AUTOENCODERS FOR COMPRESSION

Applications in Medical Image Analysis

School of Computer Science & Applied Mathematics
University of the Witwatersrand

Ifthakaar Shaik
0603446E

Dr Richard Klein

October 2, 2021

A research report submitted to the Faculty of Science, University of the Witwatersrand, Johannesburg, in partial fulfilment of the requirements for the degree of Master of Science by Coursework and Research Report

Abstract

We propose the application of autoencoders to the task of compression within the field of blood analysis. The growing need for medical care and shortage of medical personnel requires a more decentralised approach to medical diagnostics. Remote diagnostics in rural areas requires the ability to transmit digital medical image data of typically large file sizes over possible low bandwidth networks. The regularity found in medical samples may be exploited to allow for higher compression than generic compression algorithms while maintaining quality suitable for medical interpretation by human doctors. This paper explores the ability of a deep learning technique, namely autoencoders, to learn this regularity to create a lower-dimensional representation of the data. We assess the autoencoder's performance by comparing it to traditional JPEG compression. We measure performance by considering the compression levels achieved and the quality of the resultant compressed images. Quality is measured using Peak Signal to Noise Ratio and Structural Similarity Index Measure, with the best techniques maximising values under each of these. Our results show that JPEG outperforms autoencoders at low compression levels, but as the compression level increases, autoencoders outperform JPEG based on the quality metrics used. We further test the impact of compression on the task of classification using a classification neural net. The results from our experiments show that compression negatively affects accuracy, however this loss in accuracy may be recovered by retraining the classifier on compressed images. The combination of results indicates that autoencoder-based compression techniques hold promising applications in the field of remote medical diagnostics, especially in rural areas where the ability to transmit data is limited by infrastructure.

Declaration

I, Ifthakaar Shaik, hereby declare the contents of this research proposal to be my own work. This proposal is submitted for the degree of Master of Science in Computer Science at the University of the Witwatersrand. This work has not been submitted to any other university, or for any other degree.

Acknowledgements

This paper's topic is inspired by the millions of children who suffer each day from the lack of medical care. As the medical field slowly embraces newer technological alternatives to healthcare delivery, I hope that this paper's findings can contribute in some way to alleviating their plight.

To this end, I would like to thank Vitruvian Medical Diagnostics for their work in promoting decentralised diagnostics and for allowing me access to their facilities and networks. I would also like to thank Dr Richard Klein for introducing me to the fascinating field of computer vision and supervising my research.

This paper is dedicated to my late grandmother, who encouraged me to seek knowledge from the cradle to the grave and my mother, who instilled in me the idea that tomorrow is a place we create.

Contents

Chapter 1

Introduction

1.1 Introduction

Access to healthcare is a significant issue faced by people in developing countries. High cost of care, shortage of trained personnel and geographic limitations present considerable challenges to effective healthcare service delivery in Emerging Markets. Worldwide there are 3.5 billion people without access to essential healthcare [World Health Organisation 2017].

If we consider the field of pathology in Africa, access to reliable diagnostic testing is severely limited, with an average ratio of pathologists to patients of 1:1000000 [Fleming 2019]. This problem leads to misdiagnosis, inadequate treatment, increased mortality, and an inability to determine diseases' true prevalence. An obvious solution to the problem focuses on building Machine Learning ("ML") based software to facilitate the diagnosis of diseases and parasite detection. There have been many papers published in this field across various verticals within pathology. A fundamental assumption in all these techniques is the availability of digital whole slide images. This assumption is also a critical failing in practice. While it may be available in high end, large, private pathology labs, the machinery typically used to digitise slide samples is prohibitively expensive (>USD150,000 per machine). These machines are also non-portable, require regular maintenance and stable electricity supply; conditions that all labs across the developing world do not readily meet.

Companies such as Vitruvian Medical Diagnostics ("VMD") have invented simple, low-cost hardware that retrofits existing microscopes and enables capturing digital slide images. These digital images can then be processed via software to yield diagnostic results. However, an additional practical limitation is the transmission of these images over data networks. In rural areas where network coverage is typically poor, data costs tend to be high and bandwidth low, making transmission of typically large digital files difficult. While generic compression techniques such as JPEG exist, these are typically not well suited to the task of medical image compression as they are designed to work on natural images and do not consider the regularity specific to the images in question. For this reason, novel compression techniques need to be employed to enable the transmission of slide image data from a field site to a central lab where trained pathologists can interpret the image for diagnosis or for a cloud-based ML engine to process the image to provide a diagnostic result.

While much work has been done on general image compression, this research project investigates the usefulness of ML-based compression techniques (Autoencoders) in facilitating the compression of slide image data for transmission.

The application case is very specific, focusing on blood image data, with the foundational idea being the exploitation of the limited dimensionality of blood image data leading to much higher compression rates for this particular case of images.

Given the regularity found in blood images, autoencoders may be well suited to learn the features of the cells and use this to enhance compression and quality. The encoder part of the network transforms an input image into a latent space representation with a smaller bit size than the input, providing the compression required. In contrast, the decoder part of the network reconstructs the image from the latent representation. In the task of transmission, it is the latent representation that is of interest. The ability to separate the encoder and decoder networks allows the encoder to be packaged and distributed with imaging hardware, allowing an image to be captured, transformed and transmitted. The decoder can be stored on the receiving end of the transmission, allowing incoming data to be reconstructed into an image that humans can diagnose.

The combination of the above features makes autoencoders an exciting topic for research as a compression tool to enable remote diagnostics.

1.2 Overview of the Paper

In Chapter 2, we present an overview of compression. Here we explore the key principles of compression, the history of its development and how performance is typically measured. The chapter ends with a summary of the types of compression used.

Chapter 3 explores in more detail the types of compression as introduced in chapter 2. This chapter considers popular techniques for both lossless and lossy compression types.

Chapter 4 discusses deep learning techniques in compression by introducing neural networks and their extension to autoencoders. We then explore how autoencoders may be applied to compression.

Chapter 5 centres on medical images (specifically blood images) and how compression may be applied. We present the need for compression and investigate the features of blood that make it an attractive candidate for autoencoder compression.

Chapter 6 presents the methodology followed for our research and outlines the questions that have been investigated.

Chapter 7 presents the results from our experiments and Chapter 8 concludes our report with a summary of key findings.

Chapter 2

Compression: An Overview

Digital compression is the application of mathematical techniques to encode information such that the encoded version is represented by fewer bits than the original representation. In the specific case of image compression (which is the focus of this research paper), it is applying data compression on digital images. Uncompressed digital data requires more digital storage space and transmission bandwidth than compressed data. As the access to technology increases, the creation of digital multimedia has become more widespread. Limitations in digital storage and transmission from either technologically imposed limitations or cost have created a need for compression methods, making it central to digital communication.

2.1 General Principles of Compression

Data compression is the process in which the size of files or data is reduced in a manner that requires significantly less space than they have in their original format. In this process, the amount of data required for storage or transmission of a piece of information (graphics, text, video, or sound) is reduced.

In digital image compression, the basic principles are premised on the assumption that pixels in a neighbourhood are correlated and hence contain redundant information. Based on this assumption, the goal of compression is to find an uncorrelated or less correlated representation of the same image. To further expand the idea, the notion of redundant information is built on two key concepts: redundancy and irrelevance.

Redundancy refers to spurious data, wherein bits are used to store data that provide no meaningful information of the image in question. Removing redundant information still allows the image to be displayed without any error. This is referred to as lossless compression and is an important goal in the task of compression.

Irrelevance refers to data in the image that does not contribute meaningfully to its contextual understanding. The removal of this data allows the image to be stored with fewer bits; however, information is lost in this process. Typically, the information removed is not perceivable by the human visual system, and in that sense is irrelevant for a human to understand and interpret the image.

2.2 A Brief History of Compression

In 1808, Jean-Baptiste Joseph Fourier presented an idea in a paper titled *"Mémoire sur la propagation de la chaleur dans les corps solides"* (Treatise on the propagation of heat in solid bodies) [Poisson 1808], wherein he introduced the idea that a smooth function can be decomposed into sums of sinusoidal waves. While introduced for the analysis of heat, this idea has proved to have many applications in engineering and mathematics, not least of which in the field of compression as this type of transformation is a key step in popular compression techniques.

In 1838 Morse Code was invented. This was an early form of compression wherein shorter codewords were used to represent letters of the alphabet.

In the 1940's work began on the modern compression techniques that we use which culminated in the development of the field known as Information Theory.

In 1949 Claude Shannon and Robert Fano invented a means of systematically assigning codewords based on probabilities of blocks [Shannon 1949]. This was the foundation of what is known as Entropy Coding.

In 1952 David Huffman presented an optimised method of codeword assignment [Huffman 1952]. Early adoption of this method was typically for hardware, with static codeword choices which presented a trade-off between error correction and compression.

In the Mid 1970s the Huffman encoding idea was revised with dynamic updates of codewords which are based off the data actually encountered, thus making it an adaptive algorithm. As storage of text-files online began to gain traction in the late 1970's, compression software was being developed and adopted, many of which were based on this adaptive Huffman coding technique.

Ahmed *et al.* [1974] introduced the Discrete Cosine Transform ("DCT") which was an important development for image compression. This technique forms the basis of the JPEG compression and catalysed the creation of the Discrete Wavelet Transform ("DWT") which uses wavelet transforms in image compression [Hoffman 2012].

In their 1977 and 1978 papers, Ziv and Lempel [1977] and Ziv and Lempel [1978] introduced the idea of dictionary based coding. This idea was explored further and, in 1984, Terry Welch published his paper in which the Lempel Ziv Welch ("LZW") technique was presented [Welch 1984]. The LZW method became widely adopted as the choice algorithm in most general-purpose compression systems. LZW is used by the GIF file format.

As digital images gained popularity from the 1980's, the need for a standardised way of communicating this data became important which led to the development of various image compression standards. Many of these standards were typically based on lossy techniques (such as DCT in the case of JPEG). Examples of these standards include JPEG [Wallace 1992], GIF and PNG [Deutsch 1996].

2.3 Measuring Performance

When dealing with various compression techniques, it is important to be able to select the method that is most appropriate for the task at hand. This requires benchmarking

parameters that allow us to compare the various methods and choose the one that is most optimal. Key comparison factors are discussed below.

2.3.1 Compression Ratio

This is a measure of the ratio between the original data size and the compressed data size. It indicates how well a compressor can compress data, however it gives no information regarding loss of quality. One technique may produce exceptional compression, but the resultant data may be so lossy that it is incomprehensible. Equation (2.1) shows the formula used to calculate the compression ratio.

$$CR = \frac{S_u}{S_c} \tag{2.1}$$

where CR represents the compression ratio, S_u represents the size in bits of the uncompressed input image and S_c represents the size in bits of the compressed output image or latent space representation.

2.3.2 Compression Quality

Quality metrics are only necessary when considering lossy compression techniques. Lossless techniques (by definition) compress images with no loss in data.

The most important method of measuring compression quality relies on subjective observation by humans. However, this subjectivity may vary from person to person and hence it is more robust to employ an objective quality metric.

One method for this is the Peak Signal to Noise Ratio ("PSNR"). PSNR measures the amount of noise or distortion that is created due to lossy compression. It is the ratio between the maximum value of a signal and the value of the distorting signal. The amount of the distortion is typically measured using the Mean Square Error ("MSE") of the reconstructed image. The calculation for MSE and PSNR are shown in Equations (2.2) and (2.3) respectively.

$$MSE = \frac{1}{m * n} * \sum_{0}^{m-1} \sum_{0}^{n-1} ||O(i,j) - D(i,j)||^2 \tag{2.2}$$

$$PSNR = 20 * log_{10} \frac{Max_O}{\sqrt{MSE}} \tag{2.3}$$

where O represents the matrix data of the original image, D represents the matrix data of the distorted image after decompression, m represents the numbers of rows of pixels of the images and i represents the index of that row, n represents the number of columns of pixels of the image and j represents the index of that column and MAX_O is the maximum signal value the original image.

Considering the formula above, the PSNR approaches infinity as the MSE approaches zero, hence a high PSNR value implies a higher image quality (lower distortion in the image). Conversely, a lower PSNR implies that there is a large difference between the images, mathematically.

Another approach to measuring quality is the Structural Similarity Index Measure ("SSIM") introduced by Wang *et al.* [2004] and is considered to be more closely correlated to the quality perception of the human visual system. SSIM models loss of image quality as a combination of distortion in luminance, distortion in contrast and a loss of correlation between the two images (original and reconstructed). For any two images $\boldsymbol{x}$ (original image) and $\boldsymbol{y}$ (the compressed and reconstructed image) the SSIM is defined as follows:

$$SSIM(\boldsymbol{x}, \boldsymbol{y}) = f(l(\boldsymbol{x}, \boldsymbol{y}), c(\boldsymbol{x}, \boldsymbol{y}), s(\boldsymbol{x}, \boldsymbol{y}))$$

Luminance, $l(\boldsymbol{x}, \boldsymbol{y})$ is defined as:

$$l(\boldsymbol{x}, \boldsymbol{y}) = \frac{2\mu_x\mu_y + C_1}{\mu_x^2 + \mu_y^2 + C_1},$$

where μ_x and μ_y are the mean intensities in each image.

Contrast, $c(\boldsymbol{x}, \boldsymbol{y})$ is defined as:

$$c(\boldsymbol{x}, \boldsymbol{y}) = \frac{2\sigma_x\sigma_y + C_2}{\sigma_x^2 + \sigma_y^2 + C_2},$$

where σ_x and σ_y are the standard deviations of image intensity in each image.

Correlation, $s(\boldsymbol{x}, \boldsymbol{y})$ is defined as:

$$c(\boldsymbol{x}, \boldsymbol{y}) = \frac{\sigma_{xy} + C_3}{\sigma_x\sigma_y + C_3},$$

where C_1, C_2, C_3 are constants used to avoid instability when the denominator is zero (or close to zero).

The result from the SSIM calculation is a value in the range $[0, 1]$, where results near the upper bound indicate minimal degradation in the perceived quality between the images.

2.4 Types of Compression

Image compression can be divided into two broad categories, Lossless Image Compression and Lossy Image Compression.

Before exploring these, we need to understand the concept of entropy as it relates to information theory.

2.4.1 Information Entropy

Information Entropy and specifically entropy coding is a key element to understand when exploring compression techniques. It is applied in both lossless and lossy compression methods and was introduced by Shannon [1948].

Entropy refers to the probability weighted number of bits of information in a message, when taken over all possibilities. It measures how much useful information a

message is expected to contain and provides a lower bound on the size of the encoded message. If a compression algorithm achieves the lower bound, then it is lossless.

Entropy is defined as:

$$H(X) = -\sum_{i} P_X(x_i) log_2 P_X(x_i)$$

where X is a random variable associated with the contents of a message, with possible outcome x and probability $P_X(x_i)$. As can be noticed, as the probability of the content increases (example a more frequently occurring symbol in a message), the entropy decreases. The term $log_2 P_X(x_i)$ is also called the information content or the surprisal. This effectively implies that the more surprising an event (lower probability) the more information is contained by that event. Hence, when applying entropy coding, the most frequently occurring items in the data are assigned to the lowest bits, as by the entropy defined above, the higher frequency events provide less information (as they have a higher expectation of occurring).

2.4.2 Lossless Image Compression

In lossless compression, data are compressed with no loss of detail. For this reason it is also called reversible compression, as the compressed image can be decoded to reveal the original input data.

This form of compression is effective on images with few colours and few fine details. Given that lossless compression preserves all data in its process, the exact original image can be reconstructed from the compressed image [Sayood 2017]. Thus lossless compression is used where small deviations from the original data may be detrimental (example cancer diagnosis using prostate tissue images) or when the pixel values have semantic meaning (example semantic segmentation masks [Liu *et al.* 2019]).

Examples of common file formats that use lossless compression include PNG, GIF, and ZIP.

While lossless compression preserves the original image or data, the tradeoff lies in that these techniques result in lower compression ratios than can be achieved using lossy compression.

Lossless compression is often used as a component within a lossy compression process. Examples include JPEG that uses transform coding (a lossy technique) and Huffman coding (a lossless technique).

Lossless compression is typically broken down into two steps. In the first step a statistical model is generated for the input data. The second step uses this model to map the input data to a bit sequence such that frequently encountered data (more probable) produces shorter output than the infrequently occurring data (less probable). Examples of algorithms that produce these bit sequences are Huffman coding and Arithmetic coding.

The statistical models used for the coding may be either static or adaptive. In a static model, the data is first analysed and a model of the data is constructed. This model is stored with the compressed data. It has the advantage of being easy to implement and produces a fixed set of symbol probabilities (coefficients) that are used across the input data. Conversely, it may result in longer run times as a first pass over the data

is needed to produce a model and then a second pass to apply the coding technique. Further, given that the coefficients are fixed it performs poorly on heterogeneous data as the single model is applied to all the data (which is frequently changing). These models are also inefficient on storage as it requires the model to be stored with the compressed data.

An adaptive model is created as the data is read and compressed. The symbol probabilities of the data are updated as the compression process is run, allowing the model to adapt to the changing data. It has the advantage of improving performance as the encoder and decoder learn more about the data. A disadvantage of this approach is that the process starts with a trivial model that does not necessarily explain the data well. This results in the earlier part of the data experiencing poor compression (and if no initial model defined, then there no compression in the beginning) and hence not suitable for small files.

2.4.3 Lossy Image Compression

Lossy compression is a data encoding method that discards some data in order to compress. This results in compressed content that differs from the original content, but is similar enough to be of use. Since the reconstructed data differs from the original by losing some data, it is also known as irreversible compression.

In the case of image compression, fine details of an image can be removed to save bandwidth with minimal impact on the image quality. The result is an image that is degraded in quality, but usually still perceptible to the human eye.

Lossy techniques are commonly used for various multimedia data including streaming media (audio, video) and images.

Lossy compression typically results in higher compression ratios when compared to lossless techniques. The tradeoff that needs to be managed in designing these compression algorithms lies in balancing the level of compression and the acceptable quality of the reconstructed image.

The following chapter explores a number of these techniques in detail.

Chapter 3

Exploration of Compression Techniques

3.1 Lossless Compression Techniques

In the case of image compression, either lossless or lossy techniques may be employed to compress the data. Amongst the lossless class of techniques applied to images, a sample of the most commonly used methods are discussed below.

3.1.1 Huffman Coding

Huffman coding is an example of Lossless data compression. It was introduced in a paper published by David Huffman in 1952 [Huffman 1952] with the idea centered on exploiting the fact that in messages, characters are repeated with various frequencies. Huffman coding makes use of Variable Length Coding wherein each character in a message (example the letters a, b, c etc.) are represented by different codewords each of a varying length of bits (example "a" represented by 1 bit or 0, "b" represented by 3 bits or 111 and so forth for each letter). This differs from using a fixed length coding approach where each codeword that represents a letter is of fixed length (example 8 bits).

The most frequently occurring character will be assigned a code with the smallest bit length, thus reducing the expected number of bits per codeword (as the smallest codeword is assigned to the highest frequency character and the largest codeword assigned to the lowest frequency character).

From the simple example of using the letter "a" and "b" as defined above and assuming the letters occur with equal frequency, the fixed length coding would have an expected number of bits per codeword of 8 while the variable length coding would have an expected number of bits per codeword of 2 (which shows the improvement in storage efficiency under the Huffman technique).

Further, to avoid any ambiguities in decoding the message, Huffman makes use of Prefix Free Coding (where no codeword is a prefix of another code).

The following is a simple example to illustrate the Huffman coding technique with a binary tree.

3.1.1.1 Message to Encode

Suppose the following message string is to be sent over a network: $Message$. Each of the seven characters of the string is 8 bits and so the message occupies 56 bits. We compress this string using Huffman Coding which will result in a coded representation of the message. The first step is to create a frequency table.

3.1.1.2 Frequency Table

Using the message string, we count the frequency of each character and represent this in a frequency table. This is shown in Table 3.1.

Table 3.1: Frequency Table

Letter	**Frequency**
M	1
E	2
S	2
A	1
G	1

The frequency table is then used to construct the binary tree which follows.

3.1.1.3 Binary Tree

Using the information in the frequency table, we construct a binary tree. The frequency table is sorted by ascending frequency into a priority queue. The tree is constructed by starting at the end and working backwards. We start off by making each character a leaf node. Node 1 is created and a leaf created to the left with a character that has minimum frequency and another leaf created to the right with a character of second minimum frequency. In our example, the left leaf is A with frequency 1 and the right leaf is M with frequency 1. Node 1 is then assigned a value that is the sum of the leaf frequencies. In our example this is 1+1=2. Both A and M are removed from the priority queue.

Next, Node 2 is created. On the left of Node 2 we add a leaf with frequency less than the value of Node 1. This leaf is chosen from the remaining characters in the priority queue. In our example, this leaf is G with a frequency of 1. Node 2 is then assigned a value that is the sum of its leaves, namely G and Node 1. Based on our example, this is 1+2=3. The character G is removed from the priority queue leaving the last 2 characters in the queue, namely E and S.

Next, we create Node 3 that will have as leaves E and S. They both have frequency 2 and hence Node 3 will have a value of 4.

We then merge Node 3 into the rest of the tree. This is done by creating a new node, Node 4, that has as children Node 2 and Node 3. Node 2 is on the left because it has a lower value than Node 3. Node 4 is then assigned the value of 7, which is the sum of the values of its child nodes.

Next, for each non-leaf node we assign the value 0 to the outer edge and the value 1 to the right edge. The edge is where a leaf is joined to its parent node.

We then determine each characters code by starting from the top of the tree, and reading the edge values until we get to the leaf of choice.

The result of the process is shown in Figure 3.1.

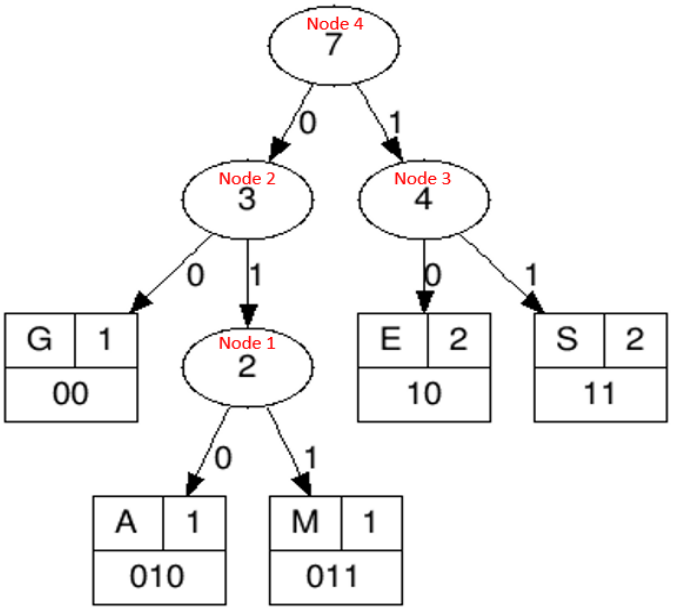

Figure 3.1: Constructed Binary Tree

3.1.1.4 Frequency Table and Huffman Code

Table 3.2 shows the final table with the frequencies, Huffman codes and size of each encoded character.

Table 3.2: Final Table

Letter	Frequency	Huffman Code	Binary Value	Size (bits)
M	1	011	3	3
E	2	10	2	4
S	2	11	3	6
A	1	010	2	2
G	1	00	1	1

Using this table, coded messages will be sent as binary sequences using the binary codes above. In our example the Huffman coding for the word "$Message$" is "0111011110100010".

To calculate the encoded message size, we multiply column 2 ("Frequency") and column 4 ("Binary Value") to give us the value in column 5 ("Size (bits)"). The binary value in column 4 is the decimal value of the binary number in column 3 ("Huffman

Code"). Summing the values in column 5, we find that the encoded message has a size of 16. This is less than the original message size of 56 bits.

To recover the original message from the encoded message, we read from the left of the binary code and simultaneously traverse through the binary tree, starting from the root node, following the edges that correspond to the binary digit being read. Once a leaf node is reached, the letter is recorded and we start from the root node again reading from the remaining binary digits.

3.1.2 Arithmetic Coding

Arithmetic coding is another example of an entropy coding technique where frequently occurring characters/symbols are represented by fewer bits (and less frequently occurring characters are represented by more bits). The technique represents data as a fractional value in the interval between 0 and 1 [Langdon 1984]. The technique is symbol-wise recursive, in that each iteration of the algorithm encodes (or decodes) a single symbol (example the letter "a" and then the letter "b"). As each iteration occurs, the interval is partitioned such that for distinct messages they never overlap. In applying Arithmetic Coding a model is defined that accurately represents the probability of a symbol occurring. The probability model and the encoder are independent, allowing for various models to be used that may be more accurate at representing the data (thus resulting in overall better compression). The technique follows the following general steps:

1. the initial interval is created between 0 and 1
2. for each symbol in the data, the initial interval is subdivided so that each sub-interval corresponds to a symbol in the data. The size of the sub-interval corresponds to the probability of that symbol in the data (this probability is determined by the model used).
3. the actual symbol that occurs next in the file (example the next letter in a message) is determined and the interval that corresponds to this symbol is then selected. This interval is treated as the initial interval (as in step 1 above) and subdivided as per step 2 for each symbol.

The result after iterating through the data is a single fractional value that represents the entire message.

3.1.3 Lempel Ziv Welch

Welch [1984] proposes an approach to compression that groups a sequence of symbols into strings and then converts these strings into codes. This is called the Lempel Ziv Welch technique ("LZW") The codes take up less space than the strings they represent and hence the data is represented with fewer bits. The key difference between LZW and the Huffman approach is that the mapping table is built as the algorithm runs through the message, whereas Huffman requires two passes through the data; the first pass providing the frequencies of symbols and the second pass actually encoding the data.

In comparison to Huffman, LZW builds a string table that maps strings (sequences of characters) to codes (as opposed to Huffman that maps individual characters to codes). Further, the dictionary that is used to encode the message does not need to be transmitted for decoding the message. The decompressor derives the dictionary as it decompresses the image.

3.2 Lossy Compression Techniques

As the name suggests, lossy compression techniques result in a loss of digital information. As a consequence of information loss, lossy techniques result in higher compression ratios than can be achieved with lossless compression. In the case of images, the techniques employed aim to provide a close approximation to the original image such that the differences are either very slight or even imperceptible to the human eye. These differences are measured by distortion metrics (examples include Mean Square Error, Peak Signal to Noise Ratio and Structural Similarity Index).

Typically lossy encoding follows three broad steps:

1. Transformation

   Input data is converted into transform vectors to decorrelate the pixel values. If we consider images, it is likely that there will be strong correlations between neighbouring pixels, and hence these pixels may be similar. By using transformations to decorrelate these pixels, we can represent regions of data with less information by discarding some information and retaining the pixels that provide the most information. As the correlation between samples decreases, the efficiency of the encoding increases. Spatial frequency theory is important here as it approximates how the human visual cortex operates. Spatial frequency refers to the number of times that a pixel value changes in a given image block (grouping of pixels). The human eye is less sensitive to higher frequency components (which is associated with fine details) and more sensitive to lower frequency components (which is associated with the more salient features of an image including shape). Hence by thresholding the spatial frequency, we can exclude information that would not normally be detected by an average human's eye.

2. Quantization

   In this step the (typically) continuous input data is mapped to a finite representation of the data. There are three forms of quantization used, namely

   (a) Uniform scalar quantization where the domain is subdivided into output values at equally spaced intervals

   (b) Non-uniform scalar quantization where the domain is subdivided into output values that are non-regular in spacing

   (c) Vector quantization

3. Entropy Coding

   In this step the quantised information is actually coded to yield the compressed image.

3.2.1 Transform Coding

Transform coding works well for data that is natural, such as audio or images as these can be typically represented as signals (sound in the case of audio and light in the case of images). The transformation function is usually invertible, such that the original input can be obtained without loss, however the quantisation step results in data loss (i.e. quantisation is not invertible). Common compression techniques based on transform coding are explored below.

3.2.1.1 Discrete Cosine Transform

DCT is a Fourier related transform proposed by Ahmed *et al.* [1974]. A typical Fourier transform decomposes a signal (or a vector of amplitudes) into the sum of different combinations of sine and cosine waves (each of different frequency). DCT decomposes signals into a combination of cosine waves only. Ahmed found that fewer cosine functions were needed to represent a typical signal, which as a result of fewer components results in better compression. Unlike the Discrete Fourier Transform (DFT) that uses imaginary numbers, the DCT uses only real numbers and hence requires only real arithmetic.

In image compression, DCT transforms pixels into sets of spatial frequencies, with high frequency representing fine detail. The full image is described by a combination of different frequencies (represented by cosine functions). In the quantization step, the less important frequencies are discarded or thresholded producing a result that is irreversible, or lossy. The JPEG compression technique used in this paper is based on DCT [Wallace 1992].

3.2.1.2 Wavelet Transform

Wavelet transforms are similar to Fourier transforms in that the image (or signal) is expressed as a combination of basis functions. However unlike Fourier transforms, the basis function in wavelet transforms contains both frequency and temporal information.

The sinusoid basis functions that characterise fourier transforms provide fine resolution in the frequency domain for the individual frequencies. The downside to sinusoids is that they are a function of continuous time and hence provide no localised information. So fourier based transforms can tell us what frequencies are present, but not where these are present. Hence in an image where the frequencies are changing throughout, it does not tell us where these changes occur.

Wavelet transformation expresses a signal as a combination of wavelets, which are waveforms of limited duration. These wavelets provide information on the frequencies present and also when/where these frequencies are present.

The advantage of this (wavelets) over the typical sinusoidal functions is that they do not require the input image to be broken into image blocks (as is the case with techniques such as DCT) [Hoffman 2012]. The correlation across these blocks cannot be completely eliminated. This creates artifacts in the transformed image. The need for "blocking" the input image comes as a consequence of the sinusoidal functions being infinite over time and not providing localised information.

Given that wavelets are expressed both in frequency and time, the input image does not need to be "blocked" as its basis functions can be of variable length (as opposed to infinite in the case of sinusoids

3.3 Summary of Compression Techniques

This chapter explored examples of both lossless and lossy compression techniques. While lossless compression allows for the perfect recovery of information, it results in lower compression than can be achieved with lossy compression. The balance between compression ratio and loss of information is an important decision when choosing a compression tool and should be considered in light of the task. Lossless compression would be ideal for medical image compression; however, our goal is to maximise compression to enable transmission of images over low bandwidth networks. As such, we focus the remainder of this paper on lossy techniques as we aim to maximise the compression ratios achieved.

The following section begins our exploration into the applicability of neural networks in compression.

Chapter 4

Deep Learning in Compression: Autoencoders

To understand what an autoencoder is, we first introduce the idea of neural networks and distinguish between supervised and unsupervised learning

4.1 Neural Networks and Deep Learning

Neural networks is an idea first proposed by McCulloch and Pitts in their 1943 paper [McCulloch and Pitts 1943] where it was positioned that a logical expression can be expressed as a series of neurons (arranged together to form neural nets).

Artificial Neural Networks ("ANN's") are, simplistically, rudimentary electronic representations of the biological neural structure as found in the brain. Our current understanding of the biological brain is that it is fundamentally composed of neurons. These neurons receive stimuli and produce some type of response based on this input [Anderson and McNeill 1992]. The ANN is modelled in a similar manner, with its fundamental components being artificial neurons. These neurons may be arranged into layers, with each layer,or perceptron, consisting of many neurons. Each neuron may be connected to every other neuron in the next layer (fully connected) or not.

An input $x \in R$ is received by a neuron (or perceptron in the case of a layer of neurons) and multiplied by a weight, $w \in R$ to produce an output. If we consider each of the series of inputs and weights as vectors, then the output is represented as a dot product between the input vector X, and the weight vector W where X,$W \in R^n$ and n is the size of the input and weight vectors. This output is then passed through some type of activation function which then converts the output into a final value $y \in [0, 1]$.

In the instance where we have known inputs and outputs, the goal is to find the weights that map these inputs to the correct output. The process typically involves two steps, forward propagation and back propagation.

In the forward propagation step, the inputs are multiplied by some weights (which may be randomly initialized at outset) and passed through the activation function. the output is compared to the true output and if it does match, the weights will need to be updated.

If the result of forward propagation results in an output that does not match the true output, the process looks backwards and tries to modify the individual weights of the neurons until it gets closer to the desired output. This is achieved by optimising a given loss function using techniques such as gradient descent.

The process of forward and back propagation are continued until the calculated output matches the true output (or the loss function is minimised). In this way, the neural net learns the correct weights that provide the mapping from input to final output.

4.1.1 Supervised Learning

In the ANN example provided above, the description is given with regard to supervised learning. This is the case where the weights are learned by being provided known outputs. Because the actual outputs are known, the learning process is said to be supervised in that we teach the net what output we actually want. In practice a data set of labelled data is obtained (the known outputs that we are trying to find of which an example being the presence of red blood cells in a whole slide image). During training, the ANN is fed an image and produces a vector of scores (binary in the case of detecting the presence of red blood cells or not). An objective function is defined that measures the error between the output scores and the desired output. This objective function is optimised (error is reduced) by having the weights associated with the neurons altered such that we minimise the error function. Depending on the scale of the problem, various optimisation techniques are used with a commonly used method being Stochastic Gradient Descent.

4.1.2 Unsupervised Learning

In unsupervised learning, the network is provided with inputs but not with the desired outputs. The system must then decide what features to use to group the input data.

4.2 Autoencoders

An autoencoder is an ANN that is designed to learn a lower dimensional representation of an input. It is a nonlinear approach to dimensionality reduction that uses an adaptive, multi-layer encoder network to transform the high dimensional data into low-dimensional code and a similar decoder network to recover the data from the code [Hinton and Salakhutdinov 2006].

Figure 4.1 below shows graphically the layers of an autoencoder that transforms a high dimensional input into a lower dimensional coded layer and the layers that recover the original dimensionality from the coded layer to yield the output.

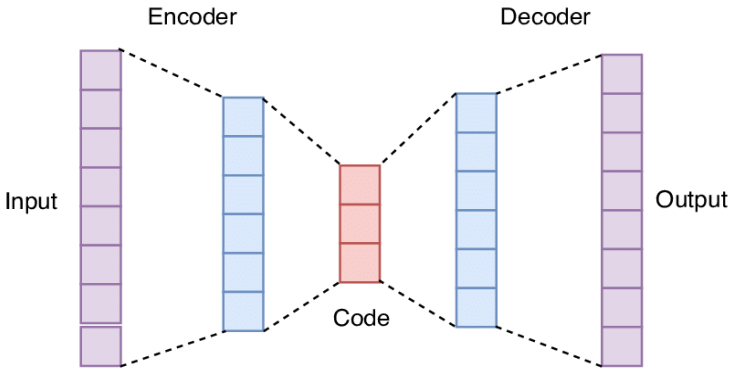

Figure 4.1: Basic Autoencoder Architecture

It is an example of an unsupervised learning algorithm (where data is unlabeled) that applies back-propagation to set the target values equal to the input values. As a consequence, the dimensions of the input and output are the same.

The autoencoder tries to learn a function $h_{w,b}(x) \approx x$ (i.e. the output is approximately equal to the input). Upon first inspection, it seems that the solution is (trivially) to find the identity matrix. However the purpose of the autoencoder is to find a more efficient representation of the data. To achieve this successfully, we restrict the dimensions of the hidden layers to be less than the dimensions of the input creating an information bottleneck [Tishby *et al.* 2000].

Given that the learned representation is based on a given set of data, autoencoders are data specific and do not generalise well to other types of unseen data. In the case of blood samples, while a particular autoencoder may learn to represent red blood cells well due to the data set used, it may not be able to do this well for dogs or cats (or even white blood cells in a less extreme comparison).

Autoencoders are a form of lossy compression technique where the decompressed image is not equal to the input image (and as a consequence some information is lost).

In contrast to the traditional techniques explored above, autoencoders are learned automatically and hence do not require specific feature engineering. This makes it practical in automatically training an instance of the algorithm to work on a specific set of input data.

4.3 Basic Autoencoder Architecture

The basic architecture is comprised of 3 parts [Hinton and Salakhutdinov 2006]:

1. an Encoding Network that compresses the input into a latent space representation with reduced dimension

2. A coded layer that represents the part of the network that contains the compressed (encoded data) that is to be fed into the decoder

3. a Decoding Network that decodes the latent representation (encoded layer) into the dimension of the original input providing the lossy reconstruction of the original input image.

The encoder function Φ maps the input data $X \in R^n$ to latent space $F \in R^m$ where $m < n$:

$$\Phi : X \rightarrow F \tag{4.1}$$

The decoder function Ψ maps the latent space representation F to the output X. As described by Hinton and Salakhutdinov [2006] the output is set to the input.

$$\Psi : F \rightarrow X \tag{4.2}$$

The goal is to find the functions that minimise the difference between the input and the output:

$$\Phi, \Psi = argmin_{\Phi,\Psi} |X - (\Phi \circ \Psi)X|^2 \tag{4.3}$$

The encoding function can be represented by the standard artificial neuron function passed through an activation function (where z is the latent variable and x is the input):

$$z = \sigma(Wx + b) \tag{4.4}$$

The Decoding function can be represented similarly, however with different weights, biases, and activation functions. The input will be the latent variable z and the output will be an estimate of the input:

$$\hat{x} = \sigma'(W'z + b') \tag{4.5}$$

The loss function is then defined in terms of these encoder-decoder network functions and a suitable back-propagation algorithm (such as gradient descent) is used to learn the weights for the mappings:

$$L = |x - \hat{x}|^2 \tag{4.6}$$

The paper by Hinton and Salakhutdinov [2006] proposes the use of multiple hidden layers to better represent the data in that each layer of features captures strong, high order correlations between units in the layer below. Hinton concludes that this is an efficient way to progressively reveal low-dimensional, nonlinear structure.

It is useful to note that while the encoder can be learned using a multi-layer perceptron it can also be learned using a convolutional neural network which implies that the architecture need not be fully connected [Cheng *et al.* 2018]. The same holds true for the decoder network which is usually symmetric to the encoder network and may use either a fully connected architecture or a transposed convolution [Dumoulin and Visin 2016].

4.4 Convolutional Autoencoders

The preceding sections introduced ANN's and the basic idea of autoencoders, however to work efficiently on images we frst consider convolutional neural networks ("CNN") and then explore their implementation in autoencoders.

4.4.1 Convolutional Neural Networks

Typically, images are large data items comprised of many pixels or input variables. Using a fully connected ANN would result in many parameters needing to be learnt and require an extensive data set to train the network successfully. Further, images are typically two-dimensional in shape and contain pixels that are spatially close together and highly correlated. By vectorising these two-dimensional images, as would be required in traditional ANN's, we lose spatial and temporal information present in the image. While a large enough network can learn these, it would require a significant amount of training data.

Introduced by LeCun *et al.* [1998], CNN's are designed to capture the spatial and temporal dependencies present in image data by applying filters across the image. These filters enable the local features (such as edges) to be extracted more accurately in the image. The features learnt by one layer are used in subsequent layers to detect higher-order features. Weight sharing allows the CNN to create feature maps that can be used to detect the same feature at all locations in the input image. The result is an architecture that requires less training data than ANN's and better fits the image data being processed.

There are three types of layers that comprise a CNN, namely: convolutional layers, pooling layers and fully connected layers.

A convolutional layer is a two-dimensional grid of neurons. Each neuron in the grid takes inputs from a two-dimensional segment of the previous layer (also two-dimensional). This is achieved by convolving a filter over the previous layer (or input image). The result is a feature map. Each convolutional layer may be comprised of several feature maps as a result of using multiple filters in one layer.

A pooling layer creates a downsampled version of the previous layer by taking rectangular sections of the previous layer and transforming it into a single value. This process may be done by calculating an average or the maximum of the values in the rectangular sections. In the case of using the maximum value, this process is called max-pooling.

The fully-connected layer is typically found in the final layers of a CNN and is responsible for performing the classification. Every input from one layer is connected to every activation unit in the next layer.

When convolving a filter over the image, there are three hyper-parameters that influence the size of the feature map, namely: zero-padding, depth and stride.

Zero-padding adds a number of layers, P, of zeros onto the border of the input, allowing the filter to convolve over the padded image such that the input size and output size are the same.

Depth corresponds to the number of filters that are used in each layer. Each filter learns a different feature, such as vertical lines or edges. When using multiple filters, we obtain a feature map volume.

Stride refers to how the filter is passed over the image. When stride is 1, the filter is moved one pixel at a time. When the stride is S, the filter moves by skipping S pixels at a time, which results in spatially smaller feature maps.

The size of the output feature map can be calculated using Equation (4.7).

$$O_{size} = \frac{(I_{size} + 2P - F_{size})}{S} + 1 \tag{4.7}$$

where O_{size} is the size of the output feature map, I_{size} is the size of the input, F_{size} is the size of the filter, P is the number of zero-padding layers added to the border of the image and S is the stride.

Similar to ANN's, the weights of the filters are learned via forward and back-propagation. In forward propagation, we calculate the intermediate value $z_{ij}^{[l]}$ for each neuron in the layer l of the CNN as per Equation (4.8)

$$z_{ij}^{[l]} = \sum_{a=0}^{m-1} \sum_{b=0}^{m-1} \omega_{ab} y_{(i+a)(j+b)}^{[l-1]} \tag{4.8}$$

where ω is a filter of size $m \times m$, and y_{ij} is an activation unit calculated as per Equation (4.9)

$$y_{ij}^{[l]} = g^{[l]}(z_{ij}^{[l]}) \tag{4.9}$$

where g is the activation function. If we assume that layer $l-1$ has a size of $N \times N$, that there is no zero-padding and we use a stride of 1, then the above will result in layer l being of size $(N - m + 1) \times (N - m + 1)$.

Weights are updated in the back-propagation step wherein the loss function, E, is differentiated with respect to the variables introduced above. Equation (4.10) shows the simplified differential equation.

$$\frac{\partial E}{\partial y_{ij}^{[l-1]}} = \sum_{a=0}^{m-1} \sum_{b=0}^{m-1} \frac{\partial E}{\partial z_{(i-a)(j-b)}^{[l]}} \omega_{ab} \tag{4.10}$$

A series of forward and back-propagation steps are performed until the loss function is minimised. The loss function is chosen to suit the task being performed. In the case of multi-class classification tasks, categorical cross-entropy is commonly used. When training autoencoders, mean squared error is typically used.

4.4.2 Using CNN's in Autoencoders

The structure of a convolutional autoencoder is similar to the basic architecture that uses ANN's, however instead of using fully connected layers, the Encoding Network and Decoding Network use CNN's.

The Encoder Network compresses an input image into a latent space representation with reduced dimension compared to the input. This latent space can be made using a

fully connected layer; however, the increase in parameter makes this computationally inefficient. Hence, the latent space is typically a rectangular volume in shape and is a convolutional layer itself.

The Decoder Network uses an inverse convolution operation to up-sample the latent space to reconstruct the input image and recover its dimensions. As with the basic autoencoder architecture, we follow a similar process to learn the network's weights that will minimise the difference between the output image and the input image.

4.5 Application to Compression

New media formats, changing hardware technology and the diverse requirements for content usage has created a need for compression technology that is more flexible than the currently used codecs. Autoencoders show promise in this space due to their ability to be learned automatically with limited feature engineering. Cheng *et al.* [2018] shows how using Convolutional Autoencoders for compression shows promise when compared against JPEG2000. However, due to the non-differentiability of compression-loss (brought about in the quantization step), autoencoders are difficult to optimise. Theis *et al.* [2017] show that changes to the loss function can allow autoencoders to be trained that can compete with existing standards such as JPEG2000.

Theis *et al.* [2017] present a lossy image compression algorithm based on Autoencoders. The paper explores a method for optimising the rate-distortion tradeoff, dealing with the non-differentiability brought about by quantization and an approximation method for dealing with the non-differentiable cost of coding the generated coefficients.

The architecture described in the paper consists of an encoder f, decoder g and a discrete probabilistic model Q. Both the encoder and decoder use convolutional neural networks which has proven to be efficient when dealing with images [LeCun *et al.* 1998]. The distribution Q is used for entropy coding by assigning a number of bits to representations based on their relative frequencies. Their method aims to minimise the tradeoff between representing the data with few bits and having low distortion:

$$\text{Tradeoff} = -log_2 Q([f(x)]) + \beta d(x, g([f(x)])) \tag{4.11}$$

The first term represents the number of bits and the second term represents distortion. The square brackets represent quantization by rounding to the nearest integer and d represents the distortion introduced by coding and decoding. Due to the non-differentiability of Q and the quantization, this equation cannot be optimised using gradient descent techniques.

To deal with the issue of quantization, the authors propose replacing the derivative of the rounding function in the backwards pass of back-propagation with the derivative of a smooth approximation, r, such that

$$\frac{d}{dy}[y] := \frac{d}{dy} r(y)) \tag{4.12}$$

Only the derivative of the rounding function is replaced by the smooth function which means that quantization is still performed in the forward pass.

Ballé *et al.* [2016] propose a solution to this non-differentiability by replacing the quantization by additive uniform noise as shown in equation (4.13)

$$[f(x)] \approx f(x) + u \tag{4.13}$$

In dealing with the non-differentiability of Q (a discrete probability distribution) the authors approximate it with a continuous, differentiable function.

4.6 Summary

This chapter introduced the idea of neural networks and the concepts of supervised and unsupervised learning. Our discussion focused on autoencoders, an unsupervised ANN architecture useful in learning a lower-dimensional representation of an input. We extended this discussion by considering CNN's and their advantage over ANN's when dealing with images due to their ability to capture spatial and temporal dependencies. Building on the basic autoencoder architecture, we explored how a convolutional autoencoder may be constructed and how we may apply this to the task of compression.

The following chapter considers the need for compression in medical imaging and describes the features of this data that support the investigation into autoencoders' use as a compression tool.

Chapter 5

Medical Image Compression

Medical imaging has a significant impact on the diagnosis of diseases. However, due to the size of medical image data, storage and transmission prove challenging. This is more pronounced in geographical regions that are underserved in terms of telecommunications infrastructure.

Given the cost of incorrect interpretation, lossy compression techniques are viewed skeptically by medical professionals as physicians feel they cannot trust the output that may introduce unexpected artifacts into medical images [Ström and Cosman 1997].

The dilemma lies in the fact that lossless compression techniques may not provide a significant advantage for transmission and storage, and lossy techniques may lose information crucial for diagnosis [Zuo *et al.* 2015].

A close examination of the data leads to a key insight; in medical images, only a small portion of the image may be diagnostically useful. Compression algorithms that deliver lossless compression in regions of interest (ROI) and lossy compression elsewhere exploit this feature of medical image data to provide suitable compression levels but still maintain physician trust in the data. Zuo *et al.* [2015] explore ROI based compression algorithms based on the traditional approaches of transform coding techniques.

If we consider medical image data further, we notice that the images are specific to a particular practice within medicine. These images may be mammograms, chest x-rays, prostate tissue samples or blood slide samples (to name a small subset). The fact that these images are niche implies that there could be some form of regularity in their structure that could be exploited for compression. Autoencoders may be useful as compression tools as the data is very specific, so they may be trained to provide compression for this very specific use case.

5.1 Application to Medical Blood Slide Images

The subject matter of this paper is focused on the compression of blood slide image data. If we consider blood images, there are a limited number of items of interest that a haematological analysis must consider. These components are grouped into four main classes:

1. Plasma: the liquid component of blood

2. Erythrocytes: Red Blood Cells make up 40-45% of the volume of blood and are bi-concave disk shaped cells.

3. Leukocytes: White Blood Cells make up around 1% of blood volume and are further categorised into Neutrophils, Eosinophils, Basophils, Lymphocytes and Monocytes each with different morphological characteristics.

4. Platelets: small fragments of cells that aid in coagulation

The blood components above, except in abnormal cases, typically show repeated, regular features in terms of morphology, size and colour. Variation may be introduced by the quality of camera and lighting used by the microscope which may be adjusted by pre-processing. However, the fundamental components of blood can largely be expected to be regular.

The image below shows an example of a blood slide image showing erythrocytes and leukocytes (with Wright's stain making the leukocytes appear purple)

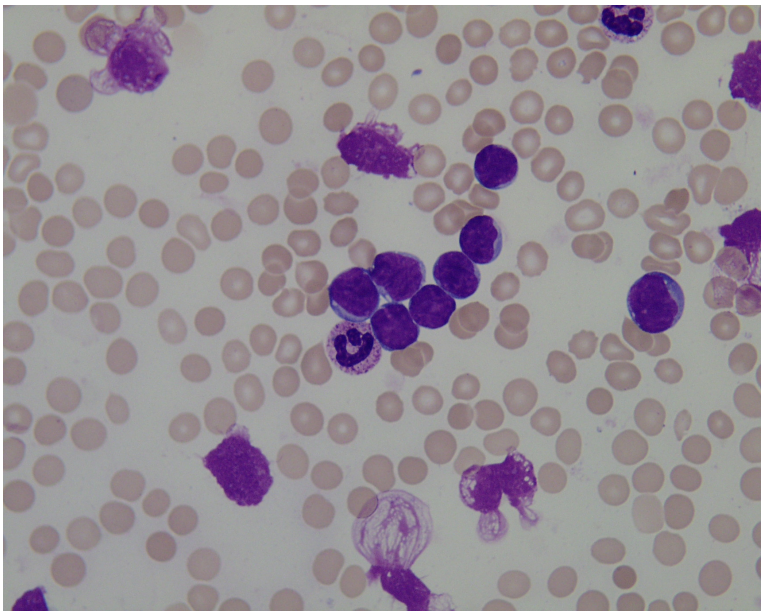

Figure 5.1: Blood Slide Image Showing Erythrocytes and Leukocytes

The image highlights the regularity of blood cells and that between the cells is effectively empty space. These regions that contain no blood components or artifacts contribute to the overall size of the image as bits are required to code each pixel in these regions.

By using a suitable autoencoder architecture, we may be able to train the network to identify the individual components of blood and using this knowledge to reduce the dimensionality of the data by removing the redundancy that exists between components.

The following chapter describes the methodology followed in investigating autoencoders as a viable compression tool in medical imaging.

Chapter 6

Research Methodology

6.1 Hypothesis

Machine learning-based autoencoder techniques perform better than traditional compression techniques in the specific case of blood sample images when measured on the basis of maximising compression ratio and quality.

6.2 Research Questions

To understand the efficacy of using autoencoders as compression tools for medical images, it is important to research the implications of any noise that is introduced into the data due to the lossy nature of the technique. As such, the following questions have been researched as part of this paper:

1. Can an autoencoder be trained to learn the regularity of the blood slide images and exploit this for improved compression rates?
2. How does the architecture of an autoencoder affect its performance in terms of compression ratio and reconstruction quality?
3. Do autoencoders perform better than JPEG compression? If so, for a given compression ratio, how similar is the original image to the reconstructed image?
4. Does the use of compression impact the classification accuracy of a downstream neural net trained to detect particular blood components? (i.e. assuming a net is trained on a given set of images, is its classification accuracy affected by using compressed and reconstructed images of the same data set)
5. If trained on reconstructed images, to what extent can the downstream classifier recover its classification accuracy after the test set has been compressed?

6.3 Approach

In exploring the application of compression to blood images, we apply and compare various autoencoder architectures' performance with that of JPEG compression.

Comparison of the methods is made with reference to compression ratio, SSIM and PSNR as benchmarks and their performance impact on a downstream classification task.

Specifically, our approach to answering the first question entails collecting relevant image data, designing a suitable autoencoder architecture, and subsequent training of this autoencoder by providing it with the data. The output is tested using the PSNR and SSIM metrics to measure quality and the compression ratio to measure its efficacy in compressing the original image.

As part of our research, we tested two distinct autoencoder architectures, and the performance is compared with reference to the benchmarks above. The best of these autoencoders is selected and used for the classification task.

Given that JPEG is a widely used compression standard, we test our best autoencoder's performance against JPEG using the selected benchmarking criteria of quality and compression ratio.

Our fourth question explores the idea of classification accuracy and how this is affected by using a compressed image. We use a classification neural net trained on original data to identify blood artefacts to explore this. We then compress those images and pass them through the same neural net, and test accuracy. Intuitively we consider the classification net to be a proxy for a human doctor and determine what effect compression has on interpreting the image.

Our fifth question relates to enhancing accuracy, exploring the possibility of training the classification net to deal with the errors introduced through compression. To do this, we adjust the classification net by training it on reconstructed images with the same labels as their originals. The classification accuracy is then tested by passing through an original and reconstructed version of test images, thus comparing the classification accuracy.

6.4 Data

The data used for this research report was obtained from Kaggle's public repositories and private data provided by Vitruvian Medical Diagnostics (Pty) Ltd ("VMD"). VMD has requested that its private repository not be made available for public use and will remain its sole property until such time the company provides written consent to make the data freely available.

The Kaggle data consists of 12500 augmented images of blood cells created using an original set of 410 images. The blood images are grouped into four classes based on the four most common leukocyte types: Neutrophils, Lymphocyte, Monocyte and Eosinophils. These classes are relevant when training a classifier to test its ability to perform on reconstructed images. The data provided was already balanced, with each class containing approximately 3,000 images. The bar graph in Figure 6.1 below shows the distribution of data as used for training and testing:

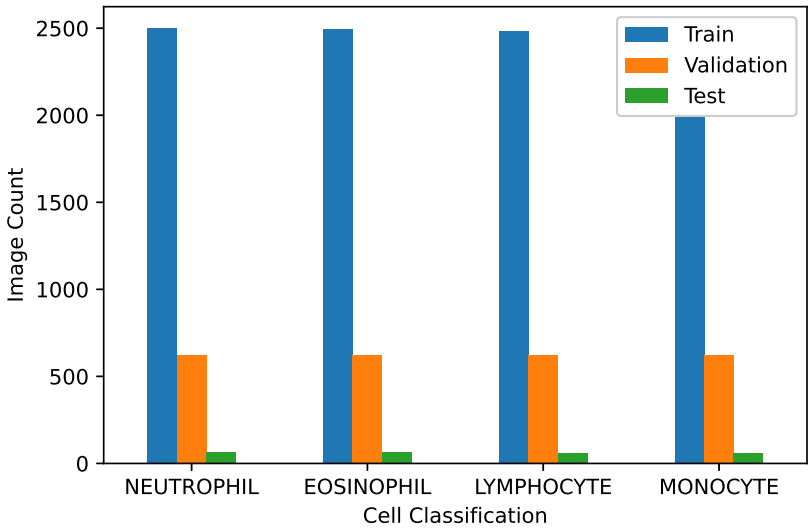

Figure 6.1: Distribution of Kaggle Data

For classification, it is assumed that the images were labelled with absolute accuracy. 250 images were used for the Test Set. The images in the Kaggle dataset were of size 256x320, each with 3 channels. Figure 6.2 shows a sample of images from the Kaggle dataset showing an example of each class of leukocyte.

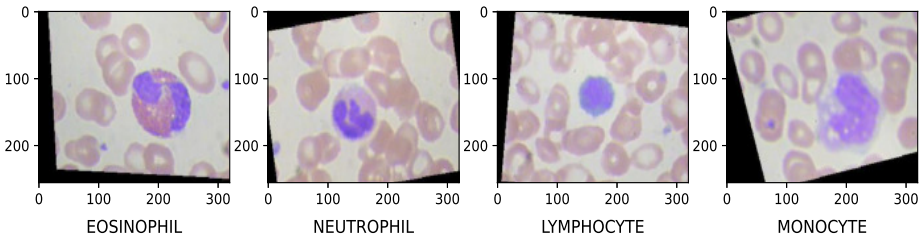

Figure 6.2: Example of Kaggle Images

The VMD data was originally obtained by scanning blood slides with a Cellavision DM96 digital microscope. The images provided were a random sample of 38,686 images of the four most common leukocyte types; however, these were unlabelled. As part of our experimentation, this data was split into a distinct Training Set and Valida-

tion Set using an 80%/20% split between them, respectively. 787 distinct images were used for the Test Set. Due to the lack of labels, the data was used in training the autoencoders only, and not classification experiments were performed. The images in the VMD dataset were of size 256x256, each with 3 channels. Figure 6.3 shows a sample of unlabelled blood cell images.

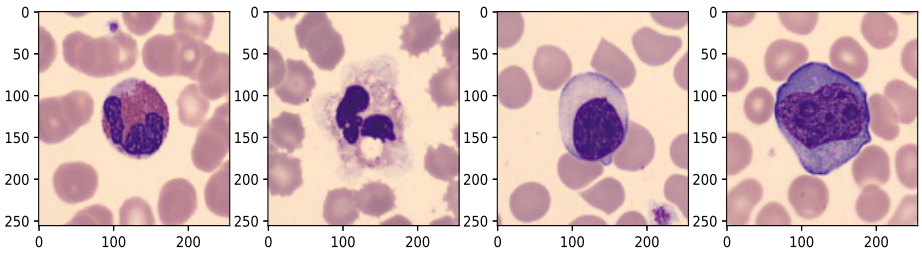

Figure 6.3: Example of VMD Images

With both the Kaggle and VMD datasets, the data provided were in JPEG format. Using compressed data adds some variance to the results as some quality loss is introduced through the initial compression. This is especially true when comparing the performance of the autoencoders against JPEG compression (as the images that are already compressed using JPEG were compressed again using JPEG).

Given that the same data is used as a base for all models, the error introduced is consistent for both the autoencoder and JPEG experiments, and hence the comparative results are still valid.

6.5 Autoencoder Architectures

In our approach to exploring the research questions and testing our stated hypothesis, we experiment with various autoencoder architectures and test them for different compression levels.

We test two broad architectures:

1. a simple autoencoder architecture ("SimpNet"), where the latent space is shaped to provide approximately 2x, 5x, 10x and 15x compression levels

2. an autoencoder architecture where the encoder portion is based on the VGG16 architecture as introduced by Simonyan and Zisserman [2014] and using pre-trained weights based as a starting point ("VggNet"). Again the latent space is shaped to provide the approximate compression levels that we require.

For each architecture, we create an autoencoder to cater for a specific compression level. Each autoencoder is comprised of two networks, namely the encoder network ("Encoder") and the decoder network ("Decoder"). The Encoder's purpose is to reduce the dimensionality of the image and represent it in the encoded or latent space. The Decoder takes as input the latent space and produces a reconstructed image representing the original image. The key difference between each autoencoder is the latent space's size, which determines the level of compression. As such, for each compression level required, a distinct autoencoder is trained.

The size of the latent space required for each compression autoencoder is calculated as follows:

$$L = \frac{(hwc)}{(bx)} \tag{6.1}$$

Where L represents the number of parameters in the latent space. The numerator is the memory size of the image in bits, calculated by multiplying the height (h), width (w) and the number of colour channels (c) together. The denominator is the compression level required (x) multiplied by the number of bits required to store each real-valued parameter in the latent space (b).

6.5.1 SimpNet Architecture

SimpNet is a simple autoencoder architecture comprised of 12 layers. The Encoder and Decoder layers are symmetrical, and each contains 6 layers. The input to SimpNet is aligned to the size of the RGB images considered, i.e. 256x320x3 for the Kaggle data and 256x256x3 for the VMD data. No material pre-processing was performed during training, apart from scaling image values to fall in the range [0,1].

The input image is passed through a series of convolutional layers, each using a small filter size (2x2) and a stride of 2 to downsample the image features after each convolution. Strided convolutions were chosen to ensure the weights associated with downsampling are learned instead of using a fixed operation such as MaxPooling. At each convolution, we increase the number of filters.

The Decoder network is designed to be symmetrical to the Encoder; however, we use transpose convolution operations to upsample the feature space and recover the input dimensions. For each intermediate layer of the SimpNet, we use a Rectified Linear Unit ("ReLu") activation function.

Figure 6.4 graphically represents this architecture for the SimpNet. The latent space is configured to provide a 2x compression ratio. All other SimpNet architectures may be found in Appendix A.

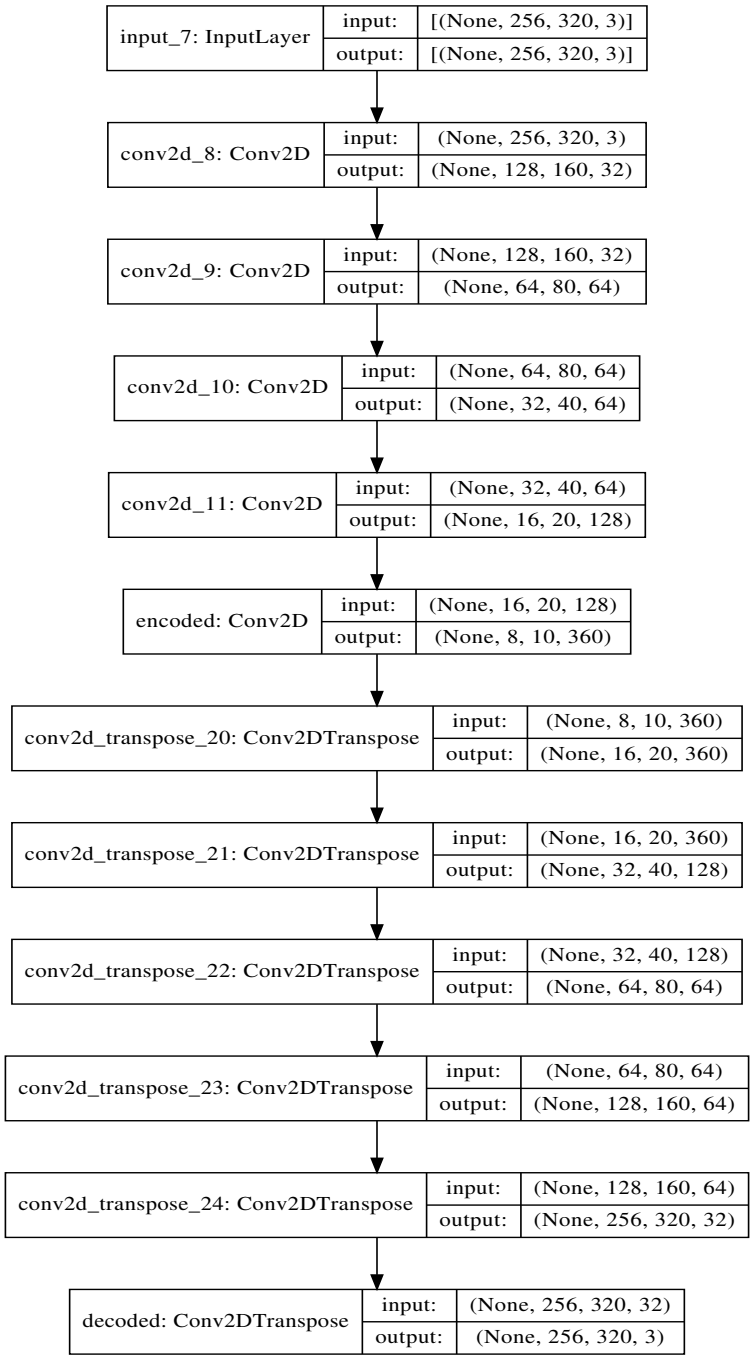

Figure 6.4: Architecture for SimpNet with a compression ratio of 2x

6.5.2 VggNet Architecture

The VGGNet is comprised of an encoder network that follows the model structure proposed by Simonyan and Zisserman [2014] however the fully connected layers are excluded and replaced with an additional convolutional layer that provides the latent space dimensionality required to achieve our desired compression ratio. The decoder network follows a similar structure as used in the SimpNet.

In training, the Encoder was loaded with pre-trained ImageNet weights, and all layers were allowed to be trainable.

Figure 6.5 graphically represents this architecture for the VggNet. The latent space is configured to provide a 2x compression ratio. All other VggNet architectures may be found in Appendix B.

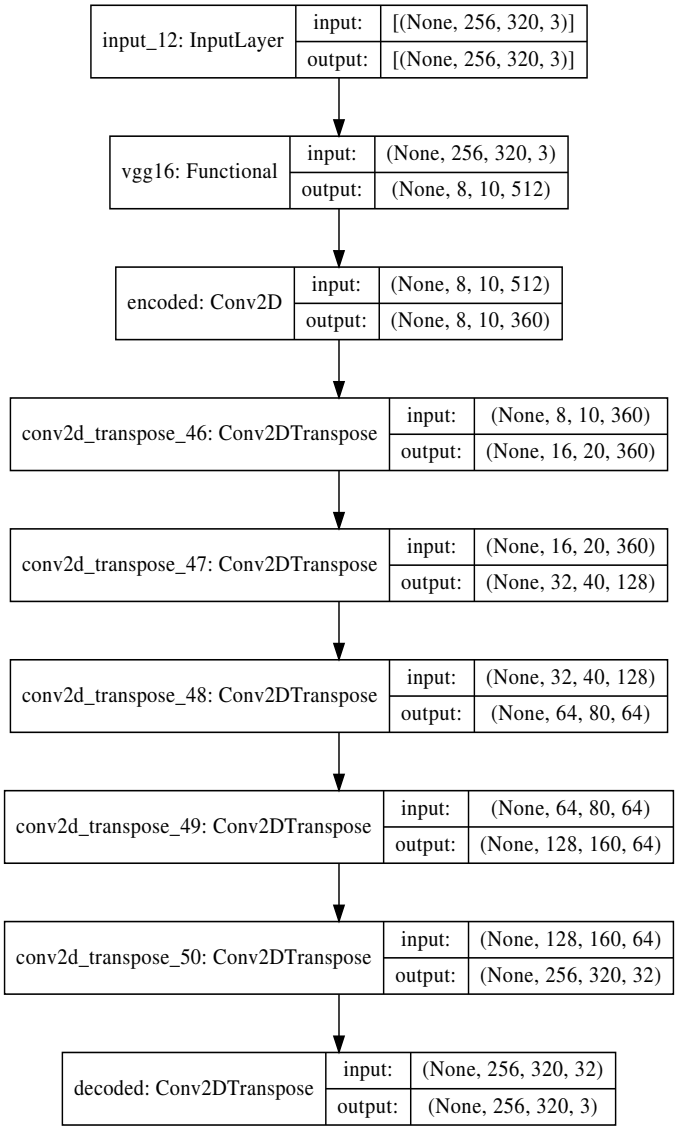

Figure 6.5: Architecture for VggNet with a compression ratio of 2x

6.6 Quality Metrics

The efficacy of compression techniques are determined across three metrics:

1. Compression Ratio;
2. PSNR; and
3. SSIM

For a given compression ratio, the PSNR and SSIM are calculated for reconstructed images produced by JPEG and the various autoencoder nets explored in this paper.

6.7 Classification

To understand these compression techniques' real-world applicability, we need to explore how classification accuracy is impacted if a reconstructed image is used instead of the original image. As a means of elucidation, we consider the case where an image is captured in a remote area but needs to be transmitted to a doctor in a major city for analysis. Digital transmission may require the image to be compressed before sending. If the reconstructed image received by a doctor is of poor quality or details relevant to making a diagnosis are missing, the result can be catastrophic for the patient.

As such, we consider training a classification net on the original data and treat this as our "ground-truth" in classification. Effectively, this net replaces the role of a doctor. We then test its classification accuracy on the reconstructed images.

The classification experiments were performed using the Kaggle data only as the VMD dataset was unlabelled.

6.7.1 Classification Net Architecture

To build a classification net, we use the VGG16 architecture and load the ImageNet weights. For training, we keep the initial twelve layers fixed and allow the last four to be trainable. We then add three additional convolutional layers and a MaxPooling layer before the fully connected layers. We also add normalisation layers and dropout layers to assist with regularisation and assist training. The final layer is a softmax output with four categorical classes. The loss function used was categorical cross-entropy.

A summary of the model is shown in Figure 6.6.

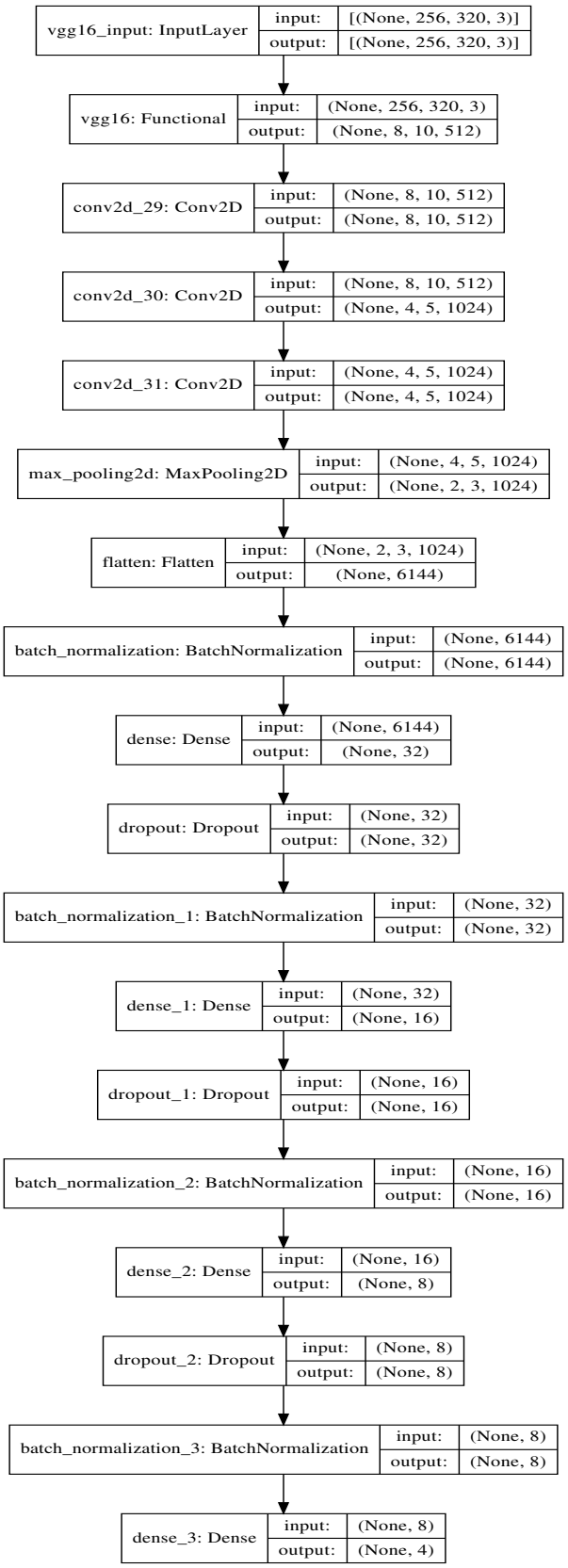

Figure 6.6: Architecture for Vgg16 based classifier

Chapter 7

Results

The experiments were performed separately using the Kaggle and VMD datasets. Due to the VMD data being unlabelled, no classification based experiments were carried out on this dataset. The results obtained are discussed below.

7.1 Compression Experiments: Kaggle Data

Compression is tested at the following levels using autoencoders:

1. 2X compression ratio
2. 5X compression ratio
3. 10X compression ratio
4. 15X compression ratio

Using JPEG, the maximum compression ratio achieved was 10X, using the lowest quality setting.

7.1.1 SimpNet

The autoencoders were trained over 50 epochs, a batch size of 50, using ADAM optimiser and a mean-square error loss function. To provide better convergence we make use of learning rate decay with a starting learning rate of 0.001 and a final learning rate of 0.000001.

7.1.1.1 Training Curves

The results below show that the models learn with reasonable accuracy the general structure of the blood images. We notice that as the compression level required increases, the accuracy of our model decreases. This makes intuitive sense because, in lossy compression, we expect higher compression to be associated with more loss in detail. If we consider the shape of the training curves, we notice that our training and validation loss curves tend to converge smoothly; however, our accuracy curves

converge but still exhibit some volatility. The combination of results indicates that our model is learning well, albeit with some element of overfitting. This imperfect convergence is expected as the data set is quite limited in size. The data we used was compressed and limited in volume, with 12500 augmented images created from 410 initial images. Despite this, the limited dimensionality of blood images seems to be exploited well by our autoencoders.

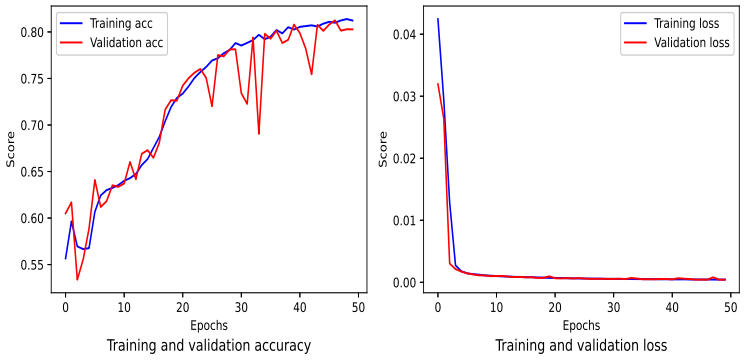

Figure 7.1: Training Curves for SimpNet 2X Compression

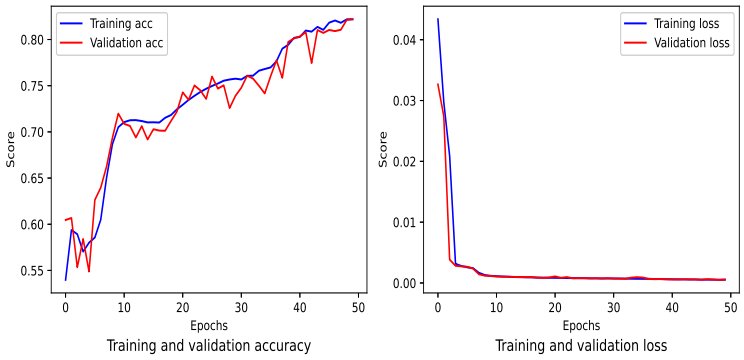

Figure 7.2: Training Curves for SimpNet 5X Compression

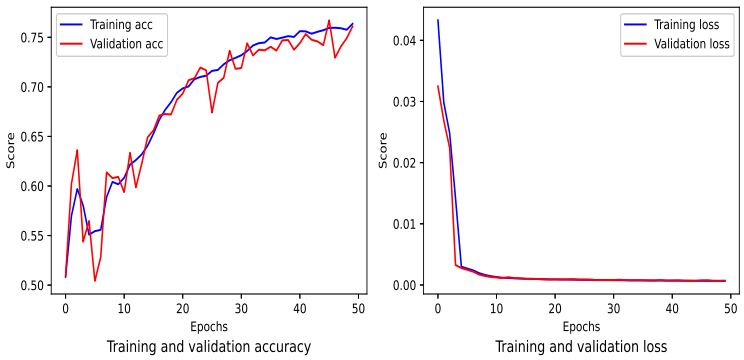

Figure 7.3: Training Curves for SimpNet 10X Compression

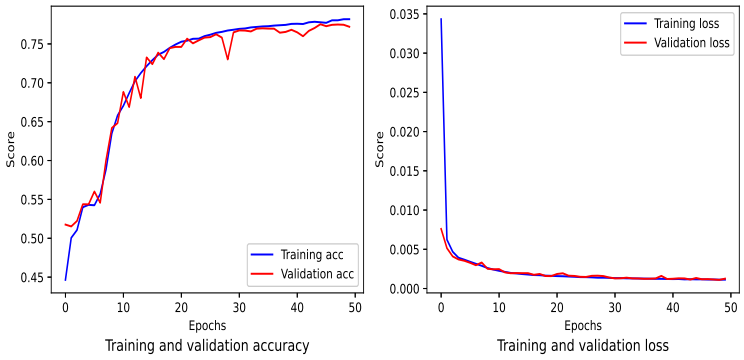

Figure 7.4: Training Curves for SimpNet 15X Compression

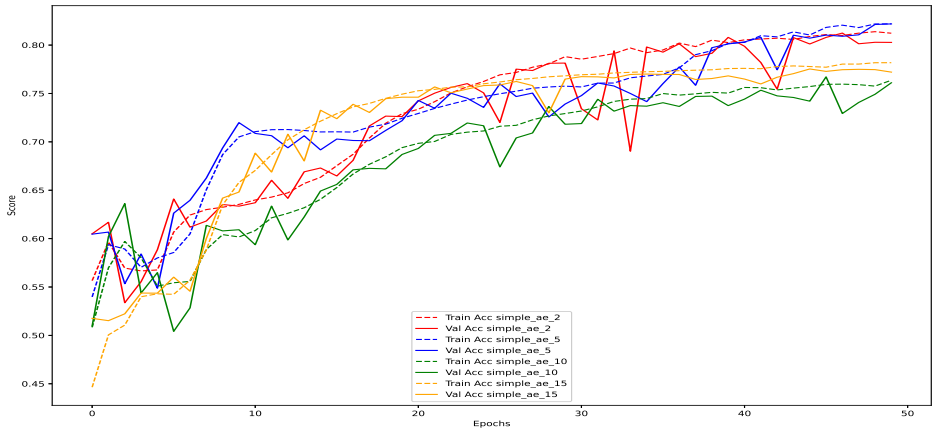

Figure 7.5: Summary of Training Accuracy Curves for SimpNet

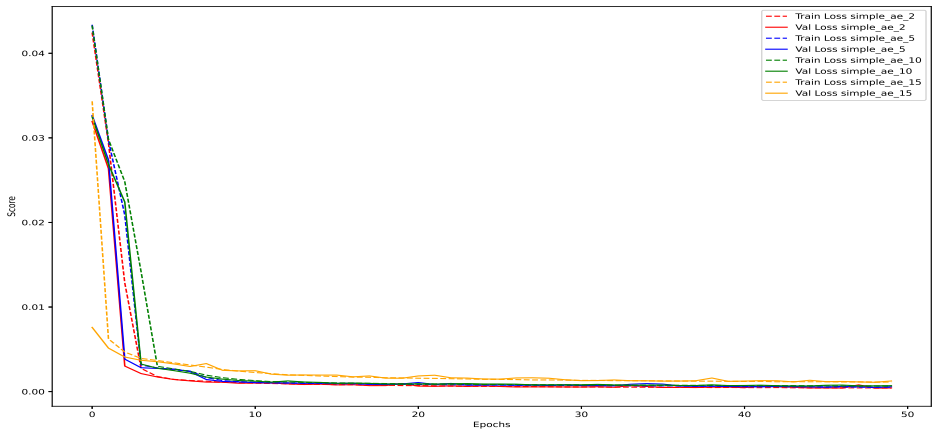

Figure 7.6: Summary of Training Loss Curves for SimpNet

7.1.1.2 Predicted Images

The sample of images below shows an original image that has been compressed and subsequently reconstructed by our autoencoders. As we progress from low to high compression (left to right), there is noticeable detail lost with the 15X images being more "blurry" in appearance. In particular, we notice that the minor stain artefacts are lost as compression increases. This shows that the autoencoder has prioritised the preservation of spatial information over spectral information. The spatial information is crucial as it is most relevant for identifying cell shape and identifying the type of blood cell. The colour (from staining) serves to help humans differentiate cells under a microscope by making white blood cells visible. As is noticed from the output images, the general shape of the blood cells is maintained, allowing the cell morphology to be determined visually.

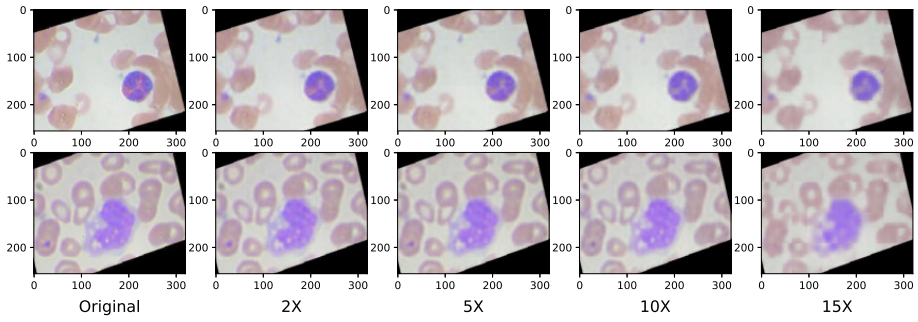

Figure 7.7: SimpNet Predicted Images at Each Compression Level

7.1.1.3 Quality Metrics

Figure 7.8 shows the performance of the SimpNet under PSNR and SSIM. For each image in the test set, we calculate the PSNR and SSIM values by comparing the original test image to a compressed version produced by the various autoencoders. The x-axis refers to each of the test images and the y-axis represents the score achieved. The results were sorted from lowest value to highest value to make for easier comparison. Each line represents a SimpNet model with a different compression level. The blue line (highest) shows the results for the 2X compression, and the red line (lowest) shows the results for 15X compression. This representation provides an intuitive way to understand the performance of the models on each of the test images individually. The results follow logically that the higher compression models have a lower PSNR (more noise) and lower SSIM (less structurally similar to the original image). The range under each metric is reasonably narrow, apart from the 15X compression model that has an SSIM value as low as 0.70 with a maximum of 0.85. We note that the average SSIM for 2X compression is 0.87, and that of 15X is 0.80.

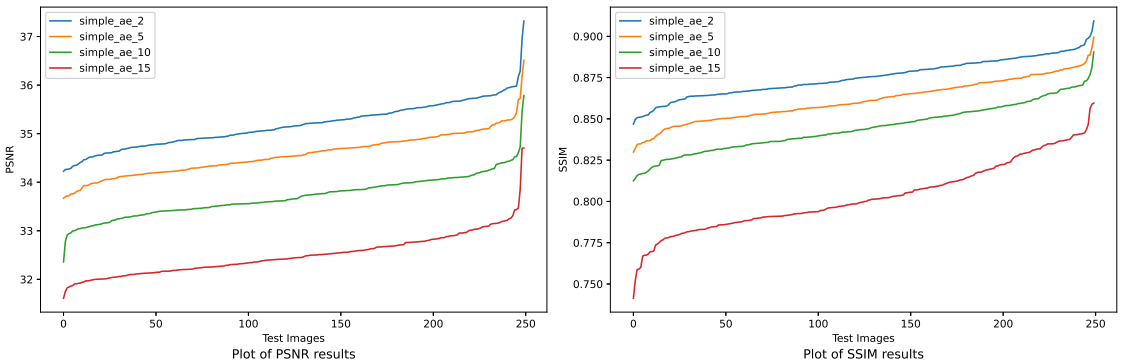

Figure 7.8: SimpNet Results for PSNR and SSIM

7.1.2 VggNet

The VggNet autoencoders were trained in the same manner as SimpNet: over 50 epochs, using ADAM optimiser, a mean-square error loss function and learning rate decay.

7.1.2.1 Training Curves

The training curves show that the training and validation loss converge quite smoothly, while there remains volatility in the accuracy curves. Despite this, we notice that the accuracy curves are clearly increasing while the loss curves are decreasing, indicating that the models are fitting well and can generalise with reasonably high accuracy.

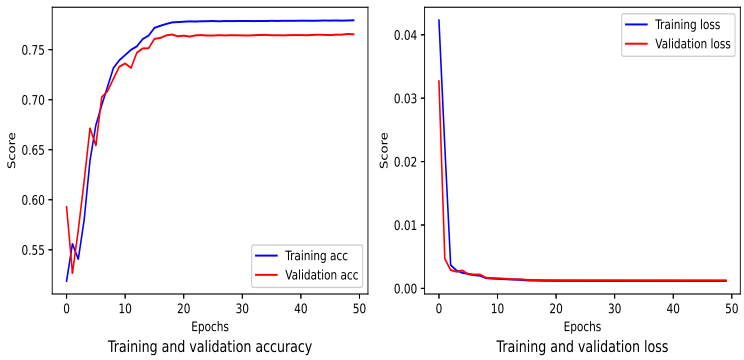

Figure 7.9: Training Curves for VggNet 2X Compression

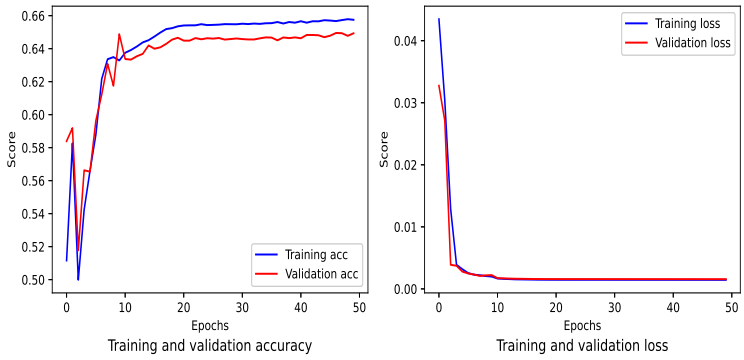

Figure 7.10: Training Curves for VggNet 5X Compression

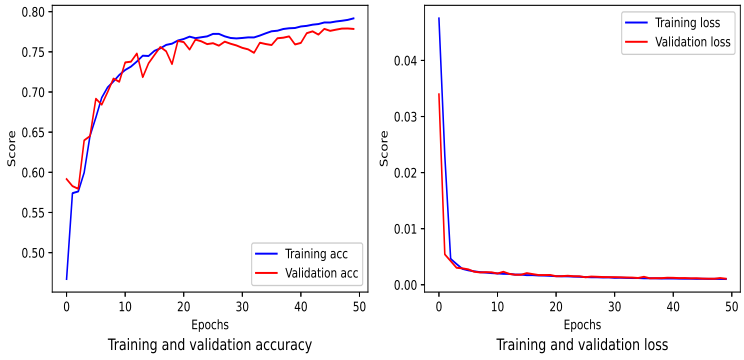

Figure 7.11: Training Curves for VggNet 10X Compression

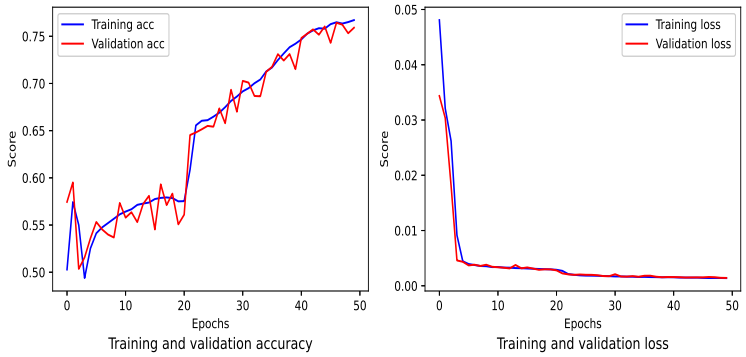

Figure 7.12: Training Curves for VggNet 15X Compression

7.1.2.2 Predicted Images

The images produced by VggNet are of slightly lower quality (by inspection) with noticeably more "blurriness" at the 15X compression level. Despite this, we can discern the overall morphological characteristics of the cells. In addition, the loss of minor stain artefacts aligns with the results of SimpNet, with the autoencoder prioritising spatial information over spectral information. This is an acceptable outcome for diagnostic use, as it's the shape and presence of leukocytes that matters more than colour.

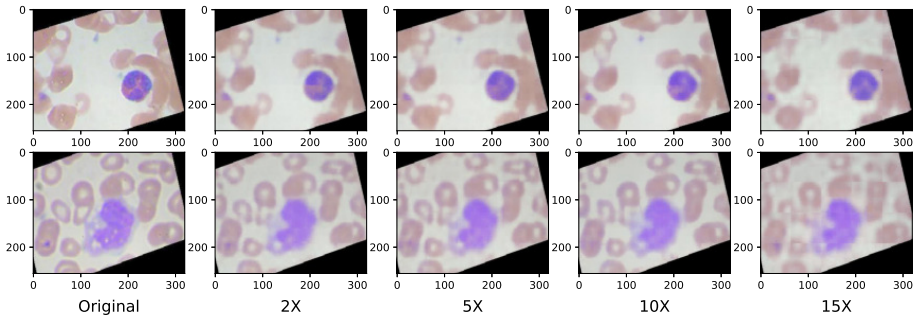

Figure 7.13: VggNet Predicted Images at Each Compression Level

7.1.2.3 Quality Metrics

The quality produced by VggNet exhibits more variance across the test data with both PSNR and SSIM having a wider range than found under SimpNet.

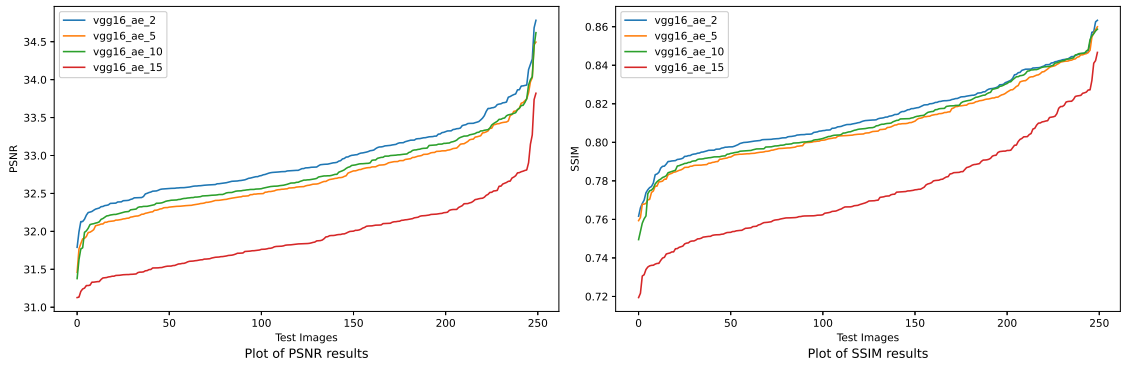

Figure 7.14: VggNet Results for PSNR and SSIM

7.1.3 JPEG

JPEG compression is applied to the original data, and its quality metrics are calculated. In implementing the JPEG algorithm, we use the Python Imaging Library (PIL) and adjust the quality setting to achieve the desired (approximate) compression level. The quality value is an integer value in the range $[0, 1]$. It is important to note that at the lowest quality setting, the maximum compression ratio obtained is 10X.

7.1.3.1 Predicted Images

The sample images presented in figure 7.15 show very noticeable degradation in image quality as we move from low to high compression. At 10X, the image quality is too poor to see the different blood cells present clearly. At 2X, however, JPEG produces an image that visually looks very similar to the original image. Of note is the trade-off between spatial and spectral information. We notice that at 10x compression, JPEG has lost significant amounts of spectral information, with less loss of spatial information. This results in the loss of colour but retention of shape information. When compared to the autoencoders, we find that JPEG makes this trade-off at lower compression levels. The autoencoders still preserve much of the spectral and spatial information at 10x and only present noticeable degradation 15x.

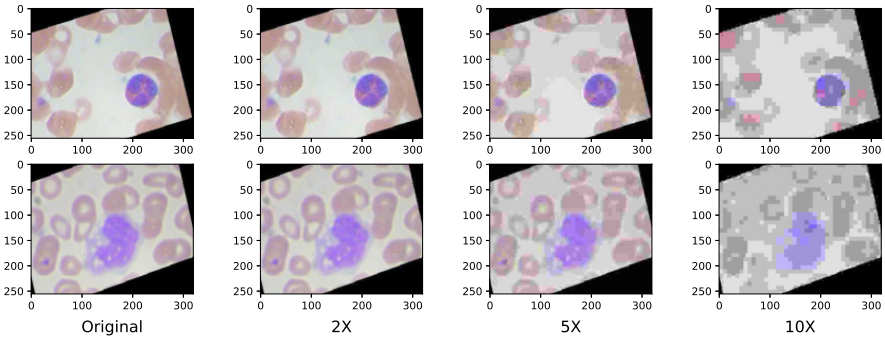

Figure 7.15: JPEG Compressed Images at Each Compression Level

7.1.3.2 Quality Metrics

The observation made above when visually inspecting the output images is reflected in the quality metrics produced. Figure 7.16 shows that at lower compression levels (2x) JPEG performs very well, with deterioration in quality as the compression level increases. At the 2x compression level, the SSIM measure produces a score above 0.90, implying that the image based on luminance, correlation and contrast is very similar to the original image. It is clear from these graphs that performance is much worse at 10X compression, where across the test set, the maximum SSIM score achieved was 0.75.

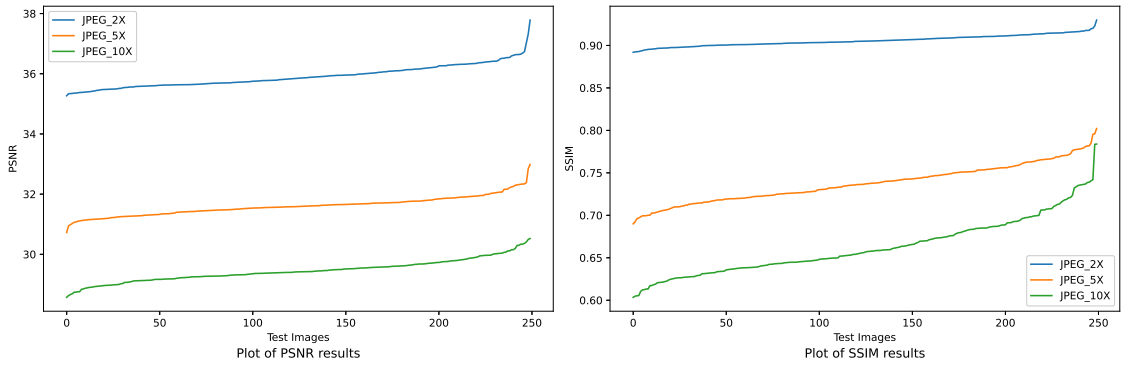

Figure 7.16: JPEG Results for PSNR and SSIM

7.1.4 Comparative Performance

To provide a more meaningful comparison, we plot the quality metrics on the same set of axes. It is clear that JPEG outperforms the autoencoders at low levels of compression; however, at 5X and 10X, its performance is markedly worse. These results imply that an autoencoder can be trained to exploit a particular type of data's specific characteristics to provide higher compression with less loss than JPEG (at higher compression levels).

7.1.4.1 SimpNet vs JPEG

At levels above 2X compression, SimpNet outperforms JPEG based on the quality metrics. It is evident from figure 7.17 that the 15X autoencoder performs better on the quality metrics than the 5X and 10X JPEG compression.

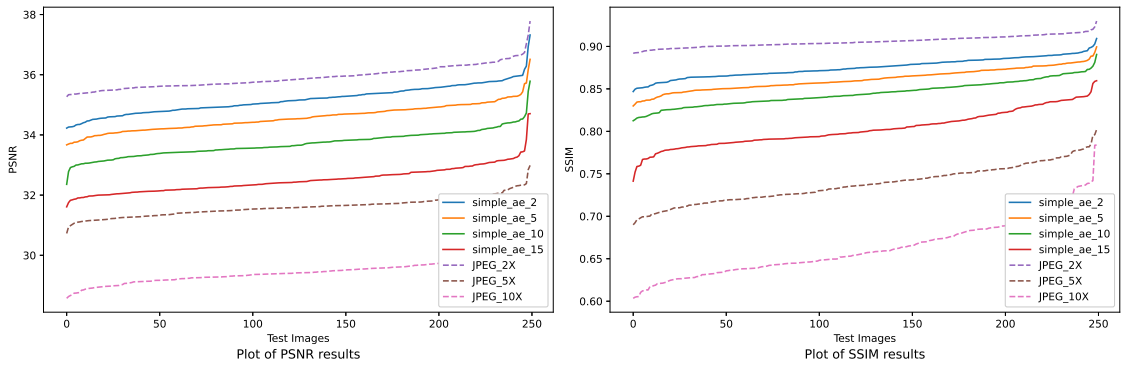

Figure 7.17: Comparison of SimpNet to JPEG

7.1.4.2 SimpNet vs VggNet

SimpNet demonstrates a clear advantage over the more complex VggNet, performing better at all compression levels.

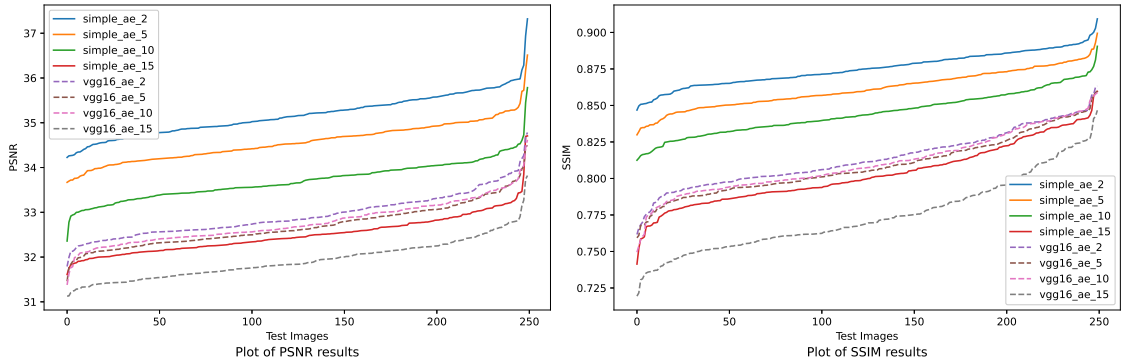

Figure 7.18: Comparison of SimpNet to VggNet

7.1.4.3 VggNet vs JPEG

A comparison of VggNet vs JPEG shows that JPEG has a very clear advantage at the 2X compression level, however performs far worse at 10X compression (with a lower result than at 15X using the autoencoder).

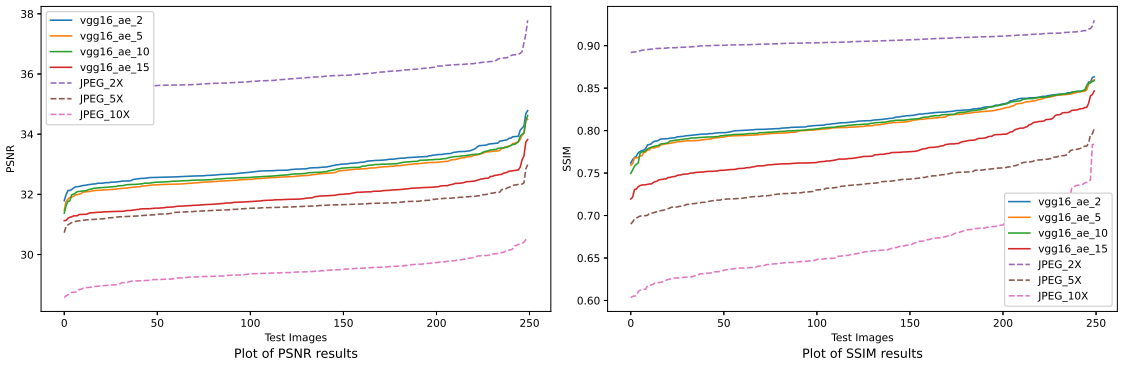

Figure 7.19: Comparison of VggNet to JPEG

7.2 Compression Experiments: VMD Data

The VMD dataset is significantly larger containing 38 686 images in the total training set. Given that the blood images have few items of interest and cell shapes are fairly uniform, using a larger dataset to learn the representation of blood showed good results that further supports our hypothesis. These are elaborated on below.

7.2.1 SimpNet

7.2.1.1 Training Curves

The training curves show a very quick progression and convergence between the training and validation curves (on both accuracy and loss). The lack of divergence between the training and validation loss indicates that the model is not overfitting which is a good outcome. In each each epoch, the network is adjusting its weights and hence learning features across a very large dataset. As such, we experience much smoother loss and accuracy curves (when compared to the Kaggle dataset). In this case, it appears that the size of the dataset adequately caters for the (limited) dimensionality present in the blood images, and hence predicts (or reconstructs) the compressed images very well.

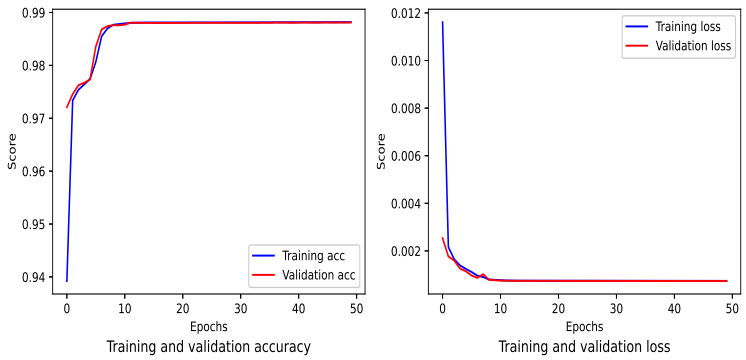

Figure 7.20: Training Curves for SimpNet 2X Compression

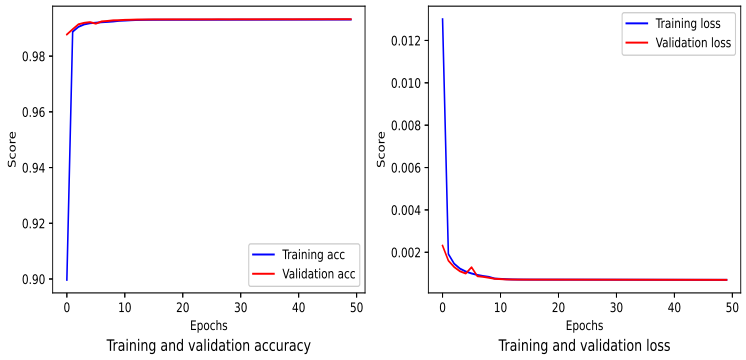

Figure 7.21: Training Curves for SimpNet 5X Compression

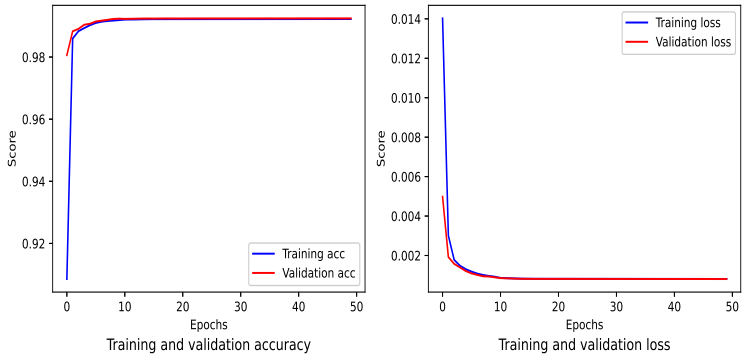

Figure 7.22: Training Curves for SimpNet 10X Compression

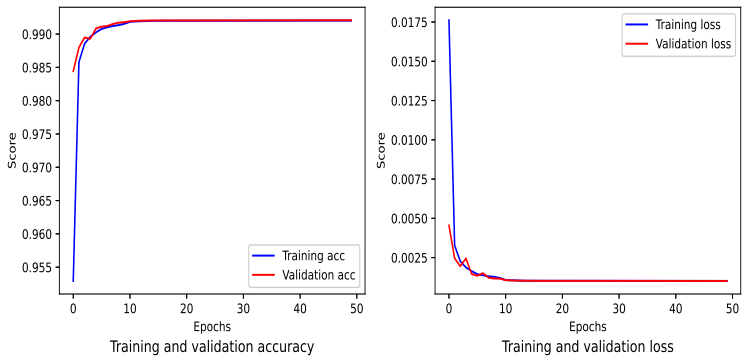

Figure 7.23: Training Curves for SimpNet 15X Compression

7.2.1.2 Predicted Images

Visual inspection of the images show that the quality of the images are largely maintained. Very slight detail, such as granules within each cell are blurred in the 15X compression, but the overall morphological characteristics are preserved. These characteristics are important to identify the type of leukocyte (white blood cell) present.

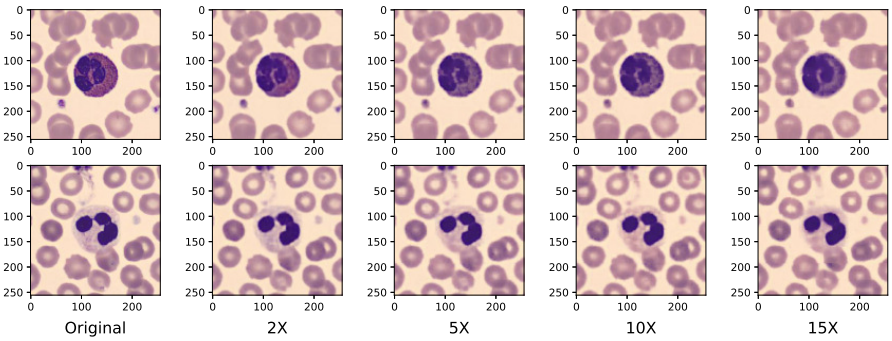

Figure 7.24: SimpNet Compressed Images at Each Compression Level

7.2.1.3 Quality Metrics

Based on the quality metrics presented in figure 7.25 we notice that the performance of SimpNet on the VMD data is good, producing average SSIM across the test set of 0.89 at the 2X level and an average of 0.85 at 15X level. This is an improvement over the results obtained using the Kaggle dataset (0.88 and 0.80 respectively). While the gains may appear marginal, the importance of near perfect replication of images is crucial to accurate diagnosis.

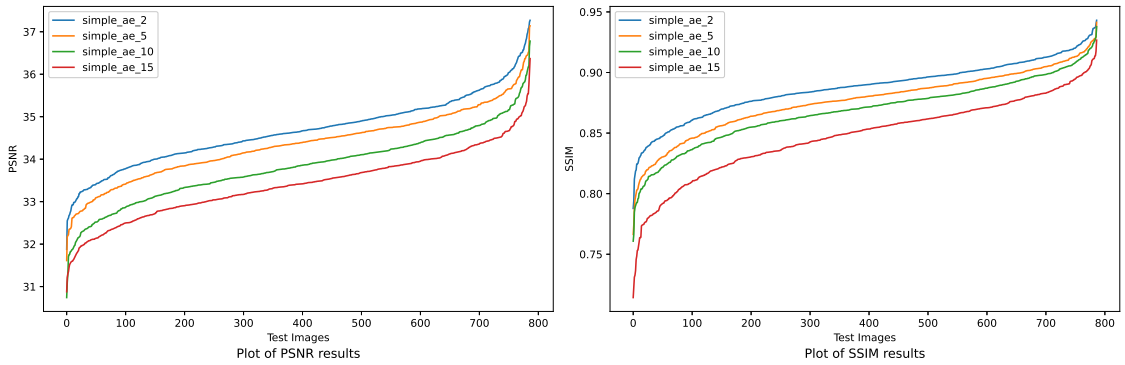

Figure 7.25: SimpNet results for PSNR and SSIM

7.2.2 VggNet

The results of compression using the VggNet show that no benefit over SimpNet is gained from the additional complexity of the network.

7.2.2.1 Training Curves

The training curves show very similar results to those achieved with SimpNet. There is no evidence of overfitting, as the training and validation curves converge and are stable. This is noticed across all models.

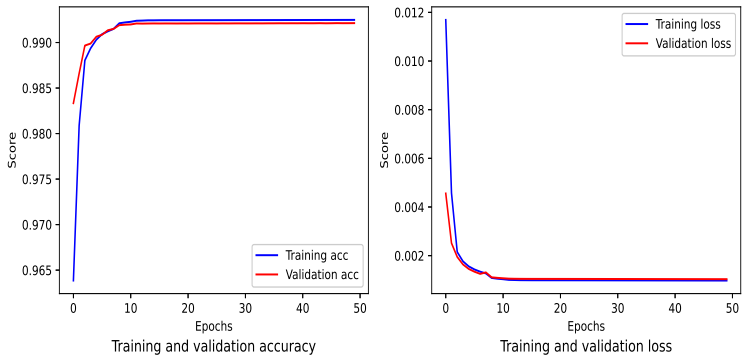

Figure 7.26: Training Curves for VggNet 2X Compression

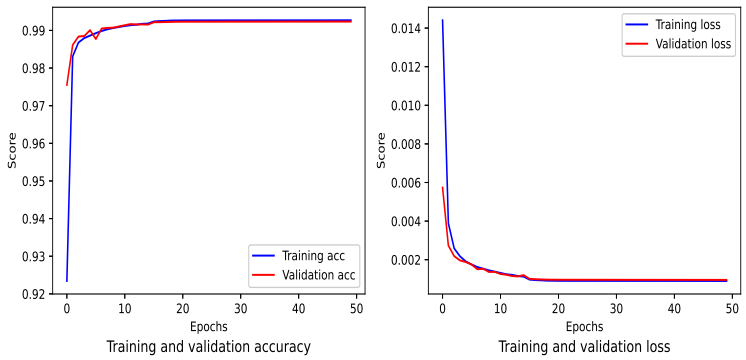

Figure 7.27: Training Curves for VggNet 5X Compression

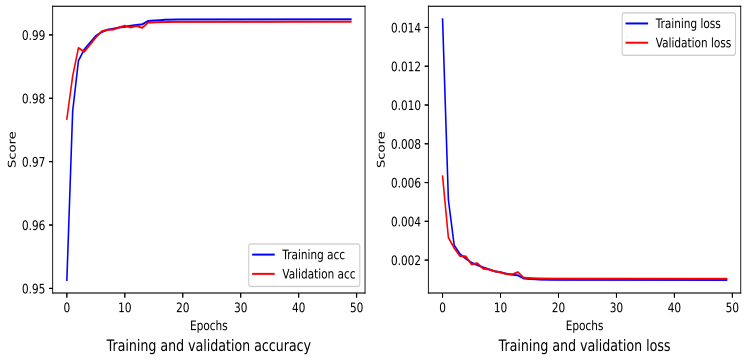

Figure 7.28: Training Curves for VggNet 10X Compression

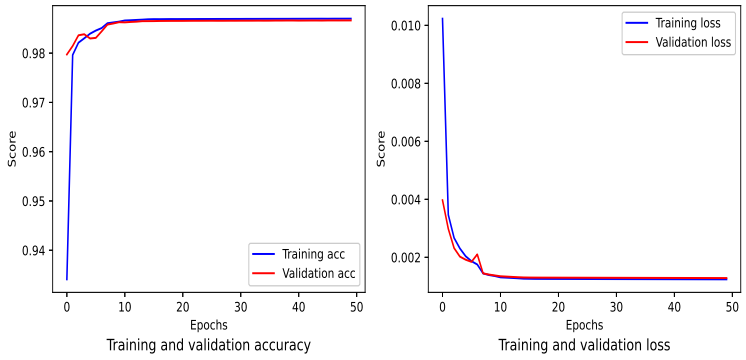

Figure 7.29: Training Curves for VggNet 15X Compression

7.2.2.2 Predicted Images

As noticed with the Kaggle experiments, the VMD images produced by VggNet are slightly lower quality (by inspection) with noticeably more “blurriness” at the 15X com-

pression level. Despite this, the overall morphological characteristics of the cells can be determined. When compared to SimpNet, we see that the outputs of the VggNet are of lower quality implying more detail loss.

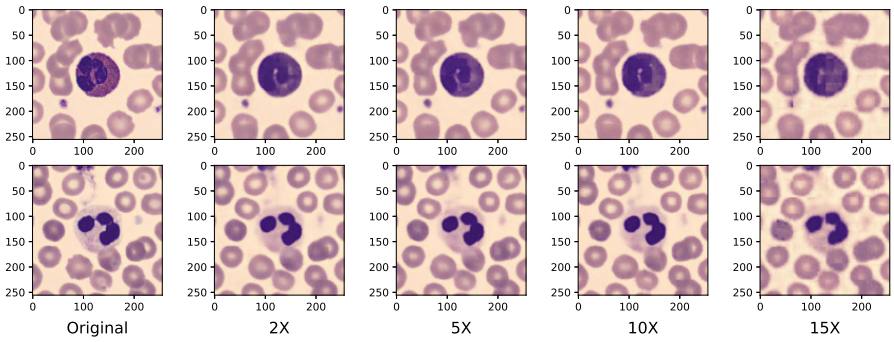

Figure 7.30: VggNet Compressed Images at Each Compression Level

7.2.2.3 Quality Metrics

We notice that the quality produced by VggNet is good, however lower than that achieved by SimpNet. The results are as expected with the lower compression levels producing higher values in both the PSNR and SSIM curves, with 15x showing the lowest values.

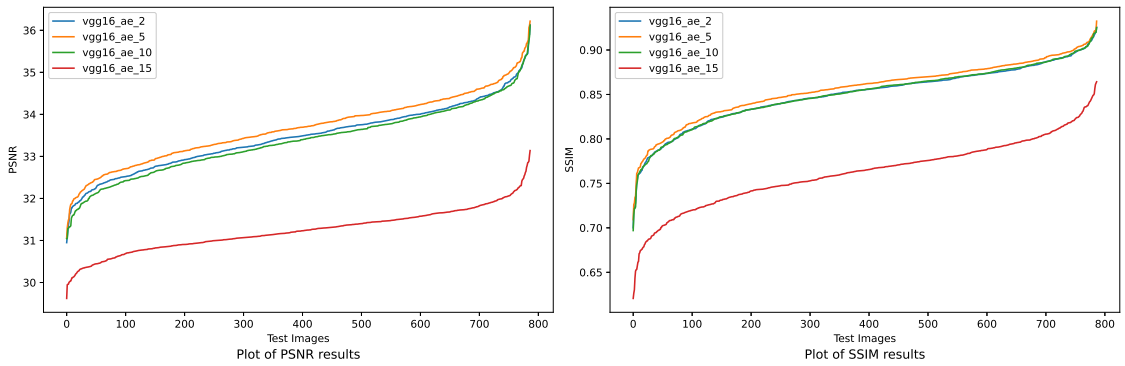

Figure 7.31: VggNet results for PSNR and SSIM

7.2.3 JPEG

7.2.3.1 Predicted Images

As noticed with the Kaggle data, by visual inspection we notice very little difference between the original image the image at 2x compression. However, as we move to higher levels of compression, the degradation becomes more noticeable with key features of

the blood images being obscured at 10x compression. The trade-off between spatial and spectral information is again apparent at the 10x level, which shows much more spectral information loss than our results from the autoencoders.

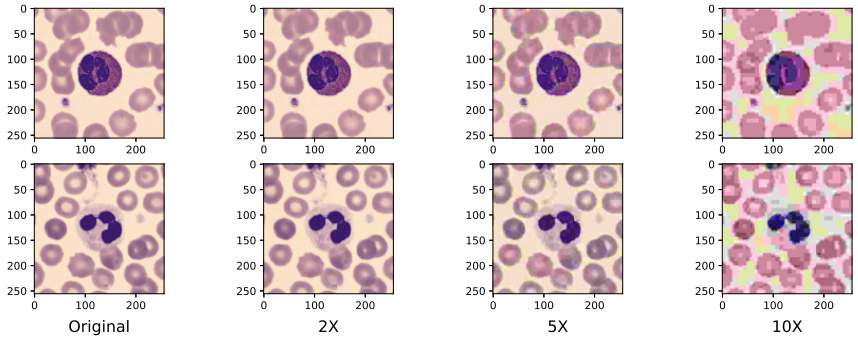

Figure 7.32: JPEG Compressed Images at Each Compression Level

7.2.3.2 Quality Metrics

Figure 7.33 shows that the JPEG quality results achieved on the VMD data are very similar to that achieved on the Kaggle data. We notice again that the SSIM values at 2x compression are much higher than those achieved at either 5x or 10x. This corroborates with the degradation in quality reflected in the sample images presented in 7.32 above.

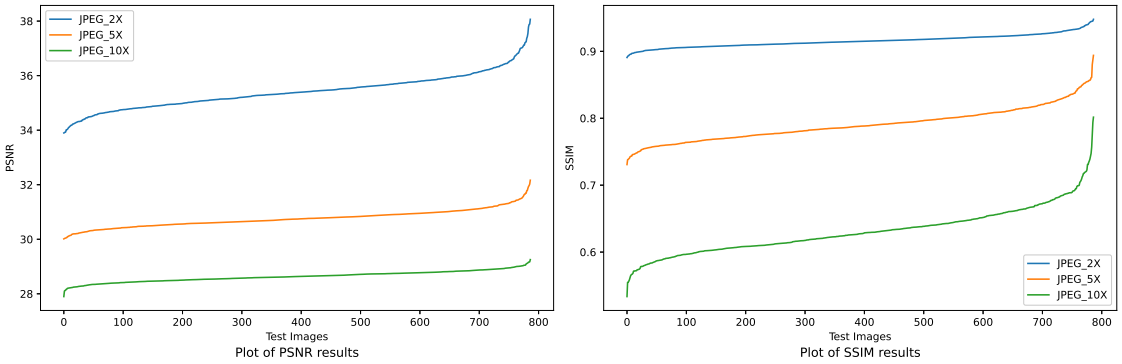

Figure 7.33: JPEG results for PSNR and SSIM

7.2.4 Comparative Performance

Given that SimpNet outperforms VggNet based on the quality curves above, we focus on comparing SimpNet to JPEG compression in figure 7.34. In this diagram we note that at low levels of compression JPEG performed better, however at all higher levels of compression, SimpNet outperforms.

7.2.4.1 SimpNet vs JPEG

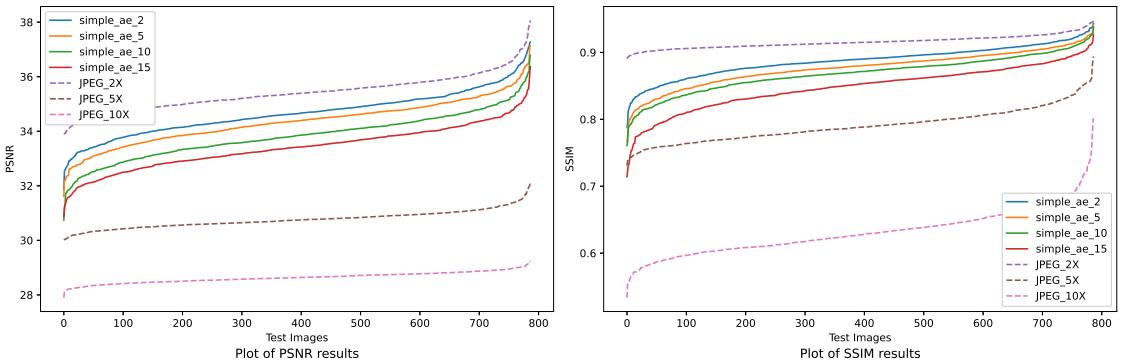

Figure 7.34: Comparison of SimpNet to JPEG

7.2.5 Summary of Compression Experiments

The tables below summarise the quality metrics calculated under each model. In each table and for each measure, the best value is highlighted in bold text. From the results, it appears that JPEG performs better than the autoencoders at the lowest compression level (2x); however, when we compare the average SSIM and PSNR values, we notice that the advantage is marginal over our best autoencoder (SimpNet).

From 5x compression and higher, the autoencoders display a clear advantage over JPEG. These results support our hypothesis that machine learning-based autoencoder techniques perform better in this specific case of blood image compression.

Table 7.1: Quality results for 2x compression using Kaggle Data

Model Name	SSIM Min	SSIM Max	SSIM AVG	PSNR Min	PSNR Max	PSNR AVG
SimpNet 2x	0.85	0.91	0.88	34.23	37.32	35.17
VggNet 2x	0.76	0.86	0.81	31.79	34.78	32.93
JPEG 2x	**0.89**	**0.93**	**0.91**	**35.27**	**37.78**	**35.91**

Table 7.2: Quality results for 5x compression using Kaggle Data

Model Name	SSIM Min	SSIM Max	SSIM AVG	PSNR Min	PSNR Max	PSNR AVG
SimpNet 5x	**0.83**	**0.90**	**0.86**	**33.67**	**36.51**	**34.57**
VggNet 5x	0.76	0.86	0.81	31.46	34.49	32.7
JPEG 5x	0.69	0.8	0.74	30.73	32.99	31.6

Table 7.3: Quality results for 10x compression using Kaggle Data

Model Name	SSIM Min	SSIM Max	SSIM AVG	PSNR Min	PSNR Max	PSNR AVG
SimpNet 10x	**0.81**	**0.89**	**0.84**	**32.36**	**35.78**	**33.71**
VggNet 10x	0.75	0.86	0.81	31.38	34.62	32.77
JPEG 10x	0.6	0.78	0.66	28.57	30.53	29.45

Table 7.4: Quality results for 15x compression using Kaggle Data

Model Name	SSIM Min	SSIM Max	SSIM AVG	PSNR Min	PSNR Max	PSNR AVG
SimpNet 15x	**0.74**	**0.86**	**0.80**	**31.61**	**34.71**	**32.5**
VggNet 15x	0.72	0.85	0.77	31.13	33.82	31.93
JPEG 15x	-	-	-	-	-	-

Table 7.5: Quality results for 2x compression using VMD Data

Model Name	SSIM Min	SSIM Max	SSIM AVG	PSNR Min	PSNR Max	PSNR AVG
SimpNet 2x	0.79	0.94	0.89	31.88	37.27	34.66
VggNet 2x	0.71	0.93	0.85	30.95	36.13	33.46
JPEG 2x	**0.89**	**0.95**	**0.92**	**33.9**	**38.07**	**35.41**

Table 7.6: Quality results for 5x compression using VMD Data

Model Name	SSIM Min	SSIM Max	SSIM AVG	PSNR Min	PSNR Max	PSNR AVG
SimpNet 5x	**0.77**	**0.94**	**0.88**	**31.61**	**37.14**	**34.34**
VggNet 5x	0.71	0.93	0.86	31.07	36.22	33.66
JPEG 5x	0.73	0.89	0.79	30.01	32.16	30.76

Table 7.7: Quality results for 10x compression using VMD Data

Model Name	SSIM Min	SSIM Max	SSIM AVG	PSNR Min	PSNR Max	PSNR AVG
SimpNet 10x	**0.76**	**0.94**	**0.87**	**30.74**	**36.78**	**33.82**
VggNet 10x	0.70	0.93	0.85	31.04	36.12	33.36
JPEG 10x	0.53	0.8	0.63	27.9	29.25	28.63

Table 7.8: Quality results for 15x compression using VMD Data

Model Name	SSIM Min	SSIM Max	SSIM AVG	PSNR Min	PSNR Max	PSNR AVG
SimpNet 15x	**0.71**	**0.93**	**0.85**	**30.88**	**36.37**	**33.4**
VggNet 15x	0.62	0.86	0.76	29.62	33.14	31.23
JPEG 15x	-	-	-	-	-	-

7.3 Classification Experiments

Given that we are investigating lossy image compression, it is crucial to understand the impact of the loss of detail on diagnosis. Our approach involves training a classification model on the original Kaggle data and then testing its classification accuracy on test data that has been compressed and reconstructed using the autoencoders and JPEG.

We then create new training and validation datasets by applying the autoencoder and JPEG compression techniques. These new datasets are then used to train classification models, and their performance is evaluated. For the reconstructed data produced by autoencoders, we use SimpNet as it resulted in better performance over VggNet at all compression levels.

7.3.1 Classification Model: Trained on Original Data

7.3.1.1 Training Curves

Figure 7.35 below shows the results of training the classification net on the original Kaggle data provided. As shown in the graphs, there is a clear generalisation gap between the training and validation curves. The model seems to exhibit an element of over-fitting. Given that the training and validation data are augmented images from an initial set of 410 images, the limited ability to generalise is reasonable. When testing the model on the test set, we achieve an accuracy of 85%, which is in line with our validation accuracy shown in the curves. The confusion matrix is shown in figure 7.36.

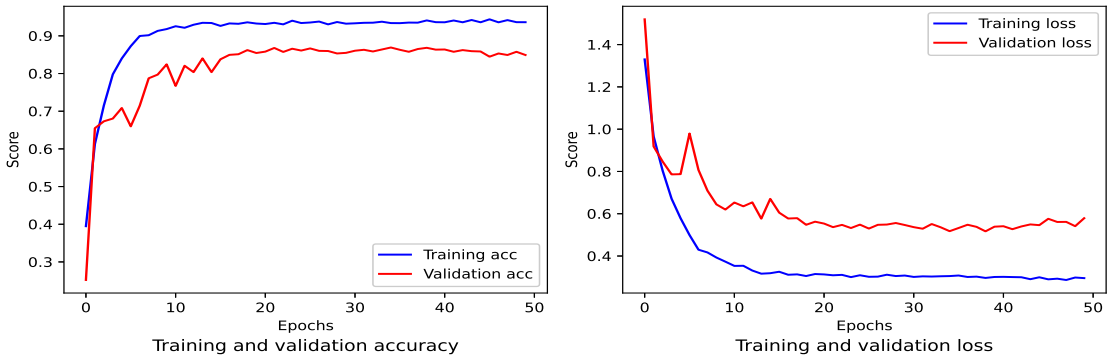

Figure 7.35: Learning Curves for the Classification Model Trained on Original Kaggle Data

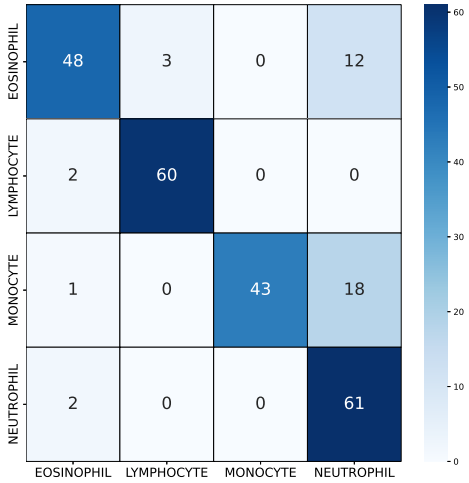

Figure 7.36: Confusion Matrix for the Classification Model Trained on Original Kaggle Data

7.3.1.2 Performance on Predicted Data

Now that we know our classification model's accuracy on the original dataset, it is informative to explore its performance on predicted data. For this, we apply the classification model to a predicted test set that has been compressed to 2X, 5X, 10X and 15X, respectively. In the case of JPEG, we stop at 10X.

Figure 7.37 shows this. The importance of this graph lies in the diminishing accuracy over the predicted data. We notice that the accuracy is at a maximum when applied to the original data. However, as we apply the classifier to predicted data produced from higher compression levels, the classification accuracy deteriorates. If we consider this in the context of providing a diagnosis, it indicates that detail lost in the compression process could negatively affect a doctor's ability to provide an accurate diagnosis. This makes intuitive sense as there are morphological differences between the blood components that must be seen clearly for classification.

Similarly, we apply the classification model to reconstructed data after applying JPEG compression. We notice that the same diminishing accuracy is present; however, the model performs better on JPEG compressed data at lower compression levels (2X compression). Beyond that point, the accuracy is higher using reconstructed data from the autoencoder. This is in line with our previous results that showed the JPEG compression outperforming autoencoders at lower levels of compression based on the quality metrics PSNR and SSIM. The graph shows that the original classifier produces a higher accuracy on autoencoder predicted data at 15X than JPEG data at 10X.

The maximum level of compression achieved with JPEG was 10x (using the lowest possible quality setting); hence the JPEG graph ends at 10x. At this level, the classifier only achieves 32% accuracy as opposed to the 61% accuracy achieved over the 10x data from the SimpNet autoencoder. This shows that the detail lost in using the autoencoder

is less than the JPEG algorithm, and hence it allows for a more accurate diagnosis of the blood images processed.

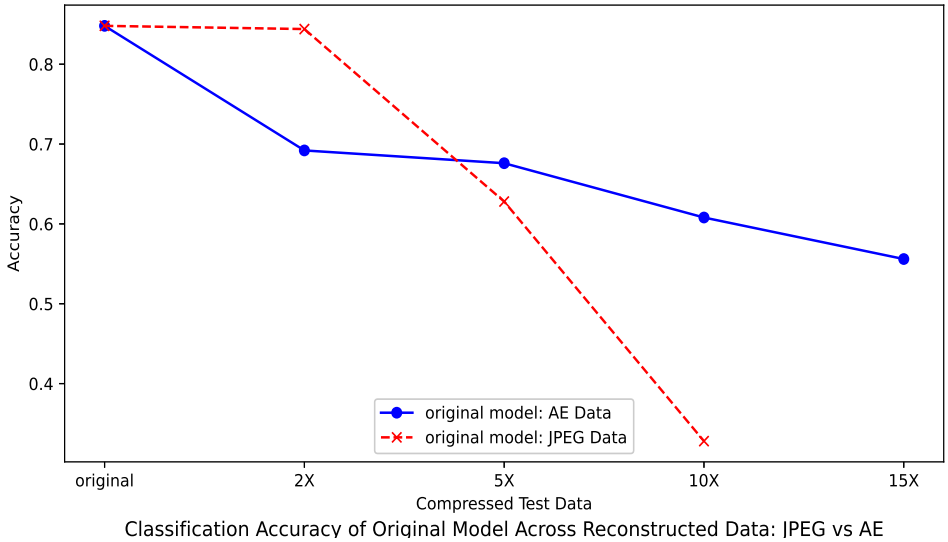

Figure 7.37: Classification accuracy over the reconstructed data

7.3.2 Classification Model: Trained on Compressed Data

As noticed, the classification accuracy diminishes as we try to classify predicted data that was subject to increasing levels of compression. Given this result, it is helpful to understand whether classification accuracy can be improved by training the classifier on reconstructed data.

To explore this, the original Training, Validation and Test set were compressed and reconstructed using the autoencoders. A classification model is created for each compression level based on data produced by the SimpNet, thus resulting in an autoencoder-classification pair wherein a classification model is optimised to deal with data produced by a particular autoencoder. The results that follow are based on data produced by SimpNet, as it performed better than VggNet in the compression experiments.

7.3.2.1 Training Curves: Data Produced by SimpNet

The training results below show that classification accuracy is recovered, albeit to a limited extent. The over-fitting problem becomes more apparent in the 15X compressed data (Figure 7.41), which is expected as there is a much more significant loss of detail than at the 2X compression level. Experimentation on a more extensive dataset could possibly yield better results. Practically these results imply that should an autoencoder be used for compression, packaging it with a classifier trained on appropriate data could help identify objects where a human may struggle due to loss of detail. This is promising for an automated diagnostic workflow. Further, it shows that the relevant

diagnostic information is still present in the compressed images, and in the absence of a classifier, some post-processing, human training or combination of both may still allow for accurate diagnostics. Which is a positive result that supports the potential of autoencoders in medical image compression.

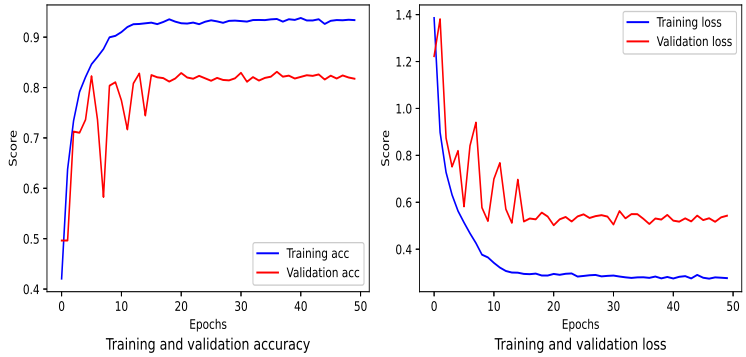

Figure 7.38: Training curves for classifier trained on SimpNet 2X data

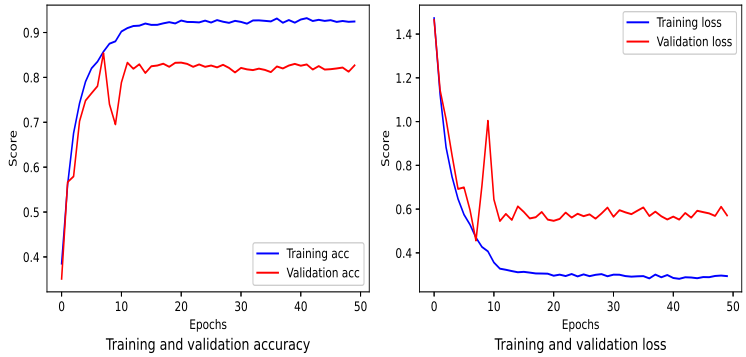

Figure 7.39: Training curves for classifier trained on SimpNet 5X data

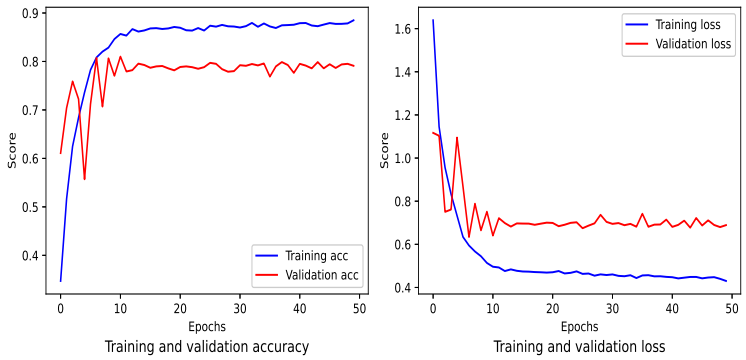

Figure 7.40: Training curves for classifier trained on SimpNet 10X data

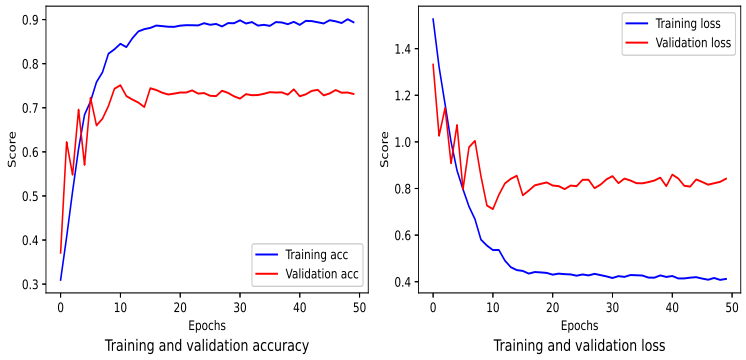

Figure 7.41: Training curves for classifier trained on SimpNet 15X data

7.3.2.2 Performance on Predicted Data

Using the optimised classification models, we test their accuracy on the test dataset. The test dataset has been compressed with each SimpNet autoencoder, producing reconstructed data from 2x, 5x, 10x, and 15x compression, respectively. Intuitively, we expect the classifiers to perform best when tested on the data that resembled the data it was trained on (i.e. a 2x classifier should perform best when used to classify data produced by a 2x compression algorithm). Figure 7.42 shows the comparative performance of each classification model over each compressed test set.

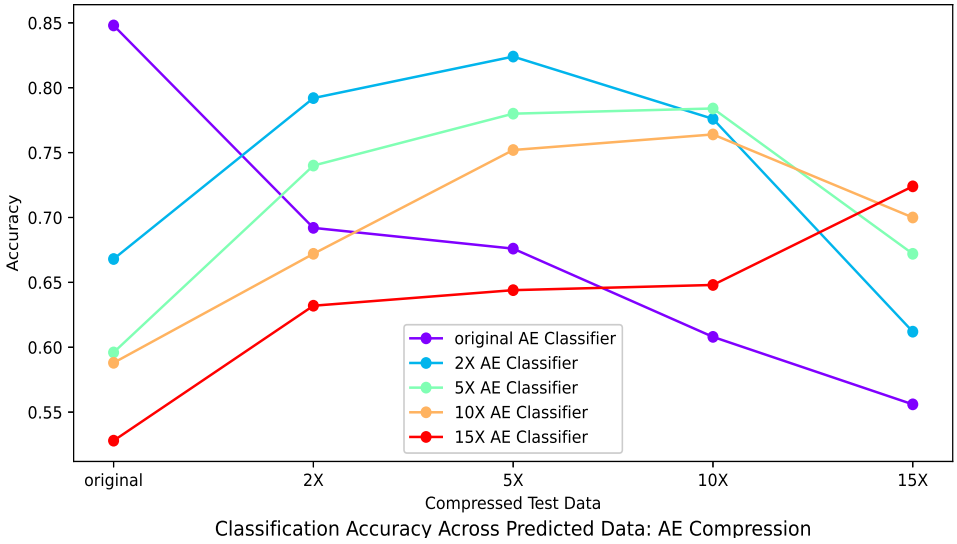

Figure 7.42: Classification accuracy over the reconstructed data under each retrained classifier

We notice in figure 7.42 above that the 2x classifier and 5x classifier performs best on compressed data at the 5x and 10x level, respectively. This is due to the classifier learning to classify the blood cell images based on their morphological characteristics. The reconstructed images from the 5x autoencoder will have more fine detail loss than the reconstructed images from the 2x autoencoder, which makes the morphological characteristics more evident in the 5x compressed image. This results in the 2x classification model showing better performance on 5x compressed data than on 2x compressed data due to information not relevant to the classification being obfuscated in the former.

We show this in figure 7.43 by taking the test set, compressing it with the autoencoders and then adding a Gaussian Blur to the images before classification. The solid lines show the accuracy of the 2x and 5x classifier on the compressed data before the blur operation, and the dashed lines show the classification accuracy after the compressed test data has been blurred. We notice that the classifiers perform better on the blurred data at the compression level that matches the classifier.

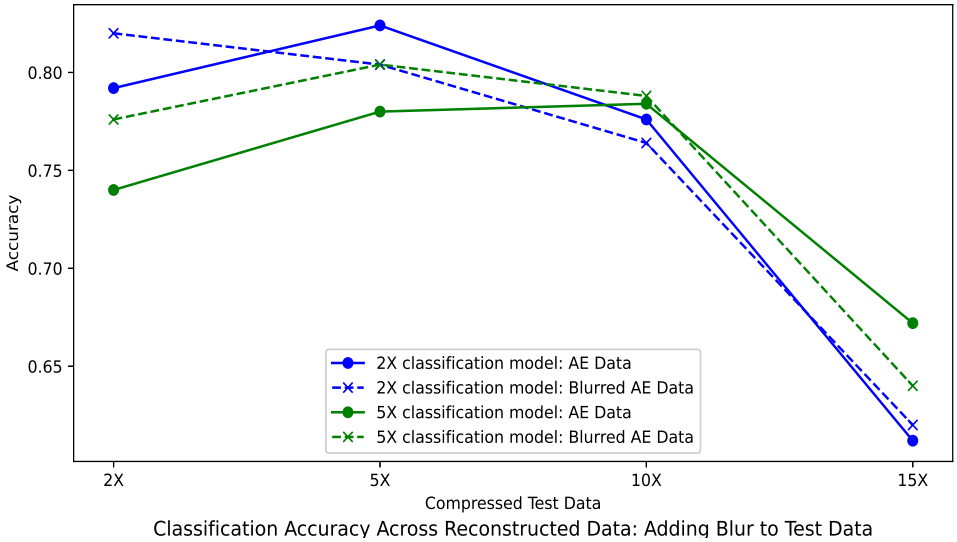

Figure 7.43: Classification accuracy over the reconstructed data with Gaussian Blur

As above, separate classification models were trained using JPEG compressed data. Figure 7.44 shows the comparative results, where the JPEG results are shown with the dashed lines. As expected, the JPEG models produce the best accuracy on the test data that matches the training data's compression level. The results show that the JPEG classification can recover at least as much accuracy as the autoencoder based classifiers.

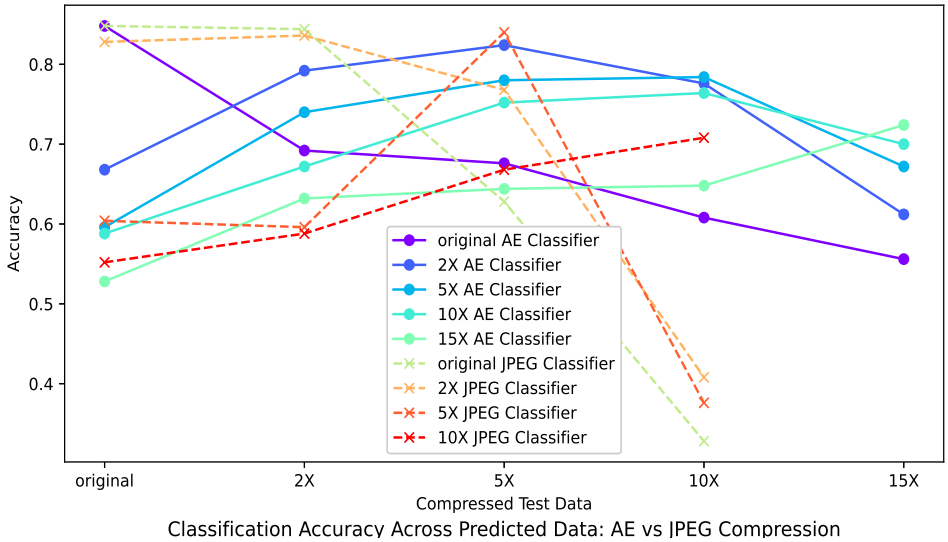

Figure 7.44: Classification accuracy over the reconstructed data: AE vs JPEG

7.3.3 Summary of Classification Experiments

From the results we conclude that compression negatively affects diagnostic accuracy, however we also conclude that some accuracy can be recovered by training the classification net on compressed data. This shows that post compression, there is still sufficient diagnostic information retained to enable diagnosis. This is a favourable outcome that supports the use of autoencoders for compression, as it implies diagnostic effectiveness is retained post compression.

Chapter 8

Conclusion

The lack of medical infrastructure in large parts of the emerging world poses a challenge to delivering medical care to these populations. To properly treat medical conditions, accurate diagnostics are required; however, this may not always be possible due to skills shortage. This challenge presents a unique opportunity for technology-based solutions to bridge the medical care need in these areas.

Various digital solutions have been brought to market, with some based on ML. These techniques presuppose digital data or at least the ability to transform analogue medical data into a digital format. There are various imaging hardware solutions in blood diagnostics ranging from low-cost to very high-end, enabling blood slide samples to be digitised and processed by ML software or sent to a doctor to review.

One issue that these solutions do not readily address is the transmission of data. The images produced by these digitisation techniques tend to be quite large and thus not easily transmitted over expensive, low bandwidth networks that are often found in parts of emerging economies.

This research paper explored the possibility of applying compression to these images to enable easier transmission and storage of this data. While various compression techniques exist, the amount of compression required to make a meaningful impact can only be achieved through lossy techniques. Given the high cost of incorrect diagnosis, existing lossy techniques are viewed with scepticism by medical professionals. We propose a possible solution to this problem by exploring the use of autoencoders to provide high compression ratios with minimal reconstruction error. Our focus in this paper has been to explore autoencoders' ability to learn the representation of blood images and exploit this to increase compression while minimising loss of detail when images are reconstructed. The results produced were promising. Our findings show that in the specific case of blood images, it is possible to design an autoencoder that can compress and reconstruct data while still scoring highly on the SSIM and PSNR quality measures. When compared to JPEG compression, we find that the autoencoders outperform JPEG at higher compression levels. We also compare two different autoencoder architectures, a simple network and one based on VGG16. Our results show that there is no real benefit to the additional complexity by having more layers.

We further explored the impact that compressed images would have on classification/diagnosis. Our findings indicate that the loss of detail at high compression does have an adverse impact on a models ability to classify the image. If we consider a

real-world parallel, the detail lost could hamper a doctor's ability to provide an accurate diagnosis. To mitigate this, we test the idea of building a classification model that trains on compressed data, and then we measure its ability to classify compressed images.

Our findings show that re-training a model on compressed data allows it to recover some accuracy, implying that despite some image quality degradation, the data relevant for diagnosis is still present. This means that we can create a diagnostic assistance tool that can support doctors in dealing with images where they may have been some detail lost due to compression.

A fundamental limitation in our findings is the data used. While the results obtained were consistent across two distinct datasets, the data used was already compressed using JPEG. The subsequent JPEG compression may have adversely affected the comparison between the autoencoder and JPEG and could form the basis for a future investigation. In addition, we note that despite the autoencoder outperforming JPEG at higher compression levels, JPEG is a general compression tool that can work reasonably well on most image types. The autoencoders in this paper require training on a particular image type generated from a particular source. This limited generality only makes it suitable for very specific tasks. We also note that blood data is limited in dimensionality, i.e. blood components are normally distributed and typically take on a regular shape. It is not certain how well this autoencoder compression approach will extend to other, more complicated medical images (such as tissue samples and x-rays). This could be investigated in further research.

We have explored the hypothesis put forward in this paper. Still, an additional research question beyond this paper's scope would be to investigate the possibility of combining an autoencoder and a classification model to create a single architecture capable of compressing, reconstructing and classifying images. The idea would be to test if a model could learn the weights in both the autoencoder and classification part of the model and optimise for the highest classification output. A model developed in this way, where the classification task is part of the learning process, would likely ensure that regions of diagnostic importance are reconstructed with higher quality and that regions with less diagnostic value are reconstructed with lower accuracy. Overall, for a given compression level, this trade-off may lead to higher overall quality in the areas of diagnostic importance and, as a consequence, higher classification accuracy. Practically, this could be packaged as a single tool that doctors could use to compress and classify images. While the more complicated architecture presented in this paper did not perform as well as the architecture with fewer layers, possible future studies could investigate the efficacy of using other architectures such as variational autoencoders [Kingma and Welling 2019] and general adversarial networks [Mentzer *et al.* 2020] to build a compression tool.

In conclusion, our findings show that subject to clinical studies, using autoencoders for compressing blood images holds promising practical applications. They have been proven to exploit the limited dimensionality of blood images to provide higher compression compared to JPEG.

Appendix A

SimpNet Architectures

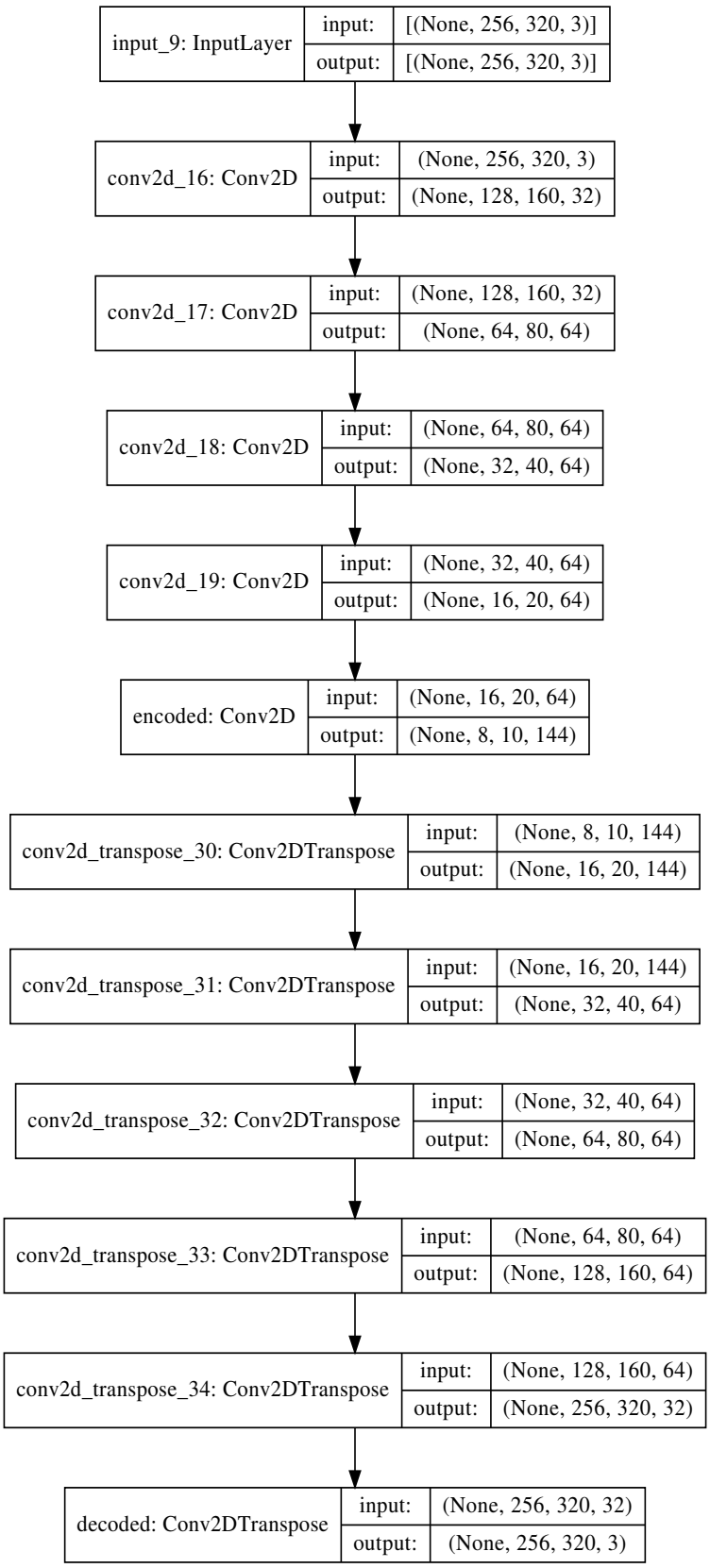

Figure A.1: Architecture for SimpNet with a compression ratio of 5x

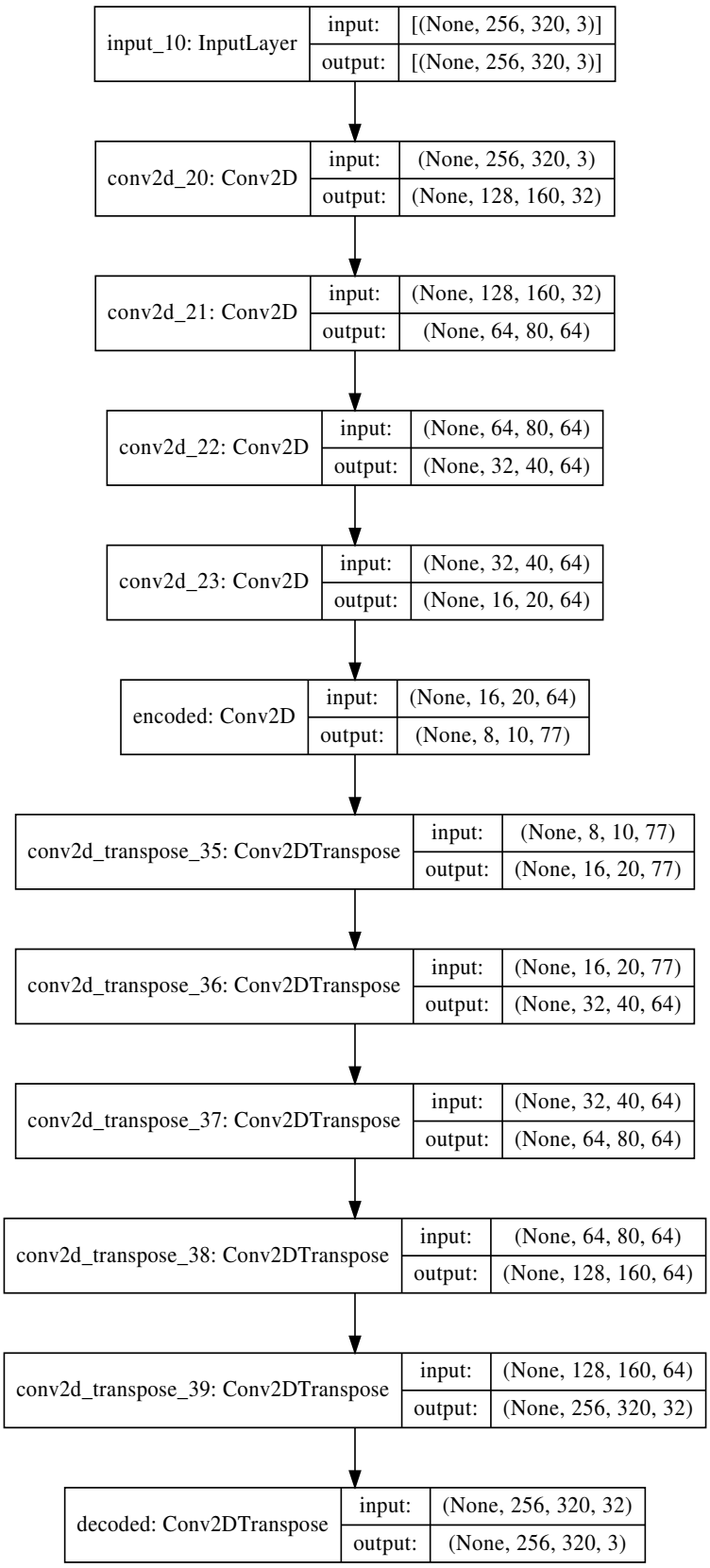

Figure A.2: Architecture for SimpNet with a compression ratio of 10x

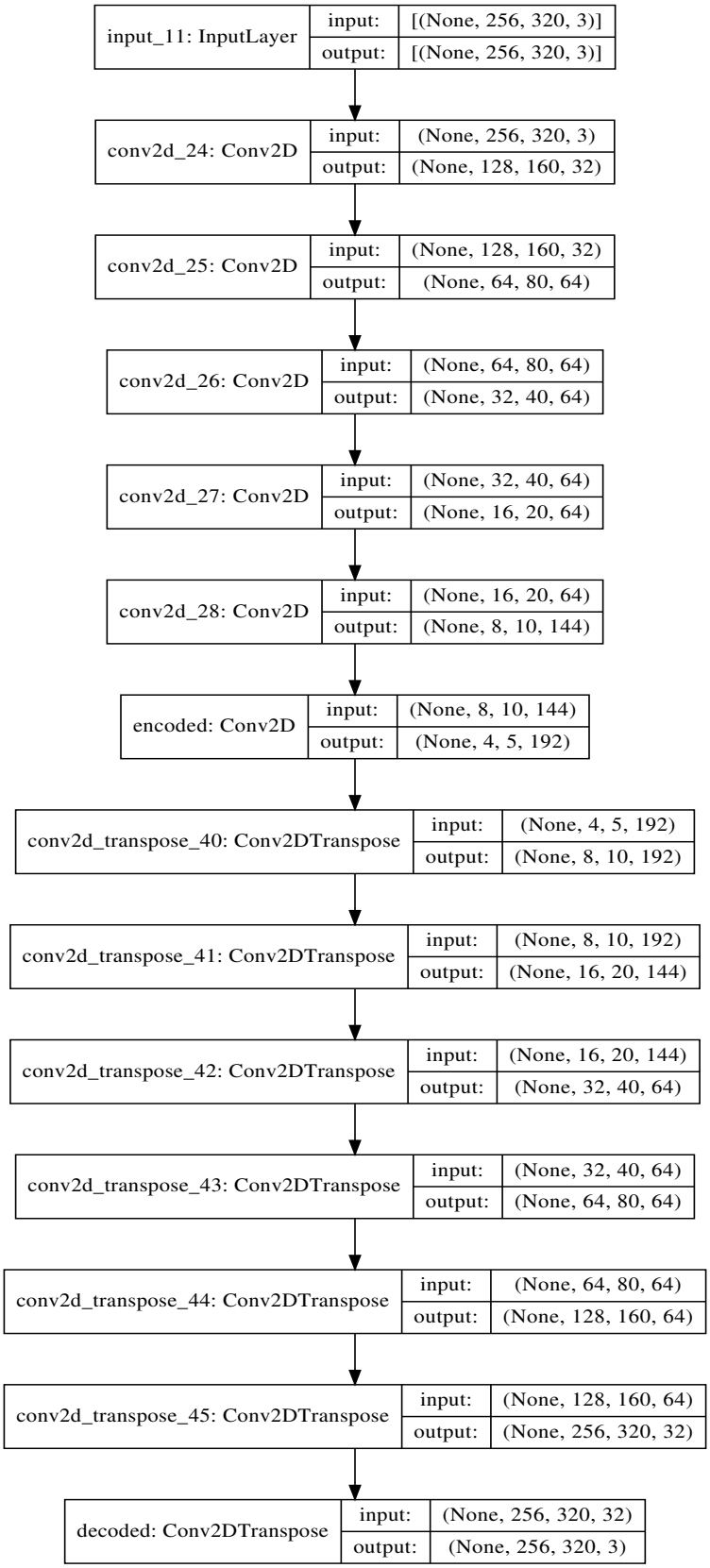

Figure A.3: Architecture for SimpNet with a compression ratio of 15x

Appendix B

VggNet Architectures

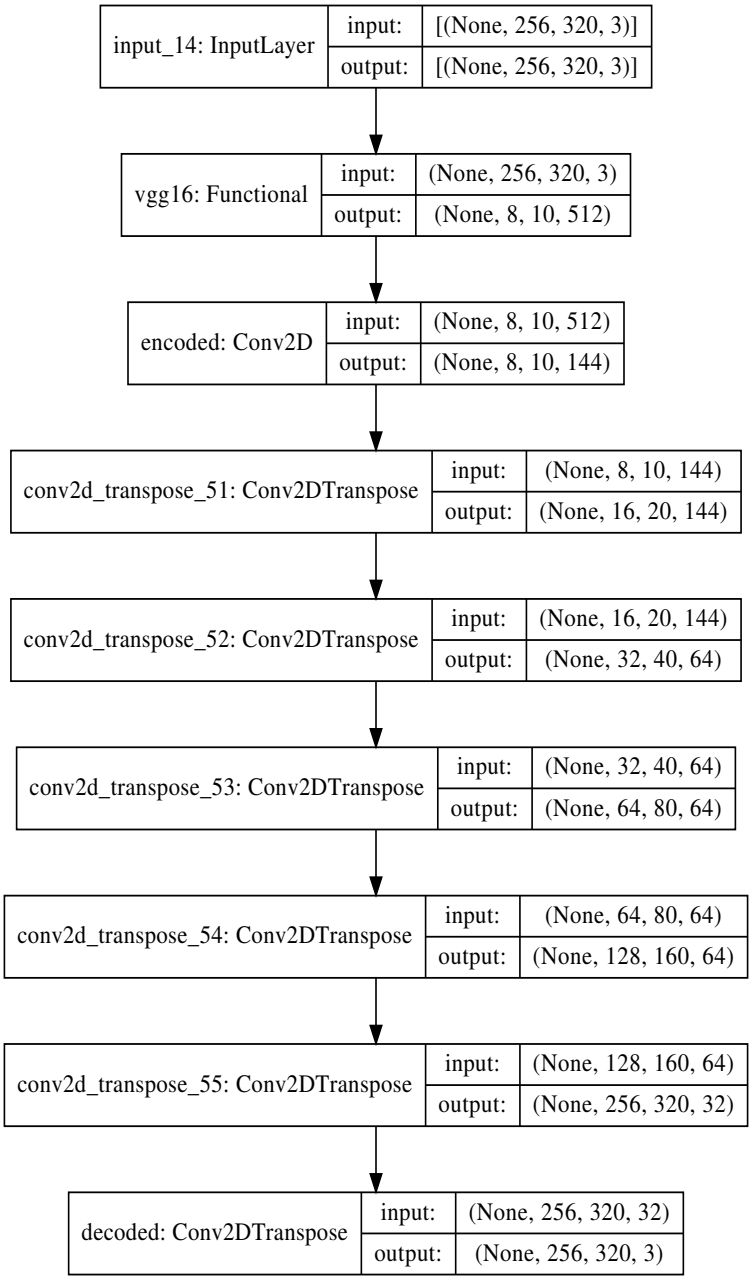

Figure B.1: Architecture for VggNet with a compression ratio of 5x

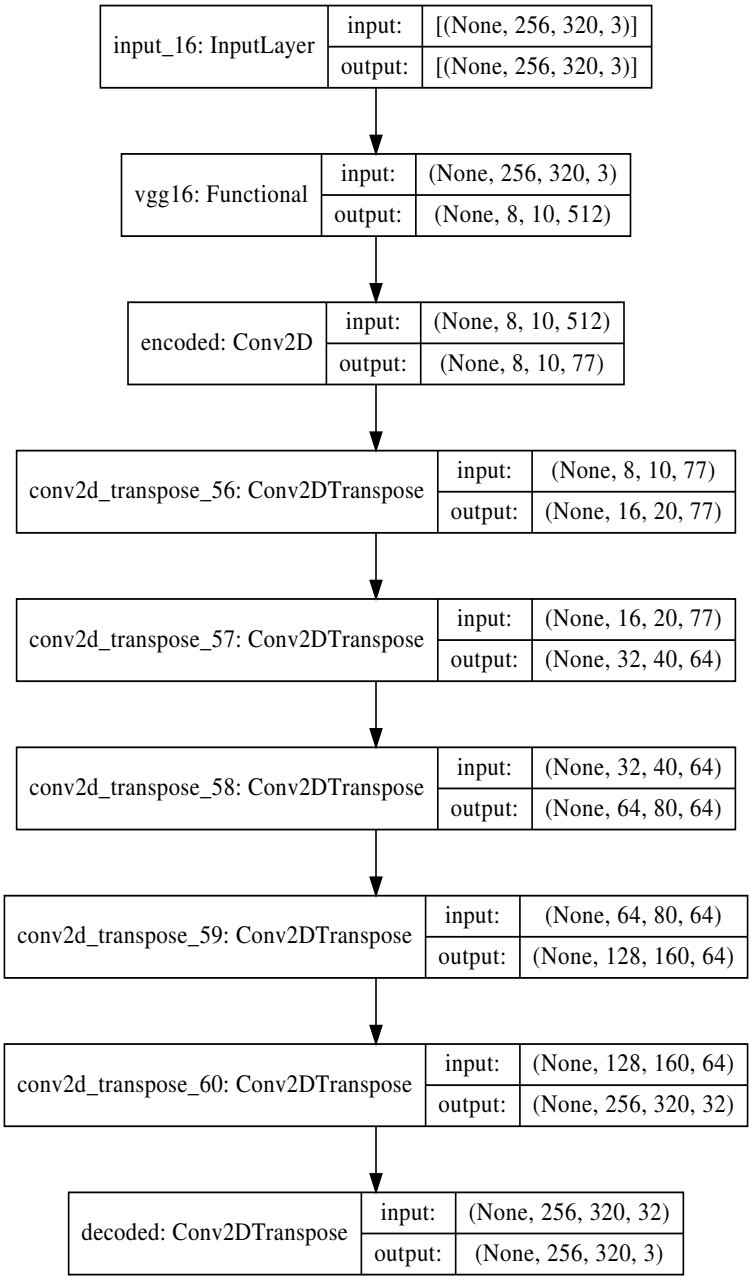

Figure B.2: Architecture for VggNet with a compression ratio of 10x

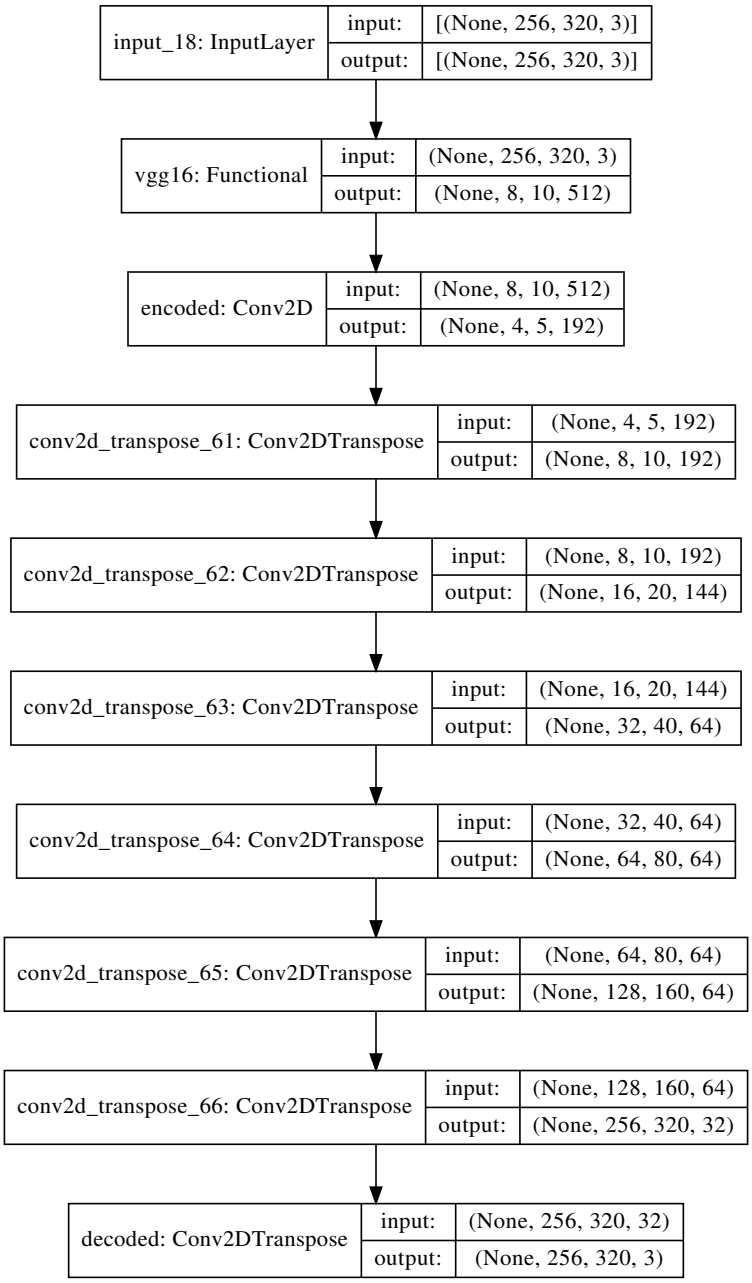

Figure B.3: Architecture for VggNet with a compression ratio of 15x

References

[Ahmed *et al.* 1974] Nasir Ahmed, T_ Natarajan, and Kamisetty R Rao. Discrete cosine transform. *IEEE transactions on Computers*, 100(1):90–93, 1974.

[Anderson and McNeill 1992] Dave Anderson and George McNeill. Artificial neural networks technology. *Kaman Sciences Corporation*, 258(6):1–83, 1992.

[Ballé *et al.* 2016] Johannes Ballé, Valero Laparra, and Eero P Simoncelli. End-to-end optimization of nonlinear transform codes for perceptual quality. In *2016 Picture Coding Symposium (PCS)*, pages 1–5. IEEE, 2016.

[Cheng *et al.* 2018] Zhengxue Cheng, Heming Sun, Masaru Takeuchi, and Jiro Katto. Deep convolutional autoencoder-based lossy image compression. In *2018 Picture Coding Symposium (PCS)*, pages 253–257. IEEE, 2018.

[Deutsch 1996] Peter Deutsch. *RFC1951: DEFLATE compressed data format specification version 1.3*, 1996.

[Dumoulin and Visin 2016] Vincent Dumoulin and Francesco Visin. A guide to convolution arithmetic for deep learning. *arXiv preprint arXiv:1603.07285*, 2016.

[Fleming 2019] Kenneth Fleming. Pathology and cancer in africa. *ecancermedicalscience*, 13, 2019.

[Hinton and Salakhutdinov 2006] Geoffrey E Hinton and Ruslan R Salakhutdinov. Reducing the dimensionality of data with neural networks. *science*, 313(5786):504–507, 2006.

[Hoffman 2012] Roy Hoffman. *Data compression in digital systems*. Springer Science & Business Media, 2012.

[Huffman 1952] David A Huffman. A method for the construction of minimum-redundancy codes. *Proceedings of the IRE*, 40(9):1098–1101, 1952.

[Kingma and Welling 2019] Diederik P Kingma and Max Welling. An introduction to variational autoencoders. *arXiv preprint arXiv:1906.02691*, 2019.

[Klein and Celik 2017] R. Klein and T. Celik. The Wits Intelligent Teaching System: Detecting student engagement during lectures using Convolutional Neural Networks. In *2017 IEEE International Conference on Image Processing (ICIP)*, pages 2856–2860, Sep. 2017.

[Langdon 1984] Glen G Langdon. An introduction to arithmetic coding. *IBM Journal of Research and Development*, 28(2):135–149, 1984.

[LeCun *et al.* 1998] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[Liu *et al.* 2019] Xiaolong Liu, Zhidong Deng, and Yuhan Yang. Recent progress in semantic image segmentation. *Artificial Intelligence Review*, 52(2):1089–1106, 2019.

[McCulloch and Pitts 1943] Warren S McCulloch and Walter Pitts. A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, 5(4):115–133, 1943.

[Mentzer *et al.* 2020] Fabian Mentzer, George Toderici, Michael Tschannen, and Eirikur Agustsson. High-fidelity generative image compression. *arXiv preprint arXiv:2006.09965*, 2020.

[Poisson 1808] Simeon Denis Poisson. Mémoire sur la propagation de la chaleur dans les corps solides. *Nouveau Bulletin des Sciences par la Société philomatique de Paris, tI*, pages 112–116, 1808.

[Raut *et al.* 2020] Yash Raut, Tasmai Tiwari, Pooja Pande, and Prachi Thakar. Image compression using convolutional autoencoder. In *ICDSMLA 2019*, pages 221–230. Springer, 2020.

[Recommendation 1992] T Recommendation. *81: Digital Compression and Coding of Continuous-Tone Still Images—Requirements and Guidelines*. Technical report, Technical report, CCITT, 1992.

[Rissanen and Langdon 1979] Jorma Rissanen and Glen G Langdon. Arithmetic coding. *IBM Journal of research and development*, 23(2):149–162, 1979.

[Sayood 2017] Khalid Sayood. *Introduction to data compression*. Morgan Kaufmann, 2017.

[Shannon 1948] Claude E Shannon. A mathematical theory of communication. *The Bell system technical journal*, 27(3):379–423, 1948.

[Shannon 1949] Claude E Shannon. Communication theory of secrecy systems. *The Bell system technical journal*, 28(4):656–715, 1949.

[Simonyan and Zisserman 2014] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[Ström and Cosman 1997] Jacob Ström and Pamela C Cosman. Medical image compression with lossless regions of interest. *Signal processing*, 59(2):155–171, 1997.

[Theis *et al.* 2017] Lucas Theis, Wenzhe Shi, Andrew Cunningham, and Ferenc Huszár. Lossy image compression with compressive autoencoders. *arXiv preprint arXiv:1703.00395*, 2017.

[Tishby *et al.* 2000] Naftali Tishby, Fernando C Pereira, and William Bialek. The information bottleneck method. *arXiv preprint physics/0004057*, 2000.

[Wallace 1992] Gregory K Wallace. The jpeg still picture compression standard. *IEEE transactions on consumer electronics*, 38(1):xviii–xxxiv, 1992.

[Wang *et al.* 2004] Zhou Wang, Alan C Bovik, Hamid R Sheikh, and Eero P Simoncelli. Image quality assessment: from error visibility to structural similarity. *IEEE transactions on image processing*, 13(4):600–612, 2004.

[Wang *et al.* 2018] Panqu Wang, Pengfei Chen, Ye Yuan, Ding Liu, Zehua Huang, Xiaodi Hou, and Garrison Cottrell. Understanding convolution for semantic segmentation. In *2018 IEEE winter conference on applications of computer vision (WACV)*, pages 1451–1460. IEEE, 2018.

[Welch 1984] Terry A. Welch. A technique for high-performance data compression. *Computer*, (6):8–19, 1984.

[World Health Organisation 2017] The World Bank World Health Organisation. Tracking universal health coverage: 2017 global monitoring report. In *Advances In Neural Information Processing Systems*, pages 2–9, 2017.

[Ziv and Lempel 1977] Jacob Ziv and Abraham Lempel. A universal algorithm for sequential data compression. *IEEE Transactions on information theory*, 23(3):337–343, 1977.

[Ziv and Lempel 1978] Jacob Ziv and Abraham Lempel. Compression of individual sequences via variable-rate coding. *IEEE transactions on Information Theory*, 24(5):530–536, 1978.

[Zuo *et al.* 2015] Zhiyong Zuo, Xia Lan, Lihua Deng, Shoukui Yao, and Xiaoping Wang. An improved medical image compression technique with lossless region of interest. *Optik*, 126(21):2825–2831, 2015.