous, extrinsic igneous, sedimentary, and metamorphic [28]. It was accomplished by extracting Tamura features [13] from a set of 50 images and then using these as standard features for each class. Subsequent images are then classified by comparing the basis features with the Tamura features of the input image. The image is then assigned to the class with the minimum distance with the basis. Through this method, the authors obtained a classification accuracy of at least 87%. 
As well, Mlynarczuk et. al. classified thin section images of nine different rock samples [29]. First order statistics were used as features for each image. The features were extracted in four different color spaces namely RGB, HSV, YIQ, and CIELAB [30,31] to examine the effect of color spaces in automatically classifying each image. Using these features, the authors found that the Nearest Neighbours and K-Nearest Neighbours algorithms performed best across all color spaces, achieving an accuracy upwards of 96%.

Meanwhile, Shang and Barnes hand crafted 54 different features based on first order statistics and used three different classifiers (SVM, KNN, Decision Trees) to classify rock images based on 14 different textures [32]. However, to help with efficiently choosing which features are more important than others, they employed a reliability based method for feature selection. For each training sample, the reliability of a feature is given by its distance to the same feature from other training samples belonging in the same class. Farther distances suggest that the feature might not be a good indicator of class membership, and can therefore be omitted from the feature set. The features are then ranked based on the reliability and only the top \( m \) reliable features are selected. As well, the researchers used an Information Gain based ranking to rank the features based on how well they separate data points with respect to their underlying class labels. This is done by calculating the entropy of the class before and after taking a feature into account and then measuring the additional information about the class that the feature provides [33]. Again, the top \( m \) features are selected.

Overall, the authors achieved the highest accuracy by using the top 20 features out of the possible 54 with an SVM classifier. In using all 54 features, the authors did not see any significant increase in accuracy for the SVM. However, in both KNN and Decision Tree, the authors obtained the highest accuracy by using 53 and 48 features respectively. Similar results are observed when using Information Gain based ranking.

Ishikawa and Gulick takes a different approach in mineral classification, this time by using Raman Spectra instead of color images [34,35]. The authors gathered Raman data from 13 different minerals and segregated them according to mineral group. A total of 190 spectra each with 765 dimensions were collected. To reduce the number of dimensions on the feature set, the authors used Principal Component Analysis to map the features into a space where the features are most varied. This reduced the total number of dimensions to 3. The authors
then used a Decision Tree and an Artificial Neural Network with 6 nodes in 1 hidden layer to classify the minerals. Overall, the authors obtained an average accuracy of 83% in classifying the samples by mineral group and 73% by classifying individual minerals.

Baykan and Yilmaz [36] also classified thin section images of rocks as well as the percentage of each mineral present in the thin section. The images were taken under both plane polarized and cross polarized light with each pixel being labelled according to its mineral content. The authors used raw pixel values on the RGB and HSV spaces as the features which was then fed into an Artificial Neural Network with 1 hidden layer. The authors obtained an accuracy of 89.53% when using the RGB color space, 87.5% in HSV and 87.45% using both color spaces.

Finally, Singh et. al used a mix of first and second order statistics, Image Features (percentage of most common gray level, number of edge pixels, etc.) and Region Features to classify textures of basaltic rock images into three classes [37]. In total, 27 features were selected. Principal Component Analysis was again used to reduce feature dimensionality. An Artificial Neural Network was also used, having 2 hidden layers and 15 and 20 nodes respectively. The authors obtained an average classification accuracy of 92.22%

2.5 Convolutional Neural Networks

2.5.1 AlexNet and the Rise of ConvNets

The previous work that has been shown has a common thread: manually extract features and build a feature set, try to optimize feature selection, and finally train a classifier. However, recent developments in image classification has a more different approach. The idea is to automate the whole process and have an algorithm learn which features to extract and then train a classifier at the same time. This method was first proposed by LeCun et. al. and has been dubbed Convolutional Neural Network [38]. In their work, the authors developed the LeNet-5 (Figure 2.17), a ConvNet that classifies hand written digits from the MNIST dataset consisting of 60,000 training samples and 10,000 test samples. The method boasts an accuracy of 99.2% on the test set, performing better than SVM and KNN.
More recently, ConvNets have risen to popularity because of the work of Krizhevsky et. al. [25] in achieving unprecedented accuracy in the ImageNet Large Scale Visual Recognition Challenge 2012 dataset [39]. The feat is impressive because the ImageNet dataset is composed of 1000 different classes with a total of 1.2 million images in the training set and 200,000 images in the validation and test set. Also, the images are of different scales and often off-center objects. Obtaining very accurate results on this dataset translates well into classifying real life natural scene images. Because of the success of Krizhevsky et. al., subsequent top submissions to the contest used variations of ConvNets [40–43].

![LeNet architecture](image1.png)

**Figure 2.17:** LeNet architecture taken from [38].

![AlexNet architecture](image2.png)

**Figure 2.18:** AlexNet architecture taken with permission from [25].

What exactly is a Convolutional Neural Network?
2.5.2 Defining Convolutional Neural Networks

Convolutional Neural Networks (ConvNets) have the same structure as the Artificial Neural Network (ANN) with hidden layers and trainable parameters for each node. The main difference is that the trainable parameters are the weights inside the convolutional kernels (also called the filters). There is also the addition of algorithm specific layers like the pooling layer and the fully connected layer, both of which are going to be explained below. Finally, the output layer is similar with the ANN where the output is the probability of a feature set belonging in a particular class.

ConvNets work in the same mechanics as the ANN in such a way that it also attempts to mimic how the brain interprets images. Imagine for example how an infant would differentiate between shapes. One would imagine that that the edges of each objects are one of the main factors in deciding shape. Over time, an infant learns that an object with four edges is a square, and object with no corners is a circle, and so on. Then, in differentiating different objects, combinations of shapes, edges, as well as colors are taken into account. This object has this shape and this color. The face of mommy and daddy have these shapes and their hair is that color. And as children learn more shapes and more objects, the distinguishing features tend to be more complicated.

Convolutional Layer

At the core of ConvNets is the convolution layer. A convolution with an image is simply a weighted sum of the pixels within the window size of the filter called the receptive field. To illustrate, Figure 2.19 shows a $4 \times 4$ sample image convolved with a $3 \times 3$ filter with weights $[1 \ 1 \ 1; \ 1 \ 1 \ 1; \ 1 \ 1 \ 1]$. This simply results to a linear sum of all the pixels in the receptive field of the filter. Then the window moves over to the next central pixel which fits the filter, also known as the stride. In this example, the stride is 1. This process is repeated until the convolutions all through out the image are exhausted, creating a feature map.

Even though what was showing in Figure 2.19 is a simple $4 \times 4 \times 1$ image (only one color channel) and a $3 \times 3 \times 1$ filter, the convolutions would extend all the way through the depth of the input. So for instance, if we have a 3 color channel image, $4 \times 4 \times 3$, the filters would also
be $3 \times 3 \times 3$, matching the depth of the input.

There are a few things to consider with this process. When we convolved the $4 \times 4$ image with a $3 \times 3$ filter, the result is a $2 \times 2$ feature map. Increasing the receptive field size of the filter to $4 \times 4$ would then result into a $1 \times 1$ feature map. In the same way, if the receptive field size of the filter is decreased to $2 \times 2$, the result would also be a bigger feature map, $3 \times 3$ to be precise. However, padding the borders of the original image with zeroes (therefore increasing the number of convolutions throughout the image) could result in an output image with the same dimensions as the original.

The output of each filter would then be stacked on top of each other, creating a 3-dimensional feature map. In general, the output of the convolution layer is a 3-D volume $[H_2 \times W_2 \times K]$ where:

$$H_2 = \frac{H_1 - F + 2P}{S} + 1 \quad (2.31)$$

$$W_2 = \frac{W_1 - F + 2P}{S} + 1 \quad (2.32)$$

Figure 2.19: Convolution Demonstration.
where: $H_1 = \text{Input image height}$

$W_1 = \text{Input image width}$

$K = \text{Number of filters}$

$F = \text{Filter Receptive Field Size}$

$P = \text{Zero padding}$

$S = \text{Strides}$

The number of filters per convolution layer depends on the discretion of the researcher. Even the most common networks employ varying numbers of filters per layer, and there is no hard and fast rule on how many filters there should be per layer.

In this algorithm, the weights of the filters are learned through the same process as in Artificial Neural Networks. A loss function is defined, and the weights of the filters are updated using an optimizer as in equation 2.26.

Convolving images with specific filters create interesting feature maps that would be useful for classification purposes. For instance, convolving an image with a Sobel Kernel produces a feature map of vertical and horizontal edges in the image (Figure 2.20).

Similar to how nodes in ANNs are activated by a non-linear activation function, image convolutions are followed by an activation function as well. The most commonly used activation function is the Rectified Linear Units (ReLU) where:

\[ f(x) = \max(0, x) \]  \hspace{1cm} (2.33)

The activation is simply a threshold at 0. A graph of the activation function is shown in Figure 2.21. Variations of this activation function exists, one of which is the Leaky ReLU which does not impose a hard threshold on 0, but instead lets linearly adjusted negative values pass through. Another variation is ReLU(n) where $n$ is a maximum threshold arbitrarily chosen by the researcher. The choice of activation functions is left for researchers to decide.
Figure 2.20: Edge Detection with Sobel Kernels. Top: horizontal edge detection kernel. Bottom: vertical edge detection kernel.

Figure 2.21: Rectified Linear Units activation function graph.

Pooling Layer

Convolutional layers are often followed by a pooling layer which reduces the dimensionality of the input. The mechanism is similar to the convolution layer where a sliding window propagates through the image. However, the output of the pooling operation is a choice between the maximum pixel value within the window (Max Pooling), or the average of the pixel values (Average Pooling). Going back to Figure 2.19, if the operation was instead Max Pooling, the result would be Figure 2.22. The choice of pooling operation is left to the researcher, however, both pooling methods are plausible [44]. Throughout this research, max pooling is the method...
of choice.

The purpose of the pooling layer is to aggregate responses within a local area of the image. On one hand, this reduces the number of parameters within the network. This provides two benefits: reducing the computational complexity of training the network, and combats overfitting by reducing the feature map. On the other hand, this operation brings a level of invariance to positional changes within the image, meaning the exact position of pixels in the image would matter less while preserving the structure of the whole image. Ultimately, this affords the classifier some leeway in that regardless of where the object is in the image, the classifier would still be able to successfully recognize the image [44].

**Fully Connected Layers**

After a series of convolutions, activations, and pooling, the resulting 3D feature map is then flattened, creating a 1-D vector of size $H \times W \times K$. This essentially creates the input for the hidden layers of an ANN, which comprises the last few layers of the CNN. Again, similar to an ANN, the output layer indicates the probability that the image belongs to a particular class.
2.6 ConvNet Architectures

Over the years, different ConvNet architectures have been built and different intricacies have been placed to improve classification accuracy. The following section outlines the most popular ConvNet architectures in current literature.

AlexNet

As was mentioned earlier, the work of Krizhevsky et. al. [25] jump started the popularity of ConvNets in image classification. In particular, the authors achieved the best accuracy in the 2012 ImageNet Large Scale Visual Recognition Challenge (ILSVRC2012) [39] achieving a top 5 error rate of 15.32%, while the second place holder achieved a significantly worse top 5 error rate of 26.17%. Their architecture composed of 5 convolutional layers activated using the ReLU operation, and 3 pooling layers in between. There are three fully connected layers after the convolutions containing 4096, 4096, and 1000 nodes respectively. The simplified architecture is shown in Figure 2.23.

The Dropout layer before the fully connected layer is one of the major contributions of Krizhevsky’s work. This layer randomly shuts off the output of a hidden node during training. The probability that a node would be shut off is defined by the dropout rate, which is 50% in the case of AlexNet. Essentially, this process changes the architecture of the whole network every time a new input is presented to the network. During testing however, all neurons are activated. The overall effect of the technique is that it removes the reliance of neurons to one another, forcing the network to learn more complex features that would classify an image correctly regardless of whether all neurons are firing or not [45, 46]. As a side effect, the training accuracy may be lower than validation accuracy when all neurons are activated.

![Figure 2.23: AlexNet Simplified.](image)

This network has over 62.3 million parameters to learn. Although that sounds like a daunting task, Krizhevsky et. al. used 2 parallelized GPUs (Graphics Processing Unit) to train the
network. Using GPUs reduces training time significantly compared to using just the CPU. This is because even though the GPU has slower processor speeds than the CPU, it has significantly more processing cores. A comparison of the hardware is shown in the implementation section in chapter 3.

Following the work of Krizhevsky et. al., the following year’s winner in the ImageNet classification contest was achieved by fine tuning AlexNet [47]. Zeiler et. al. first deconstructed AlexNet and visualized the activated feature maps after passing through each convolutional layer in the network. The authors found out that the first layer extracts basic edges and edge orientations, and deeper layers extracts more complex structures that are often class specific (i.e. face of a dog, eyes, flower petals, etc.) [43]. Figure 2.24 shows sample feature map activations in the different layers of AlexNet.

With this, the authors learned that the first two layers had innate problems within their activations. The first layer extracts high frequency and low frequency information, but lacks mid frequency information thus prompting the authors to reduce the receptive field sizes of the filters in the first layer down to 7 × 7. Also, the second layer showed aliasing artefacts or blurred edges, which is indicative of subsampling in the images. To mitigate this, the authors reduced the stride of the first layer down to 2. Finally, the authors achieved a top 5 error rate of 11.7%, a significant improvement from AlexNet’s 15.32% [43].

**VGG**

Inspired by the work Zeiler et. al. in decreasing the receptive field size of the first convolutional layer in AlexNet, Simonyan et. al. decided to push the idea further by using only 3 × 3 convolutions in their network [48, 49]. By decreasing the receptive field size of the filters in each layer, this also allowed the authors to create very deep networks.

Figure 2.25 shows the architectures Simonyan et. al. designed for the deep networks using smaller convolutional filters. The authors tested 6 architectures in total, each with varying depths.

On their own, all the networks performed better than AlexNet with the 11-layer network achieving a top 5 error rate of 10.4% and the 19-layer network achieving 8.0%. Ensembling all the networks won them the 2014 ImageNet classification contest with a top 5 error rate of
Figure 2.24: AlexNet feature map visualization taken with permission from [43].
7.4% while achieving a top 1 error rate of 24.4% [47].

**ResNet**

Work done by He et. al found that in training very deep networks, the training accuracy seems to approach a ceiling and then degrades rapidly [50]. They claim that this is not a problem of overfitting on the training set, but that of unnecessarily adding depth to the network [41, 51]. To solve this problem, the authors introduced Residual Learning, where the output of a layer skips a connection and is appended to the output of a subsequent layer (Figure 2.26).
The authors evaluated their hypothesis in 18, 34, 50, 101, and 152 layer networks using the ImageNet dataset. The architecture of the 34-layer residual network is shown in Figure 2.27. The deeper networks simply appended more ResNet blocks and convolutional layers. The authors achieved the best result in the 152-layer network with a top 5 error rate of 4.49%. Combining the different networks in an ensemble, the authors achieved a maximum of 3.57% in the top 5 error rate, enough to win the ImageNet contest in 2015 [47].

**Inception, InceptionV2, and InceptionV3**

Stemming from previous work about improving classification accuracy with deeper and wider networks [48, 51], it seems that the most intuitive way of achieving an even higher accuracy is to go even deeper and wider [52]. There is a limitation, however. As more and more layers are added to the network, the computational costs for training the classifier increases dramatically. After all, the memory capacity of the GPU is very limited. To address this dilemma, sparse connections in the fully connected layer or even in the convolutional layer must be introduced [53]. Sparse connections reduces computational complexity by zeroing out most of the neurons in the network. However, there is a trade-off: GPUs and CPUs are optimized for dense matrix calculations and using them for sparse matrix operations would have a detrimental effect to training duration. In fact, Liu et. al. explored the idea of Sparse Convolutional Networks and managed to create a network with 90% sparsity. But, even with most of the parameters zeroed out, the actual training times failed to reach the theoretical speed up times because of sparse matrix computational overheads [54].
Figure 2.27: 34-layer Residual Network Architecture taken with permission from [50].
Szegedy et. al. then introduced the Inception Module (Figure 2.28) as an approximation to a sparse structure all the while using dense matrices. The overall effect is deepening the network, and training less parameters at the same time. This is achieved by combining various operations in each block, and concatenating the result of each filter to form a 3-D feature map that has scale variance stemming from the different operations. In the case of the Inception Module, $1 \times 1$, $3 \times 3$, $5 \times 5$ convolutions were contained in a block, as well as $3 \times 3$ max pooling (Figure 2.28a). Zero padding has presumably been added so that all operations have the same output dimensionality.

With adding these operations per block, the number of trainable parameters is sure to increase. To remedy this, the dimensionality of the output volumes are reduced by adding $1 \times 1$ convolutions before the $3 \times 3$ and $5 \times 5$ convolutions (Figure 2.28b). The $1 \times 1$ convolutions operate on individual pixels, but extends through the depth of the 3D volume. As an illustration, if the 3D volume has dimensions $128 \times 128 \times 256$ and $1 \times 1$ convolutions were operated on the volume, the output would be a $128 \times 128 \times K$ volume where $K$ is the number of $1 \times 1$ filters used. As an added benefit, the Inception module follows the intuition that visual information should be processed at various scales and then aggregated so that the next stage can abstract features from the different scales simultaneously [52].

With the Inception module, the authors created GoogleNet which has 9 inception blocks stacked on top of each other including 22 layers with trainable parameters. The architecture of the whole network is shown in Figure 2.29. By ensembling 7 networks, the authors won the 2014 Imagenet classification contest with 6.67% top 5 error rate.

Operating on the same principle, different versions of the Inception module has been proposed by the author to optimize the networks further [55]. In [56], the authors used consecutive $3 \times 3$ convolutions to replace the $5 \times 5$ convolution in Figure 2.30 as well as widening the network further by concatenating the results of operations with the same output size. Thus, using the inception modules as building blocks, InceptionV2 and InceptionV3 was born. Upon training and testing, both networks achieved 21.2% and 18.77% top 1 error rate respectively and both exhibited improved top 5 error rates at 5.6% and 3.58% respectively.

Finally, the authors combined the principles of the Inception module with Residual Networks [50] to create the Inception-Resnet module. The idea is to allow the Inception modules
2.6. ConvNet Architectures

Figure 2.28: Inception Module taken with permission from [52].

(a) Inception module, naïve version

(b) Inception module with dimensionality reduction

to propagate through the input feature map for each block. This combines the best of both worlds in that deep network degradation is adressed using Residual Learning, and computational overhead reduction is tackled by Inception Modules. One of the many modules used by the authors is shown in Figure 2.31. A complete list of the Inception-ResNet modules can be seen in [55].

On its own, an Inception-ResNet network achieved a top 1 error rate of 17.8% and top 5 error rate of 3.7%. Ensembling a ResNet, InceptionV3 and multiple Inception-ResNet networks achieved an even better top 5 error rate of 3.1%.
Figure 2.29: GoogleNet architecture taken with permission from [52].
2.6. ConvNet Architectures

Figure 2.30: Inception V3 Module taken with permission from [56].

DenseNet

With the seeming trend of making convolutional networks deeper while reducing trainable parameters, Huang et. al. proposes a solution similar to that of ResNet but instead of only one skip connection between layers, the DenseNet module has the output of all layers connected to all other subsequent layers [57]. Figure 2.32 illustrates the DenseNet module. Within the module, the output of layer \( x_0 \) is taken as input of the rest of the layers \( x_{1-4} \). Similarly, the output of layer \( x_1 \) is taken as input of the rest of the subsequent layers \( x_{2-4} \) and so on.

Although there are similarities with the Inception Module where multiple skip connections are made between layers, the main difference is that with DenseNet, the layers are narrower, which means less parameters to train. The authors then created four different networks with

Figure 2.31: Inception-Resnet Module A taken with permission from [55].
varying depths and applied DenseNet modules (DenseNet-121, 169, 201, 264). In testing with the ImageNet dataset, the authors obtained a top 5 error rate of 5.2% which is slightly worse than the Inception Ensemble. However, their result is better than a stand-alone ResNet, while keeping computational costs lower.

![DenseNet Module](image)

Figure 2.32: DenseNet Module taken with permission from [57].

**MobileNet**

Considering that mobile phones and embedded systems have way less powerful GPUs than full computers, lightweight networks are needed to actually provide practical uses for image classification algorithms. Because of this, Howard et. al. created MobileNet, a lightweight network that is designed for mobile deployment [58]. To reduce computational complexity, MobileNet splits a traditional convolutional layer into depthwise convolution and 1x1 point wise convolutions:

The trade-off is that the accuracy of this model does not perform well against deep fully convolutional networks. In fact, in the ImageNet dataset, the maximum top-1 error rate that MobileNet achieves is 29.4%, lower than Inception-ResNet by 11.4%.

### 2.7 Transfer Learning

With all these networks designed specifically to extract features on a huge dataset, a lot of research has been done on whether the features extracted by the deep networks could be applied
2.7. Transfer Learning

Figure 2.33: MobileNet splitting convolution layer into depth-wise convolution and point-wise convolutions. Image taken from [58].

to a task that is fairly different from the original one with which the network has been trained [59–63]. The idea is that training the whole network and relearning the filters on a more specific dataset would no longer be necessary. Instead, the weights of the filters from all the convolutional layers would be kept the same. Only the softmax or output layer is left to be trained, matching the output nodes with the target number of classes particular to the task at hand. This is known as transfer learning. The advantage of using this method is that it decreases computational costs associated with training very deep networks. A cluster of GPUs would no longer be a requirement for training, and days or even weeks of training would no longer be necessary.

Transfer learning is not limited to full Convolutional Networks only. In fact, since the flattening the feature maps is a 1-dimensional vector, this could be viewed as the feature vector of an image. This opens up the possibility of using a different algorithm as a final step in a classification problem. For instance, Razavian et. al. demonstrated this very idea in using a pre-trained AlexNet as a feature extraction method and then using an SVM as a classifier for use in a different classification task [62]. In this case, the authors demonstrated the technique on the Pascal VOC dataset and the MIT-67 Indoor Scenes dataset to train the SVM. It has to be
pointed out that while the AlexNet model has been trained on the ImageNet dataset, the SVM has been trained on the dataset of the classification task at hand:

- PascalVOC dataset contains about 100,000 images of 20 classes such as animals (e.g. bird, cat, cow, dog), vehicles (e.g. aeroplane, bicycle, car), and indoor (e.g. chair, dining table, plant) [64].

- MIT-67 Indoor Scenes dataset contains 15,620 images of 67 indoor scenes classes such as public spaces (e.g. inside a bus, library), residential rooms (e.g. nursery room, bedroom), and working places [65].

On the Pascal VOC dataset, the authors obtained a mean average precision of 77.2%. Meanwhile, the authors obtained a mean accuracy of 69.0% for the MIT-67 Indoor Scenes dataset. Both results are better compared to manually extracting the features.

The authors also used the same method for fine grained recognition where the aim is to classify subclasses, in particular bird and flower species.

- Caltech-UCSD Birds 200-2011 dataset contains 11,788 images of 200 bird species [66]

- Oxford 102 flower dataset contains 8,189 images of 102 different flowers found in the UK [67]

They obtained a Mean Average Precision of 61.8% testing in the Caltech-UCSD Birds 200-2011 dataset and 86.8% on the Oxford 102 flower dataset. Again, both of the results were shown to be better than manually selecting features.

2.8 Implementation

For this thesis, the creation of ConvNets has been implemented using Keras with Tensorflow as a backend [68–70]. Keras is a high level Python API that allows researchers to create different kinds of neural networks easily. With this API, creating a simple ConvNet could be accomplished in a few lines of code. Tensorflow is the engine that runs behind Keras, which allows for the ConvNets to be efficiently trained on a GPU.
For deployment, coremltools [71] is used to convert the Keras model into a CoreML model, which is required for deployment in an iPad application. An iPad application has been developed using Xcode and Swift as the programming language.

As for training hardware, the lightweight models were trained using the CPU, while the heavier models were trained using the GPU. The hardware used is shown in Table 2.2.

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>CPU</strong></td>
<td>Ryzen 7 1700X</td>
</tr>
<tr>
<td><strong>GPU</strong></td>
<td>EVGA GTX 1080Ti FTW3</td>
</tr>
<tr>
<td><strong>Memory</strong></td>
<td>16GB</td>
</tr>
</tbody>
</table>

Table 2.2: Hardware used for training.
Chapter 3

Rock Image Classification with Convolutional Neural Networks

In chapter 2, previous research has been shown in classifying rock images using the traditional image classification pipeline. In this chapter, we explore the effectiveness of using Convolutional Neural Networks on rock images. In addition, a section will be dedicated to outlining the process of deploying the models on a mobile application, as well as testing the accuracy of the deployed model. As a baseline, research done by Shu et. al. [7] will be the main point of comparison between traditional methods and ConvNets.

3.1 Related Work

Shu et. al. explored the effects of *manual feature selection* and *unsupervised feature learning* in classifying a dataset of rock images with 9 classes [7]. For the classification algorithm, the authors used Support Vector Machines (SVM). The dataset is composed of approximately 700 images obtained on 9 different classes of rocks. Each class contains approximately 80 images, and each image has a size of $128 \times 128 \times 3$ pixels. The samples were obtained from the Earth Sciences Department of the University of Western Ontario and the images were taken using a digital camera and an optical microscope. The lengths and the widths of the images are between 1-2cm in reality. Figure 3.1 shows sample images for each class in the dataset.
3.1. Related Work

Figure 3.1: Sample image for each class in Shu et. al.’s dataset [7]. From top to bottom, left to right: Andesite, Dolostone, Granite, Limestone, Oolitic Limestone, Peridotite, Red Granite, Rhyolite, Volcanic Breccia.

Manual Feature Selection

The authors used a combination of first order and second order statistics discussed in chapter 2 as feature representations for the images:

- Mean
- Median
- Skewness
- Kurtosis
- Angular Second Moment
- Entropy
- Contrast
- Correlation

These features were obtained in 3 color channels (RGB) for a total of 27 features extracted per image. The authors then created 5 groups of manually selected features to compare the effectiveness of different feature combinations. The groups are shown in Table 3.1.
Unsupervised Feature Selection

Unsupervised feature learning is a method of extracting features from the images without using the class labels of the images in the dataset. With this method, a more general representation of the images may be produced, and it removes the bias from researchers in selecting feature representation. To accomplish this, Shu. et. al first extracted \( n \) random sub-patches of dimensions \( w \times w \times d \) from random images in the dataset. Using the raw pixel values of the sub-patches, each flattened image can be thought of as a 1-D vector in \( \mathbb{R}^N \) space where \( N = w \times w \times d \). Then, the K-means algorithm is used to extract \( K \subset \mathbb{R}^N \) centroids from the random sub-patches. These centroids will represent basis vectors for the images in the dataset.

To extract feature representations for one image, a sliding window with stride \( s \) operates on the image to extract \( X \) sub-patches with the same dimensions \( (w \times w \times d) \). Then each sub-patch \( x_j \) is mapped to a \( \mathbb{R}^K \) vector through:

\[
f(x_j) = \max\{0, \mu - z_i\} \quad \forall K
\]

(3.1)

where: \( x_j \) = Sub-patch feature vector in \( \mathbb{R}^N \)

\[
z_i = \|x_j - \bar{K}_i\|\]

\[
\mu = \text{Average of all } z_i
\]

The operation calculates the distance of each sub-patch with each of the centroid, and then gets the deviation of said distance from the mean of the centroids. If the distance of the feature

<table>
<thead>
<tr>
<th>Group</th>
<th>Features</th>
</tr>
</thead>
<tbody>
<tr>
<td>MF I</td>
<td>All first and second order statistics</td>
</tr>
<tr>
<td>MF II</td>
<td>All first order statistics</td>
</tr>
<tr>
<td>MF III</td>
<td>All second order statistics</td>
</tr>
<tr>
<td>MF IV</td>
<td>Mean, Skewness, Kurtosis, Entropy, Correlation</td>
</tr>
<tr>
<td>MF V</td>
<td>Skewness, Kurtosis, Angular Second Moment, Entropy, Correlation</td>
</tr>
</tbody>
</table>

Table 3.1: Feature groupings used in [7].
vector from the centroid is very large, it implies that the particular centroid is irrelevant, hence
the threshold at 0. Each image is then divided into a quadrant and the sum of the transformed
sub-patches in each quadrant is obtained to create an image feature vector \( \vec{\phi} \in \mathbb{R}^{4\cdot K} \).

A diagram summarizing this method is shown in Figure 3.2

Figure 3.2: Unsupervised Feature Learning Diagram. Image taken with permission from [7].

**Shu et. al.’s Results**

Overall, the authors obtained the highest accuracy in employing Unsupervised Feature Learn-
ing with a Support Vector Machine Classifier at 96.71%. Meanwhile, using both first and
second order statistics obtained an accuracy of 96.24%, which is only slightly lower than un-
supervised feature selection. It is worthwhile to note that using different combinations of man-
ually selected features yielded highly varying results. For instance, using only second order
statistics yielded a low accuracy of 66.20% but using only first order statistics yielded a signif-
ically higher accuracy of 95.77%. Therefore, using an autonomous way of selecting features
would prove to be valuable for both generalizability and classification accuracy.

### 3.2 Applying a Convolutional Neural Network on the Dataset

This section presents the steps undertaken in applying a ConvNet on the dataset presented
in the previous section. First, transfer learning is discussed to see the effectiveness of using
pre-trained networks in extracting features from rock images. Next, custom made ConvNets
are discussed in an attempt to gain better accuracy results. A section is dedicated to outline
### Table 3.2: Results from [7].

<table>
<thead>
<tr>
<th>Features</th>
<th>Test Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>MF I</td>
<td>96.24%</td>
</tr>
<tr>
<td>MF II</td>
<td>95.77%</td>
</tr>
<tr>
<td>MF III</td>
<td>66.20%</td>
</tr>
<tr>
<td>MF IV</td>
<td>92.02%</td>
</tr>
<tr>
<td>MF V</td>
<td>74.18%</td>
</tr>
<tr>
<td>Unsupervised Feature Learning</td>
<td>96.71%</td>
</tr>
</tbody>
</table>

the reasoning behind the design of the architectures. Finally, Combinational Fully Connected Networks (CFCN) are discussed as the final optimization step for the network designs.

#### 3.2.1 Data Augmentation

Before jumping in to classifying rock images with ConvNets, it must be pointed out that ConvNets require a large number of labelled images for it to have a high testing accuracy [44]. For instance, the ImageNet dataset has more than 1.4 million images with 1000 classes. This gives about 1400 images per class. Meanwhile, Shu et. al.’s dataset only contains about 80 images in some classes and even less in others [7]. To make up for the difference, *data augmentation* is employed to increase the number of images in the dataset without having to take more pictures of the same rocks. Data augmentation is accomplished by applying transformations on the images in the training set (e.g. translation, rotation, scaling, blurring) then using the transformed images alongside the original and untransformed images in the training set. In addition to multiplying the number of images in the training set, data augmentation affords the model some form of invariance with regards to how the rock is positioned in an image. After all, regardless of the way any person looks at an image of a rock, the rock in the image would never change.

The intuition also goes back to the definition of an image being a 2-D matrix of numbers. Transforming the images would yield different pixel positions on the matrix, but the transformed matrix still belongs to the same class as the original image, thus artificially creating an image that is different to the computer, but not in reality. Overall, data augmentation is a cheap
way of improving the performance of classifiers. A diagram of some of the transformations are shown in Figure 3.3.

![Data Augmentation Operations](image)

Figure 3.3: Data Augmentation Operations.

For deployment testing purposes, 5 random images from each class were taken out from the original dataset before training. Essentially, these images would not be "seen" by the model at all during training and testing. This will have the same effect of testing the model with a new image of a rock taken with a microscope and digital camera. Therefore, these images will be then used to check if the models that are deployed to an iPad application still achieves the desired accuracy.

### 3.2.2 Transfer Learning

As was mentioned in chapter 2, transfer learning is a technique where ConvNets pre-trained on a broad dataset is applied to a more specific problem set. In total, 9 different networks trained on the ImageNet dataset have been used:

- VGG16
- VGG19
- ResNet50
- MobileNetV2
- InceptionV3
- InceptionResNetv2
- DenseNet121
- DenseNet169
- DenseNet201
The nuances and the architectures of the networks above have been previously discussed in chapter 2. One of the advantages of applying this technique is that creating a rock image classifier would become computationally cheaper. Instead of training a very deep network which could possible take days to finish, training time is reduced to just hours. As well, GPU memory requirements are much less, allowing us to train highly complex networks using just one GPU. Also, filters learned by the pre-trained convolution layers are supposed to be complex enough to extract textures from rocks given that they have been trained to extract features from a more complicated dataset.

**Experimental Setup**

For each of the networks used in transfer learning, only the output layer and the preceding activation has been modified to fit our dataset. As such, the final output is reduced from the original 1000 nodes to 9 nodes. All the preceding filter weights and biases have been frozen, meaning that the weights and biases of each filter are not adjusted during training. This dramatically reduces the number of parameters to train. A list of the pre-trained models are shown in Table 3.3 alongside the original number of trainable parameters and the reduced number of trainable parameters after freezing the intermediate layers.

Each model was trained for 200 epochs with batch size of 16. The Adam optimizer was used, with a learning rate of 0.0001. The original dataset was divided into 70% training set, 15% validation set, and 15% test set. To confirm the accuracy of the results, 10 models were trained with the whole dataset shuffled for each trial, making sure that the models are not all created on the same training set. As well, training loss, validation loss, training accuracy and validation accuracy are recorded per epoch and are plotted into individual graphs at the end of training. This gives us an idea of how well each of the models adjust and learn per epoch as well as possible asymptotes for accuracy and loss. Even if 200 epochs were chosen for training, the best model in terms of lowest validation loss and highest validation accuracy is saved regardless of which epoch the best model was obtained. This ensures that if the best model was achieved in epoch 150, 175, or even 100, that particular model will be saved and further tested with the test set to see how well the model generalizes to never before seen data. The accuracy and loss on the test set is then recorded and reported.
3.2. Applying a Convolutional Neural Network on the Dataset

<table>
<thead>
<tr>
<th>Model</th>
<th>Total Number of Parameters</th>
<th>Total Number of Trainable Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>VGG16</td>
<td>134,297,417</td>
<td>36,873</td>
</tr>
<tr>
<td>VGG19</td>
<td>139,607,113</td>
<td>36,873</td>
</tr>
<tr>
<td>ResNet50</td>
<td>23,606,153</td>
<td>18,441</td>
</tr>
<tr>
<td>MobileNetV2</td>
<td>2,269,513</td>
<td>11,529</td>
</tr>
<tr>
<td>InceptionV3</td>
<td>21,821,225</td>
<td>18,441</td>
</tr>
<tr>
<td>InceptionResNetV2</td>
<td>54,350,569</td>
<td>13,833</td>
</tr>
<tr>
<td>DenseNet121</td>
<td>7,046,729</td>
<td>9,225</td>
</tr>
<tr>
<td>DenseNet169</td>
<td>12,657,865</td>
<td>14,985</td>
</tr>
<tr>
<td>DenseNet201</td>
<td>18,339,273</td>
<td>17,289</td>
</tr>
</tbody>
</table>

Table 3.3: Pre-trained networks number of trainable parameters.

Transfer Learning Results

The training time varied from 20-30 minutes on the relatively shallower networks (VGG16, VGG19, ResNet50, MobileNet) to up to 1-2 hours on the deeper and more complicated networks (Inception, DenseNet). Table 3.4 shows the averages and standard deviations of accuracy and loss on the test set after training for 10 trials. Among all the networks, only the VGG networks achieved a fairly good accuracy of 82.33% and 80.77% for the 16-layer and 19-layer networks respectively. The rest of the networks performed worse, topping at 32.56% accuracy for DenseNet169 and reaching as low as 11.30% for the ResNet50 network. Across all 10 trials, the results are fairly consistent, having low standard deviations for both the accuracy and loss.

By looking at the graphs of accuracy and loss per epoch during training and validation, an insight could be garnered as to what happens during training and how it affects the test set. Figures A.1 to A.3 shows the said graphs. Even though the graphs for only one trial are shown, each graph is a fair representation of the trend of training across all trials.

In figures A.1a to A.1d, the graphs show that per epoch, the networks successfully learn on the training set as indicated by the increasing trend in accuracy and the decreasing trend
in loss in both training and validation sets. This is reflected in the results for the test set, seeing that both VGG networks obtained a fairly high accuracy in that regard. However, this is only true for the VGG networks as the rest of the networks simply overfitted on the training set and failed to even generalize on the validation set. Figures A.1e, A.2a, A.2c, A.2e, A.3a, A.3c, and A.3e show that ResNet50, MobileNetV2, InceptionV3, InceptionResNetV2, and the DenseNet networks displayed an increasing trend training accuracies but showed consistently low validation accuracies. This is a clear indication of the networks simply memorizing the features in the training set and then failing to generalize the features in never before seen data. Again, this is reflected in the results for the test set, as these networks exhibited extremely low accuracies.

Interestingly, the results run contrary to what has been shown previously in literature where one might expect to see fairly accurate results for transfer learning. One of the reasons that transfer learning did not perform up to task is because the filters might have been trained to extract features that are too complicated for rock textures. Keep in mind that the ImageNet dataset contains images from everyday objects at a much bigger scale than Shu et. al.’s dataset.
3.2. Applying a Convolutional Neural Network on the Dataset

Therefore, filters that could differentiate between cars, animals, human beings and inanimate objects could not translate well into objects with much finer differences such as rocks. As well, increasingly complicated and deep networks seem to have a problem with classifying simple datasets. Both the relatively well performing VGG networks are simple, linear networks as opposed to the rest, which are very deep and complicated with some networks reaching as much as 201 layers. Although transfer learning did achieve its goal of having less parameters to train, it appears that decreasing the trainable parameters had a detrimental effect to the accuracy of the models.

3.2.3 Training a Custom Network from Scratch

Seeing that pre-trained models on the ImageNet dataset performed worse that Shu et al.’s results, creating a network from scratch is the next recourse for the problem. With the VGG networks performing well, it seems that simple, linear networks could be an effective design for a network. For this reason, a shallow, simple, and linear network is going to be the focus of the design. Here, we encounter the classic problem of optimizing the architectures. How many layers is enough? And how many filters per layer is enough? What is the ideal receptive field size of the operations? To answer these questions, we employ a step by step process in designing an architecture from the ground up.

Experimental Setup

First, six one-layer networks have been created to see the optimal number of filters that would give consistently high accuracy. These networks contain only one convolution layer followed by a 2x2 pooling layer and is followed by fully connected layer with 128 nodes. Each of the convolution layers for each network then contain 4, 8, 16, 32, 64, and 128 filters each. Table 3.6 shows the number of trainable parameters for each of the network. A dropout layer with a 50% dropout rate, similar to what Krizhevsky et al. implemented, was added between the fully connected layer and the output layer. In keeping the experimental setup consistent, the hyperparameters used in transfer learning will be kept the same as shown in Table 3.5.

Table 3.6 shows the number of trainable parameters per network. This is calculated from the
<table>
<thead>
<tr>
<th>Hyperparameters</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Epochs</td>
<td>200</td>
</tr>
<tr>
<td>Batch Number</td>
<td>16</td>
</tr>
<tr>
<td>Optimizer</td>
<td>Adam</td>
</tr>
<tr>
<td>Learning Rate</td>
<td>0.0001</td>
</tr>
<tr>
<td>Training set</td>
<td>70%</td>
</tr>
<tr>
<td>Validation set</td>
<td>15%</td>
</tr>
<tr>
<td>Test set</td>
<td>15%</td>
</tr>
<tr>
<td>Trials</td>
<td>10</td>
</tr>
</tbody>
</table>

Table 3.5: Custom Networks number of trainable parameters.

Increasing the number of filters per layer increases the number of trainable parameters. This is an important thing to consider in keeping the computational costs low but at the same time having an accurate classifier.

<table>
<thead>
<tr>
<th>Model</th>
<th>Total Number of Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 layer (4 filters)</td>
<td>2,033,529</td>
</tr>
<tr>
<td>1 layer (8 filters)</td>
<td>4,065,769</td>
</tr>
<tr>
<td>1 layer (16 filters)</td>
<td>8,130,249</td>
</tr>
<tr>
<td>1 layer (32 filters)</td>
<td>16,259,209</td>
</tr>
<tr>
<td>1 layer (64 filters)</td>
<td>32,517,129</td>
</tr>
<tr>
<td>1 layer (128 filters)</td>
<td>65,032,969</td>
</tr>
</tbody>
</table>

Table 3.6: Custom Networks number of trainable parameters.

**Results**

Table 3.7 shows that across 10 trials, using 32 filters showed consistent high accuracy results at 93.63%. Increasing the filters to 64 and 128 did not show any improvement on the performance...
of the network and in fact as more filters are added, the accuracy on the test set decreases. At the same time, decreasing the number of filters also yields lower accuracies. Therefore, it seems that 32 filters is the ideal amount that gives the least number of trainable parameters and at the same time shows good accuracy results.

Figures A.4 and A.5 show the loss and accuracy per epoch during training. With the increasing trend for the accuracy per epoch, the graphs show that the network does indeed learn on the training set successfully. As well, looking at the accuracy on the validation set per epoch indicates that the network generalizes it well into never before seen data. The similar trend was seen with the VGG networks earlier, with the accuracy increasing per epoch and the loss decreasing per epoch. And again, these networks showed good results during testing.

<table>
<thead>
<tr>
<th>Model</th>
<th>Average Accuracy</th>
<th>Average Loss</th>
<th>Standard Deviation of Accuracy</th>
<th>Standard Deviation of Loss</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 layer (4 filters)</td>
<td>77.67%</td>
<td>0.5998</td>
<td>0.08608</td>
<td>0.2218</td>
</tr>
<tr>
<td>1 layer (8 filters)</td>
<td>80.33%</td>
<td>0.4972</td>
<td>0.1922</td>
<td>0.4418</td>
</tr>
<tr>
<td>1 layer (16 filters)</td>
<td>90.93%</td>
<td>0.2445</td>
<td>0.0354</td>
<td>0.08086</td>
</tr>
<tr>
<td>1 layer (32 filters)</td>
<td>93.63%</td>
<td>0.1851</td>
<td>0.05178</td>
<td>0.01837</td>
</tr>
<tr>
<td>1 layer (64 filters)</td>
<td>90.80%</td>
<td>0.2499</td>
<td>0.06632</td>
<td>0.1963</td>
</tr>
<tr>
<td>1 layer (128 filters)</td>
<td>89.33%</td>
<td>0.2927</td>
<td>0.02069</td>
<td>0.2860</td>
</tr>
</tbody>
</table>

Table 3.7: Custom Networks Results.
Experimental Setup II

After seeing the results that 1-layer networks give, network architectures with 2 layers and 3 layers were then created. Increasing the number of layers simply means that the convolution and the pooling operations have been repeated according to the number of layers in the network. For the 2-layer network, each of the convolution layer still has 32 filters. However, keeping up with the trend in literature that deeper convolution layers contain more filters, the last convolution layer in the 3-layer network has double the number of filters at 64. The pooling operations were kept the same at 2\times2 and the fully connected layer and dropout layers were kept the same. At the same time, the hyperparameters in Table 3.5 have been used. Table 3.8 shows the number of trainable parameters for both of the networks. Notice that as the networks are deepened, there are are less parameters to train with as much as a 90% reduction going from the 1-layer network to the 3-layer network. This is a result of the pooling layers and the convolution layers repeatedly shrinking the dimensionality of the feature maps.

<table>
<thead>
<tr>
<th>Model</th>
<th>Total Number of Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>2-layer Network</td>
<td>3,697,833</td>
</tr>
<tr>
<td>3-layer Network</td>
<td>1,635,561</td>
</tr>
</tbody>
</table>

Table 3.8: Custom Networks Results.

Results II

Table 3.9 shows the results of training and testing across 10 trials. The best result was obtained with the 3-layer network, averaging more than 99% accuracy on the test set, showing that the model can generalize well. In fact, in 4 out of the 10 trials, the model was able to achieve 100% accuracy. The lowest accuracy obtained by this architecture was at 98.33%. The 2-layer networks performed better than the 1-layer networks but just falls short of the 3-layer network at 97.00% accuracy on the test set. With these results, it has been shown that shallower, simpler, and linear networks could perform very well on Shu et. al.’s dataset. One of the reasons we can posit is that the filters on the custom networks have been trained on a dataset it was specifically
3.2. Applying a Convolutional Neural Network on the Dataset

designed to classify, as opposed to transfer learning, where the filters were trained on a more
general dataset not containing specific types of rocks.

<table>
<thead>
<tr>
<th>Model</th>
<th>Average Accuracy</th>
<th>Average Loss</th>
</tr>
</thead>
<tbody>
<tr>
<td>2-layer Network</td>
<td>97.00%</td>
<td>0.07628</td>
</tr>
<tr>
<td>3-layer Network</td>
<td>99.60%</td>
<td>0.01597</td>
</tr>
</tbody>
</table>

Table 3.9: Custom Networks Average Accuracy and Loss across 10 trials.

As shown in chapter 2, the first layer of ConvNets mostly learns edges and color gradients.
Clearly, edges and color gradients are not enough to differentiate between rock types, with
the slightly deeper networks performing better than the 1-layer networks. With adding more
convolution layers, the 3-layer network can learn more complicated features in the same way
that intermediate convolution layers trained on the ImageNet dataset learns to extract object
shapes and more specific patterns.

Figure A.6 shows the loss and accuracy per epoch during training. Both of the networks
exhibited an increasing trend in accuracy during training. The 3-layer network in fact, learns
quickly, where it reaches validation accuracy of around 90% before 25 epochs of training. On
the other hand, the 2-layer network barely reaches 80% validation accuracy after 25 epochs of
training. Judging from the graphs, there is no real benefit of training past 200 epochs because
the validation accuracies do not increase significantly after that.
3.2.4 Fully Connected Combinational Network

In a more recent work, Shu et. al. proposes the Fully Connected Combinational Network to aid with the search of the optimal architecture for a given problem [26]. The idea is to combine different networks with different configurations into one path. This allows us to train multiple networks and combine the results of the operations with the same output dimensionality in the hopes that the combined network would perform better than the individual. To illustrate, we take a look at the the 3-layer network in the previous section and layout the output shapes of each of the operations:

<table>
<thead>
<tr>
<th>Layer</th>
<th>Receptive Field Size</th>
<th>Stride</th>
<th>Zero Padding</th>
<th>Number of Filters</th>
<th>Output Shape</th>
</tr>
</thead>
<tbody>
<tr>
<td>Convolution</td>
<td>3x3</td>
<td>1</td>
<td>0</td>
<td>32</td>
<td>(126, 126, 32)</td>
</tr>
<tr>
<td>Max Pooling</td>
<td>2x2</td>
<td>2</td>
<td>0</td>
<td>-</td>
<td>(63, 63, 32)</td>
</tr>
<tr>
<td>Convolution</td>
<td>3x3</td>
<td>1</td>
<td>0</td>
<td>32</td>
<td>(61, 61, 32)</td>
</tr>
<tr>
<td>Max Pooling</td>
<td>2x2</td>
<td>2</td>
<td>0</td>
<td>-</td>
<td>(30, 30, 32)</td>
</tr>
<tr>
<td>Convolution</td>
<td>3x3</td>
<td>1</td>
<td>0</td>
<td>64</td>
<td>(28, 28, 64)</td>
</tr>
<tr>
<td>Max Pooling</td>
<td>2x2</td>
<td>2</td>
<td>0</td>
<td>-</td>
<td>(14, 14, 64)</td>
</tr>
<tr>
<td>Flatten</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>12544</td>
</tr>
</tbody>
</table>

Table 3.10: Output shape for each layer in the 3-layer network.

Modifying the architectures and tweaking the parameters of the operations could also go through the same path i.e., the output shapes of the operations would be equivalent at different points but end up at the target output shape of 12,544 when flattened. For instance, tables 3.11 and 3.12 lays out the output shapes of 2 other different networks with similar paths.

Notice that in tables 3.11 and 3.12, purely convolution layers are used instead of pooling layers. Also, different receptive field sizes are used for the filters in each layer. To match the outputs of each operation with the ones in the original 3-layer network, the strides of each convolution was increased, and zero padding was added accordingly. Combining these 3 networks opens up the possibility of optimizing the choice of receptive field sizes by considering all of
3.2. Applying a Convolutional Neural Network on the Dataset

<table>
<thead>
<tr>
<th>Layer</th>
<th>Receptive Field Size</th>
<th>Stride</th>
<th>Zero Padding</th>
<th>Number of Filters</th>
<th>Output Shape</th>
</tr>
</thead>
<tbody>
<tr>
<td>Convolution</td>
<td>4x4</td>
<td>2</td>
<td>1</td>
<td>32</td>
<td>(126, 126, 32)</td>
</tr>
<tr>
<td>Convolution</td>
<td>5x5</td>
<td>2</td>
<td>0</td>
<td>32</td>
<td>(63, 63, 32)</td>
</tr>
<tr>
<td>Convolution</td>
<td>4x4</td>
<td>2</td>
<td>1</td>
<td>64</td>
<td>(14, 14, 64)</td>
</tr>
<tr>
<td>Flatten</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>12544</td>
</tr>
</tbody>
</table>

Table 3.11: Output shape for each layer in another 3-layer network that follows the same "path" as the previously created 3-layer network.

<table>
<thead>
<tr>
<th>Layer</th>
<th>Receptive Field Size</th>
<th>Stride</th>
<th>Zero Padding</th>
<th>Number of Filters</th>
<th>Output Shape</th>
</tr>
</thead>
<tbody>
<tr>
<td>Convolution</td>
<td>12x12</td>
<td>4</td>
<td>0</td>
<td>32</td>
<td>(30, 30, 32)</td>
</tr>
<tr>
<td>Convolution</td>
<td>4x4</td>
<td>2</td>
<td>1</td>
<td>64</td>
<td>(14, 14, 32)</td>
</tr>
<tr>
<td>Flatten</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>12544</td>
</tr>
</tbody>
</table>

Table 3.12: Output shape for each layer in another network that follows the same "path" as the previously created 3-layer network.

To combine these three networks, the results of the operations that have the same output shape are stacked on top of each other. This would then be the new input for the next set of operations. Figure 3.4 shows a diagram of how the concatenation is performed. The concept is reminiscent of the Inception Module where the results of a different sized convolutions are concatenated within a block. Although this time, the convolutions are not contained within a
block and are allowed to propagate through the entire network. As well, one could think of the Combinational Fully Connected Network as an ensemble of individual ConvNets. In fact, Veit et. al. shows evidence that residual networks like the ResNet50 behave as a collection of many paths of differing lengths, thus creating an ensemble of ConvNets [72]. In the same way, by allowing convolutions of differing sizes and and paths, the CFCN could be seen as an ensemble of networks.

Again, to keep things consistent, the hyperparameters in Table 3.5 have been used in testing the effectiveness of this type of network.

Overall, the full architecture of the CFCN for this problem is shown in Figure 3.5. Three separate networks have been combined, and the output of operations with the same output dimensionality have been concatenated.
3.2. Applying a Convolutional Neural Network on the Dataset

Figure 3.5: Combinational Fully Connected Network Architecture.
Results

<table>
<thead>
<tr>
<th>Model</th>
<th>Average Accuracy</th>
<th>Average Loss</th>
<th>Accuracy Standard Deviation</th>
<th>Loss Standard Deviation</th>
</tr>
</thead>
<tbody>
<tr>
<td>CFCN</td>
<td>99.36%</td>
<td>0.1881</td>
<td>0.007630</td>
<td>0.01088</td>
</tr>
</tbody>
</table>

Table 3.13: Combinational Fully Connected Network Results.

Across 10 trials, the accuracy of the CFCN performed slightly worse than the 3-layer linear network at 99.36%, further reinforcing the idea that linear and non-complicated networks seem to work best for relatively simpler datasets. Figure A.7 shows the accuracy and loss per epoch during training and validation. As expected, the graphs show that the network successfully learns on the training set. At the same time, the accuracy and loss on the validation set shows that the network improves constantly even on never before seen data during training.

3.3 Model Deployment on an iPad

For the models to be useful to a geologist, they have to be portable enough to be deployed in a mobile device. Since the best model in terms of the lowest loss is saved for each trial, one of the best performing models for both the Linear and CFCN model have been chosen to be deployed on an iPad. To accomplish this, the model has to be converted first from a Keras module into one that is supported by Apple applications, i.e. into a CoreML module. The conversion is done using CoreMLTools [71], a toolkit published by Apple Inc. The conversion is straightforward, and a simple application has been developed that accepts an image and outputs the predicted type of rock. Figure 3.6 shows a screenshot of an iPhone simulator that displays the intended functionality of the app. The input image is displayed at the center and the predicted class, alongside the confidence of the prediction is shown as text at the bottom.
Finally, the models were installed on an iPad, and was tested with the 5 random images that were excluded in the dataset. The duration of the operation from the selection of the image up until displaying the prediction was recorded. At the same time, the accuracy and the confidence of each prediction was recorded.

<table>
<thead>
<tr>
<th>Model</th>
<th>Average Confidence</th>
<th>Prediction Duration</th>
<th>Confidence SD</th>
<th>Prediction Duration SD</th>
</tr>
</thead>
<tbody>
<tr>
<td>Linear</td>
<td>99.5%</td>
<td>0.06795</td>
<td>0.01818</td>
<td>0.05041</td>
</tr>
<tr>
<td>CFCN</td>
<td>99.7%</td>
<td>0.1869</td>
<td>0.01202</td>
<td>0.04639</td>
</tr>
</tbody>
</table>

Table 3.14: iPad deployment results.

As expected, both of the models achieve an accuracy of 100% on the iPad. There really is no reason to expect otherwise because the model already achieves 100% on the test set. On average, the time it takes for each prediction is a respectable 0.0680 seconds for the Linear model, equivalent to about 14 images per second. As for the CFCN model, the whole
operation takes about a magnitude longer at about 0.1869 seconds per prediction, equivalent to about 5 frames per second. Understandably, it takes longer for the CFCN because the image has to go through more layers than in the Linear model. Although it takes longer for each prediction, the CFCN has a slightly higher confidence for all the predictions.

3.4 Summary

In this chapter, we showed prior work in classifying rock images using the traditional image classification pipeline of feature selection and then training a classifier. Shu et. al. showed impressive results with an unsupervised method of selecting features alongside an SVM classifier [7]. Then, we explored the application of ConvNets on Shu et. al.’s dataset by first using transfer learning from networks trained on the ImageNet dataset. The networks performed worse than the SVM implemented by Shu et. al., and therefore calls for a custom network. We then showed the design process of creating an accurate classifier, by training networks with varying depths and filter numbers. The optimization is taken a step further by creating Combinational Fully Connected Networks(CFCN). Ultimately, the highest accuracy was achieved with a 3-layer linear network, with an average of 99.60% across 10 trials of training and testing. The CFCN only performed slightly worse at 99.36%.

We have also shown that the models could be deployed in a mobile application and could be useful for geologists who wish to identify images of rocks. However, the model is currently limited to the 9 original classes and expanding to more types of rocks should be included in a future work. This could be achieved by taking images of more types of rocks through the same method, i.e. using a microscope and a digital camera. However, this could be a tedious and arduous process for a geologist.
Chapter 4

Natural Scene Rock Image Classification

In the previous chapter, it has been shown that using Convolutional Neural Networks (ConvNets) is an effective way to classify clean, uniform, and standardized images of rocks. As well, it has been discussed in chapter 2 how deep ConvNets are effective in classifying natural scene images i.e., the images in the ImageNet dataset. Would it then be possible to combine the two concepts and create a natural scene rock image classifier using ConvNets?

We define natural scene rock images as images of rocks that are taken without special equipment (other than a camera), without a standardized method of taking the image, and without cleaning the rock samples. Often, these are images that a geologist would take in the field using a simple digital camera. In effect, the images would not purely consist of the rock in question. Shadows could occlude the target rock sample, foliage would be present, and soil and dirt could cover up the rocks. At the same time, the images would be of different scales and orientations, depending on the manner with which the image was taken.

In the field, it would be more useful to create a model that would classify natural scene images of rocks because a microscope with a digital camera attachment would not be readily available at that point. Considering the time it takes to bring a sample back to the laboratory to take clean images of the rock, it would make much more sense to accommodate the conditions in the field rather than constrain the geologist to clean and uniform images. Planetary rovers would also benefit greatly from a natural scene image classifier because rock sample images taken from a rover would rarely be clean and consistent. Therefore, a model that affords spatial invariance and compositional inconsistencies would be greatly advantageous overall.
4.1 Related Work

Detecting and classifying rock images in natural scenes has been previously examined by Dunlop et. al [73]. In their work, the task has been divided into two parts: first detecting the rock and segmenting it from the overall image, and second, classifying the specific rock type of the segmented image. Their dataset is composed of 8 color images with resolution of 2048 x 1536. Each image contains approximately 15 rocks, with the rocks placed in a bed of sand to mimic the environment with which the rocks would have been seen naturally. Figure 4.1 shows the rock samples present in the images.

To segment the rocks from the images, the authors used the Super Pixel Segmentation algorithm [74, 75] alongside hand crafted features including albedo, color, texture, shape features and Haralick features. The authors achieved good results in detecting and localizing the rocks in the images with this method, obtaining a region labelling accuracy of 99.6% in distinguish-
4.1. Related Work

ing background from the actual rock in the images. Figure 4.2 shows an image with the rocks segmented out.

Figure 4.2: Individual rocks segmented from the original image taken from [73].

To classify the segmented rock images, a training set has been created from 7 out of the 8 images, with each rock classified into two classes: chemical and clastic. Chemical rocks have been defined as rocks whose grains were chemically precipitated or recrystallized, and clastic rocks were defined as rocks whose grains were mechanically deposited [76]. The authors then used the K-NN algorithm as well as an SVM to classify the images. The features used were similar to those used to segment the images. Figure 4.3 shows the accuracies of the tested algorithms. Overall, the SVM performed best with a classification accuracy of 86.3%.
4.2 ConvNets on Natural Scene Rock Images

4.2.1 Dataset

For this research, the dataset of rock images in natural scenes is taken from the database of a field notebook iOS application being developed in the University of Western Ontario under the supervision of Dr. Gordon Osinski. With this app, geologists can take photos of outcrops and samples, add notes for each entity, and upload them into a database which can then be viewed through a website. The notes often describe the lithology of the rocks in the image, which provides a rudimentary label for each image. Images without notes are discarded from the dataset. Also, there is no standard when it comes to how the geologist is supposed to take images of rocks. One could be standing a few meters away from an outcrop, or have a zoomed in image of a sample in hand. This results into images without a uniform scale. It goes without saying that this dataset would be more representative of rock images in natural scenes than the ones presented by Dunlop et al. because the photos would have been taken when the geologist was out in the field.

Another thing to note with this dataset is that there are extraneous objects in the images that would cause problems in classifying the underlying rocks. For instance, looking at Figure 4.4, the image on the top right has the sky and green bushes in the background that is irrelevant to the type of rock in the foreground. As well, the two images on the right have rock hammers in them, and the one on the left has a lens cap, both of which serve the purpose of being a crude scale for the image. Creating a classifier that could disregard these objects is tricky and would be very difficult to implement. To further illustrate this difficulty, Figure 4.5 shows images that
4.2. **ConvNets on Natural Scene Rock Images**

Figure 4.4: Sample images from the database.

have different labels or notes in them but contain similarly looking rock hammers. It must be noted however that images in the ImageNet dataset have the same complications but ConvNets were able to classify the images well.

On top of the complications stated previously, the fact that the images within the database have differing resolutions contribute to the difficulty in obtaining clear textural differences from the rocks contained in an image.

Taking a step towards classifying rocks in natural scene images, the task has been reduced to a binary classification problem, classifying rocks into two classes: breccia and non-breccia. A breccia is a rock that contains angular fragments that are cemented together. Figure 4.6 shows a sample of a breccia in the dataset. Notice the non uniformity in the rock and the inclusion of angular clasts. Non-breccia images would then be the rest of the images within the database that do not have the word "breccia" or "clasts" in their notes. These images could have other rock type labels attached to them like "limestone", "quartz", etc. These could be used to create a more general classifier. But for the purposes of this research, we are simply proving the concept with a binary classifier. Choosing between a breccia and a non-breccia provides an easily identifiable texture difference between the images. However, the dataset contains images
Figure 4.5: Sample images in the database of different rocks that contain similarly looking rock hammers.

that are labelled other than breccia. On that note, a sample image of a non-breccia is shown in Figure 4.7.

Again, notice how in the shown images, there is part of a human hand. To us humans, distinguishing hand, hammer, or lens cap from rocks is a fairly easy task because of the obvious texture and edge differences. For a computer however, it remains a possibility where a classifier would "see" the human hand and classify both the images as belonging in the same class. To circumvent this, the images have been manually cropped to include solely the rock. Admittedly, this is a very crude solution to the problem and ideally, the segmentation of the rocks within the image is included as a preprocessing step before the classification. Hopefully, this would be included in a future work.
4.2. **ConvNets on Natural Scene Rock Images**

The images were then cropped into 256x256 patches. A few examples of the cropped images are shown in figures 4.8 and 4.9. Cropping the breccia images produced 9,128 breccia cutouts. To create an approximately balanced distribution of breccia and non breccia images, 9,317 non breccia cutouts were produced, creating a total of 18,445 images in the dataset. A classifier that guesses all images to be purely of one class would get an accuracy of about 50%. It is therefore the goal of the classifier to achieve an accuracy that is substantially better than 50%.

4.2.2 **Transfer Learning**

This section outlines the application of Transfer learning to the dataset to see how well the networks trained on the ImageNet dataset would perform in classifying natural scene images of rocks.

**Experimental Setup**

As was in chapter 2, the same 9 networks were used for transfer learning:
The output layer is reduced from 1000 nodes to 2 nodes, as this is a binary classification problem. The hyperparameters for training the networks are very similar to the one used in the previous chapter. The only difference is that the learning rate has been further reduced to $1e^{-5}$ as using a higher learning rate yielded poor results. As well, the number of trials for transfer learning has been reduced to 5 because training each network takes about 24 hours to finish. Therefore, training 9 different networks for 10 trials would have taken more than 3 months to finish. To improve classification accuracy, data augmentation has been applied to the dataset with the exception of adding Gaussian noise and blurring because of hardware limitations. Adding gaussian noise alone in each and every one of the 18,445 images would have doubled the number of images in the dataset and would require more than 100GB of memory and we are limited to 16GB at the moment. An attempt was made to train the networks while adding Gaussian noise to the images but the computer would constantly lock up and freeze.
4.2. **ConvNets on Natural Scene Rock Images**

![Figure 4.8: Cropped Breccia Images.](image)

![Figure 4.9: Cropped Non-Breccia Images.](image)

**Results**

Table 4.2 shows the results of transfer learning after 200 epochs. Among all of them, only the VGG networks performed significantly better than random guessing, achieving more than 80% accuracy. The rest of the networks only achieved an accuracy of less than 70%. ResNet50 and MobileNetV2 did not get any better than random guessing, only achieving an accuracy of 50%. So far, the trend continues that linear networks (VGG16 and VGG19) perform better than the more complicated networks. Training and Validation loss per epoch during training reinforce these results, showing that only the VGG networks (figures B.1a to B.1d) successfully learned on the training set and generalized it well with the validation set per epoch. The rest of the
## Hyperparameters

<table>
<thead>
<tr>
<th>Hyperparameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Epochs</td>
<td>200</td>
</tr>
<tr>
<td>Batch Number</td>
<td>16</td>
</tr>
<tr>
<td>Optimizer</td>
<td>Adam</td>
</tr>
<tr>
<td>Learning Rate</td>
<td>0.00001</td>
</tr>
<tr>
<td>Training set</td>
<td>70%</td>
</tr>
<tr>
<td>Validation set</td>
<td>15%</td>
</tr>
<tr>
<td>Test set</td>
<td>15%</td>
</tr>
<tr>
<td>Trials</td>
<td>5</td>
</tr>
</tbody>
</table>

Table 4.1: Hyperparameters used for Transfer Learning as well as training the custom networks.

networks simply overfitted on the training set and failed to generalize well (figures B.1e to B.3f).

### 4.2.3 Custom Networks

#### Experimental Setup

In designing a custom network, it has been decided to create a network deeper than 3 layers because the dataset is more complicated and because the images have higher resolution than the images in Shu et. al.’s dataset. For these reasons, a 5-layer ConvNet has been designed. The model summary is shown in Table 4.3.

As in chapter 3, a Combinational Fully Connected Neural Network was created, combining 5 different ConvNets that go along the same path of the linear network shown in Figure 4.3. This way, integrating multiple networks into one allows us to optimize for different network configurations. The architecture of the network is shown in Figure 4.10.
### Table 4.2: Transfer learning results for each network.

<table>
<thead>
<tr>
<th>Model</th>
<th>Average Accuracy</th>
<th>Average Loss</th>
<th>Accuracy Standard Deviation</th>
<th>Loss Standard Deviation</th>
</tr>
</thead>
<tbody>
<tr>
<td>VGG16</td>
<td>83.98%</td>
<td>0.3615</td>
<td>0.006508</td>
<td>0.009485</td>
</tr>
<tr>
<td>VGG19</td>
<td>81.38%</td>
<td>0.4094</td>
<td>0.002606</td>
<td>0.001007</td>
</tr>
<tr>
<td>ResNet50</td>
<td>50.98%</td>
<td>0.7674</td>
<td>0.007635</td>
<td>0.1041</td>
</tr>
<tr>
<td>MobileNetV2</td>
<td>50.28%</td>
<td>1.139</td>
<td>0.002299</td>
<td>0.1712</td>
</tr>
<tr>
<td>InceptionV3</td>
<td>69.55%</td>
<td>0.6165</td>
<td>0.02088</td>
<td>0.02785</td>
</tr>
<tr>
<td>InceptionResNetV2</td>
<td>68.77%</td>
<td>0.6173</td>
<td>0.001533</td>
<td>0.02447</td>
</tr>
<tr>
<td>DenseNet121</td>
<td>57.04%</td>
<td>0.7497</td>
<td>0.05289</td>
<td>0.1119</td>
</tr>
<tr>
<td>DenseNet169</td>
<td>60.93%</td>
<td>0.6932</td>
<td>0.07564</td>
<td>0.08045</td>
</tr>
<tr>
<td>DenseNet201</td>
<td>61.63%</td>
<td>0.6836</td>
<td>0.07233</td>
<td>0.07416</td>
</tr>
</tbody>
</table>

### Results

With keeping the hyperparameters the same, Table 4.4 shows the results obtained from both the linear and the CFCN network.

### Table 4.4: Custom Networks Results.

<table>
<thead>
<tr>
<th>Model</th>
<th>Average Accuracy</th>
<th>Average Loss</th>
<th>Accuracy Standard Deviation</th>
<th>Loss Standard Deviation</th>
</tr>
</thead>
<tbody>
<tr>
<td>5-Layer</td>
<td>89.43%</td>
<td>0.2720</td>
<td>0.004186</td>
<td>0.008157</td>
</tr>
<tr>
<td>CFCN</td>
<td>93.51%</td>
<td>0.1647</td>
<td>0.005430</td>
<td>0.01229</td>
</tr>
</tbody>
</table>

Both the custom built networks performed better than the pretrained networks with the linear network achieving 89.43% accuracy and the CFCN achieving a better 93.51% accuracy after 200 epochs. Interestingly, the CFCN performs better than the linear network, presumably
because the dataset is more complicated for this task. It seems like the more complicated the
dataset, the more complicated the network needs to be.

Looking at the accuracy and loss per epoch graphs in Figure B.4, training for more than
200 epochs would not seem to yield any better accuracies for the CFCN because the validation
accuracies and loss approaches an asymptote at around the reported average accuracies and
loss. Training for longer would have improved on the training accuracies, but that would
simply mean that the classifier is overfitting on the training set.

<table>
<thead>
<tr>
<th>Layer type</th>
<th>Output Shape</th>
<th>Number of Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>Convolution</td>
<td>(254, 254, 128)</td>
<td>3584</td>
</tr>
<tr>
<td>Max Pooling</td>
<td>(127, 127, 128)</td>
<td>0</td>
</tr>
<tr>
<td>Convolution</td>
<td>(125, 125, 64)</td>
<td>73792</td>
</tr>
<tr>
<td>Max Pooling</td>
<td>(62, 62, 64)</td>
<td>0</td>
</tr>
<tr>
<td>Convolution</td>
<td>(60, 60, 32)</td>
<td>18464</td>
</tr>
<tr>
<td>Max Pooling</td>
<td>(30, 30, 32)</td>
<td>0</td>
</tr>
<tr>
<td>Convolution</td>
<td>(28, 28, 32)</td>
<td>9248</td>
</tr>
<tr>
<td>Max Pooling</td>
<td>(14, 14, 32)</td>
<td>0</td>
</tr>
<tr>
<td>Convolution</td>
<td>(12, 12, 32)</td>
<td>9248</td>
</tr>
<tr>
<td>Flatten</td>
<td>4608</td>
<td>0</td>
</tr>
<tr>
<td>Dropout</td>
<td>4608</td>
<td>0</td>
</tr>
<tr>
<td>Dense</td>
<td>128</td>
<td>589952</td>
</tr>
<tr>
<td>Dropout</td>
<td>128</td>
<td>0</td>
</tr>
<tr>
<td>Dense</td>
<td>64</td>
<td>8256</td>
</tr>
<tr>
<td>Softmax</td>
<td>2</td>
<td>130</td>
</tr>
</tbody>
</table>

Table 4.3: Custom linear network architecture with 5 convolutional layers.
4.3 Summary

In this chapter, we have shown that Convolutional Neural Networks are effective as well in classifying rock images in natural scenes. As a baseline, a thesis reporting on natural scene image classification with rocks was presented and ConvNets were able to outperform the related literature by a substantial amount. Both the related literature and this chapter attempts to solve a binary classification problem. However, the images in the dataset presented in this chapter are more representative of what a rock in a natural scene would look like as these images are not staged in any way shape or form. Overall, custom networks outperformed pre-trained networks by a wide margin, with the CFCN achieving the highest accuracy at 93.51%.
Figure 4.10: Breccia CFCN Model Architecture.
Chapter 5

Conclusion

5.1 Summary

With the recent rise of Machine Learning and Artificial Intelligence, researchers are finding more and more applications for image recognition and object detection. Furthermore, machine learning has tremendous potential in the field of geology, earth sciences, and planetary exploration. With this thesis, we have shown that Convolutional Neural Networks can be successfully applied in classifying clean images of rocks, achieving almost 100% accuracy across 10 trials of testing. Through this technique, it eliminates the need to manually select features with which to represent images of rocks, and ultimately removes the bias coming from researchers. Although the technique is more costly in terms of memory requirement and processing power compared to traditional methods of classifying images, we have shown that deploying this system in a mobile application is very much feasible with each prediction only taking about 0.06795 seconds, which is equivalent to almost 14 frames per second. We have also shown a way of optimizing and designing a ConvNet via Combinational Fully Connected Networks and it has shown results on par with a Linear ConvNet.

Although huge success has been shown in classifying clean images of rocks, a more interesting problem lies in classifying images of rocks in natural scenes. These are images that are typically taken by geologists in the field, without the aid of specialized equipment and without a standardized method of taking the pictures. This then implies that the images are occluded with random objects like foliage, soil and mud, and shadows. As well, there is a substantial
scale and orientation variance depending on how the geologist takes pictures of outcrops and samples. Being able to classify these images would demonstrate a highly universal rock classifier. On top of that, applications beyond a geologist holding a camera can be proven to be possible. For instance, an exploration team that deploys drones and rovers in remote areas would be able to identify rocks without having to bring back samples to the laboratory. As well, planetary rovers would be able to classify surrounding rocks on site, without needing a much time consuming human intervention.

Taking a step towards the said goal, we tackled a binary classification problem (breccia vs. non-breccia) for rock images in a dataset of a field notebook iPad application being developed in the University of Western Ontario. In this app, geologists can take pictures of outcrops and samples in the field and write down notes pertaining to the images. These notes often contain the lithology of the outcrop or sample, and thus can be taken as their labels. At the same time, the geologists can record the location and heading of each outcrop or sample, then upload their data to an online database for reviewing through a website. With this dataset, we then designed and tested two ConvNets for classification. We have shown that a Combinational Fully Connected Network performs well at 93.50% accuracy, slightly better than a Linear ConvNet, which achieves 89.43% accuracy.

The contributions of this thesis can be summarized the following points:

- We have shown that Convolutional Neural Networks have superior performance over traditional methods in classifying clean images of rocks
- We have shown that the models are lightweight enough to be deployed in a mobile device
- We have shown that classifying natural scene images of rocks is achievable using ConvNets
- We have shown that designing Combinational Fully Connected Networks is a viable optimization tool for ConvNets
5.2 Future Work

Ideally, any user with an iPad should be able to take any picture of a rock in any angle, lighting condition, or location, and receive feedback on the type of rock within the image. This entails that a preprocessing step be included in the pipeline, one of which is segmentation of the rock before classification. This eliminates the need to manually crop the rocks in the image and essentially, it automatically isolates the rocks before feeding them into a classification algorithm.

Even though increasing the number of rocks that the classifier can handle is an obvious research direction to take, it would be more advantageous to implement an unsupervised learning algorithm that could cluster un-identifiable rocks and have an expert geologist confirm their lithology later on. This would be more useful in the field when a rover or a drone is to be deployed at an extended period of time with limited human contact. Tying the classification of rocks back to the field notebook application, a scenario where the app classifies a rock and points to similarly looking rocks in other locations could provide invaluable insight to geologists.

Finally, deploying the model in a more appropriate system should be considered, such as a microcontroller with a camera input, or an on board computer for a rover or a drone. This would demonstrate that a more practical application is in the horizon for classifying images of rocks using ConvNets.
Appendix A

Training and Validation Graphs for Chapter 3: 9-Class Rock Dataset
Figure A.1: Training and validation graphs for VGG16, VGG19, and ResNet50.
Figure A.2: Training and validation graphs for MobileNet, InceptionV3, and InceptionResNetV2.
Figure A.3: Training and validation graphs for the DenseNet networks.
Figure A.4: Training and validation graphs for the 1-layer networks.
Figure A.5: Training and validation graphs for the 1-layer networks.
Chapter A. Training and Validation Graphs for Chapter 3: 9-Class Rock Dataset

(a) 2 layer network training accuracy.  
(b) 2 layer network training loss.  
(c) 3 layer network training accuracy.  
(d) 3 layer network training loss.  

Figure A.6: 2 Layer and 3 Layer network loss and accuracy during training.  

(a) CFCN training and validation accuracy.  
(b) CFCN training and validation loss.  

Figure A.7: CFCN loss and accuracy during training.
Appendix B

Training and Validation Graphs for
Chapter 4: Breccia vs Non-Breccia
Figure B.1: Training and validation graphs for VGG16, VGG19, and ResNet50.
(a) MobileNet Training and Validation Accuracy.  
(b) MobileNet Training and Validation Loss.  

(c) InceptionV3 Training and Validation Accuracy.  
(d) InceptionV3 Training and Validation Loss.  

(e) InceptionResNetV2 Training and Validation Accuracy.  
(f) InceptionResNetV2 Training and Validation Loss.  

Figure B.2: Training and validation graphs for MobileNet, InceptionV3, and InceptionResNetV2.
Chapter B. Training and Validation Graphs for Chapter 4: Breccia vs Non-Breccia

(a) DenseNet121 Training and Validation Accuracy.
(b) DenseNet121 Training and Validation Loss.

(c) DenseNet169 Training and Validation Accuracy.
(d) DenseNet169 Training and Validation Loss.

(e) DenseNet201 Training and Validation Accuracy.
(f) DenseNet201 Training and Validation Loss.

Figure B.3: Training and validation graphs for the DenseNet networks.
(a) Breccia Linear Network Accuracy per epoch.  
(b) Breccia Linear Network Loss per epoch. 

(c) Breccia CFCN Network Accuracy per epoch.  
(d) Breccia CFCN Network Loss per epoch. 

Figure B.4: 2 Layer and 3 Layer network loss and accuracy during training.
Bibliography


[114] E. Poisson, “Multi-modular architecture based on convolutional neural networks for on-
line handwritten character recognition,” *Neural Information . . .*, vol. 5, pp. 2444–2448,
2002.

houcke, P. Nguyen, T. N. Sainath, and B. Kingsbury, “Deep Neural Networks for Acous-
tic Modeling in Speech Recognition,” *Ieee Signal Processing Magazine*, no. November,

[116] Y. Li, X. Zhang, and D. Chen, “CSRNet: Dilated Convolutional Neural Networks for
Understanding the Highly Congested Scenes,” 2018.

[117] R. S. Sutton, “Two problems with backpropogation and other steepest-descent learning
prodecures for networks,” in *8th annual conference: Cognitive Science Community*,
1986.

Technologies to Map the Structural Geology of Central Region of Kenya,” *IEEE Journal
of Selected Topics in Applied Earth Observations and Remote Sensing*, vol. 8, no. 4,


of geological landforms on Mars using Convolutional Neural Networks,” *Computers


<table>
<thead>
<tr>
<th>VITA</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Name:</strong> Alexis David P. Pascual</td>
</tr>
<tr>
<td><strong>Post-Secondary Education and Degrees</strong></td>
</tr>
</tbody>
</table>
| Ateneo De Manila University  
Quezon City, Manila, Philippines  
2010-2015 B.S. |
| Western University  
London, Ontario, Canada  
2017-2019 MESc |
| **Related Work Experience** |
| Mitacs Accelerate Internship  
2017 |
| Teaching Assistant  
Western University  
2017-2019 |
| Research Assistant  
Western University  
2017-2019 |
| **Publications and Presentations** |