Transformer Centric Image Compression

Ryan Slater, URN: 6579465

May, 2023

Bachelor of Science in Computer Science

Supervisor: Dr. Sotiris Moschoyiannis

1 Declaration of Originality

I declare this report as my own work, anything taken from third parties is referenced with the correct credentials.

- Ryan Slater, May 2023

2 Acknowledgements

Thank you to Dr. Sotiris Moschoyiannis for giving me guidance, providing me with areas of research to look into, help with affirming the ethical basis of my research and being a reliable point of contact for any queries during this project. A further thank you for all friends and family for their time ensuring the quality of my written dissertation.

3 Abstract

Data from images and videos is increasing in both volume and velocity with the accessibility of cameras and the modern internet. This research looks at using different vision based transformers, a relatively new neural network architecture that has proven to have strong abilities with similar objectives such as image segmentation, in an autoencoder design to build a neural network that applies compression to images. Neural architecture search is used to find an optimal architecture design for three different strengths of image compression, targetting 2, 1 and 0.5 bits per pixel. Model performance is compared against JPEG, where the benefits of a machine learning based approach show significant improvements in stronger compression ratios with rate distortion performance up to 2.8 dB better. Comparison against another neural network for image compression, which uses a variational autoencoder architecture with a hyper-prior, shows that variational autoencoders may have better compression abilities. This research finds that a deeper network of this style generally produces better image reconstruction despite producing a smaller dimensionality of latent variables, furthermore machine learning compression creates more natural colours and shapes in the reconstructed image when compared with a JPEG equivalent.

Contents

4 Introduction

4.1 Motivation

Traditional compression algorithms can form strong lossy compression on image data by utilising hand crafted feature extraction and reduction, neural networks pose a possible advantage for stronger lossy compression and better image quality after decompression due to their ability to extract subtle image features by training on a large dataset of examples. There are already examples of neural networks out performing traditional algorithms in areas of pixel specific analysis such as image segmentation [1]. Modern devices such as mobile phones and high resolution cameras create a large volume and velocity of image or frame data, for example roughly 500 hours of content is uploaded to YouTube per minute [2], this creates the motivation to create stronger compression techniques to store high amounts of data efficiently.

4.2 Project Overview

In this project I propose a transformer based neural network to compress and decompress images through an auto-encoder architecture, an architecture that reduces an input image to latent features through an encoder then reconstructs the original image using a decoder. The Transformer has been chosen as

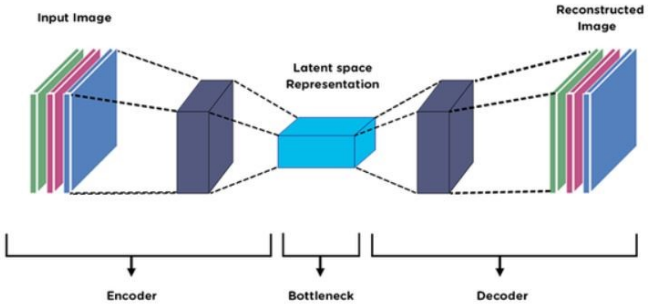


Figure 1: Diagram of an auto-encoder architecture [3]

the main building block of the network as it has proved to be very competitive in similar fields (image classification, segmentation) against convolutional neural networks. The encoder will use Shifted-Window Transformer (Swin) blocks to apply local attention and reduce the height and width of the image by using patch merging. The decoder will apply global attention using a Vision Transformer (ViT) block before using Swin blocks to apply local attention, the dimensions of the image will be restored by changing the patch merging layer of the Swin block to a patch splitting layer. While there are other compression neural networks also use transformers, they tend to use windowed (local) attention to avoid memory complexity issues of global attention. My network will differ by applying local attention until the height and width of the image are small enough to apply global attention with the hope that more useful features can be found by extracting both short-range and long-range spatial features. The features returned by the transformer encoder will be further compressed by using quantisation, a technique that transforms data into a smaller discrete domain which can reduce the amount of bits required to represent the data [4], and arithmetic coding, a lossless compression technique that compresses data by creating a codeword based on the probability of each symbol occurring [5].

Multiple different versions of the network will be made where the embedding dimension, transfer dimension, window size and depth will be varied with the intention of comparing the complexity of the networks against their compression and decompression performance. The best performing network will be compared to current leading compression networks and a traditional compression technique.

4.3 Project Aims and Objectives

1. Build a neural network that compresses images of size 224x224 as this is the same resolution used by the network that I will compare against
    - Program a model infrastructure that allows different aspects of the model to be easily changed, resulting in different levels of targetted compression.
    - Conduct a neural architecture search and track the performance of different networks.
    - Evaluate the networks to select the best performing hyper-parameters, performance will be chosen by image reconstruction quality (PSNR).
2. Have a final set of networks that target different compression ratios (2, 1 and 0.5 bpp)
    - Implement arithmetic encoding to further compress the output of the encoder.
    - Evaluate the final models with and without quantised bottleneck activations.
    - Make code to output image before and after compression along with performance metric (PSNR value).
3. Evaluate the final network against a traditional compression scheme and a neural network based scheme.
    - Conduct a qualitative and quantitative comparison of similar JPEG compression ratios to my final models.
    - At least one of my compression networks should beat the JPEG compression equivalent in PSNR performance.
    - Evaluate PSNR performance against another compression network.
    - Evaluate the training and performance of each target model to see change in image quality.

4.4 Constraints of Project

As I will be building a set of large neural networks that will be training on high amounts of data, computational resources and time to execute will be my largest issues with this project. I plan to use University of Surrey's AI compute servers to gain access to multiple high spec GPUs, this should help with computational resources as these GPUs have fast processors and high amounts of VRAM. The issue of training time cannot be easily solved and must therefore be worked around by constraining the size of training dataset and level of exploration I am able to do with adjusting my hyper-parameters, for this reason I will only be focusing on hyper-parameters that change the architecture of the network (such as depth of the network) rather than hyper-parameters they may effect optimisation (such as learning rate). To further cut down on how many training sessions I will need to run, I will not be trying every combination of my hyper-parameters but will make educated guesses as to which set of parameters would be most useful to my research, explanation of my choices start at section 7.4.1.

5 Literature Review

5.1 A Traditional Method of Compression (JPEG)

JPEG is a very general compression scheme with many options for any colour space. In this explanation I will be looking at compressing an RGB image with discrete cosine transform.

5.1.1 Colour Space Subsampling

The first step to compressing an RGB image is to change the way that colour is represented. Instead of representing a pixel as a byte of red, green and blue it is represented as a byte of luminance, red difference and blue difference, also known as YCbCr. While YCbCr takes up the same amount of data as RGB, it has been found that the human eye is much more sensitive to the intensity of light than the colour, YCbCr enables compression algorithms to exploit this by pairing and subsampling the CbCr channels so fewer bits are needed to represent a pixel, subsampled images generally follow the pattern of either 2, 4 or 8 luminance bytes for every 1 chrominance byte [6].

5.1.2 Discrete Cosine Transform

Discrete cosine transform (DCT) represents an image as being made from a set of waveforms with each waveform providing a different weighting of the information needed to rebuild the image. DCT is applied to patches of an image rather than the entire image as local patches tend to have similar features. The DCT calculates the coefficients for a set of different cosine frequencies, the coefficients can be seen as the weighting of how relevant the image channel is to that waveform. Higher waveform coefficients are more important to build the original image whereas lower waveform coefficients are less important [7]. The coefficients can be calculated with:

$$C(u,v) = a(u) \cdot a(v) \cdot \sum_{x=0}^{W-1} \sum_{y=0}^{H-1} f(x,y) \cdot cos[\frac{\pi(2x+1)u}{2W}] \cdot cos[\frac{\pi(2y+1)v}{2H}]$$

for all values u,v such that

$$u = 0, 1, 2, ..., W-1$$
$$v = 0, 1, 2, ..., H-1$$

where f(x,y) gives the data at that location of the image, W and H is the width and height of the image. a(u) and a(v) are both defined as

$$a(u) = \begin{cases} \sqrt{\frac{1}{N}}, & \text{if } u = 0 \\ \sqrt{\frac{2}{N}}, & \text{else } u \neq 0 \end{cases}$$

where N is either the height H for v or width W for u [8].

As neighbouring pixels of an image are generally similar, low frequency waveforms will be significantly more weighted than high frequency waveforms, this can be seen in figure 2. This creates a coefficient grid where values that are further away from the top left corner are less important to build the original image.

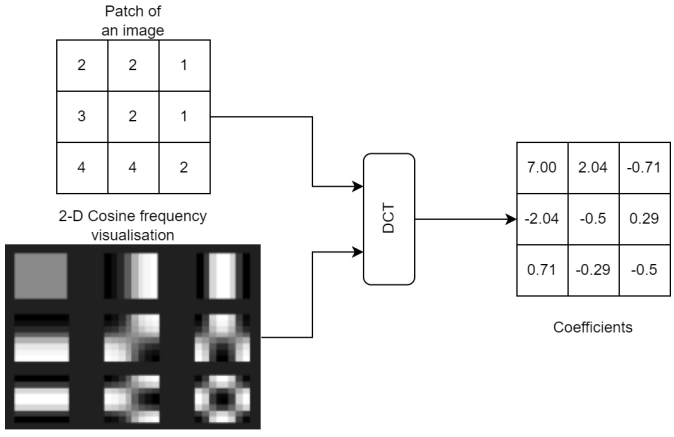


Figure 2: Discrete cosine transform on a grid of 3x3, cosine wave visualisations made by [8]

5.1.3 Quantisation

Once an image has been transformed into its coefficients of different waveforms, quantisation is applied such that important waveforms are kept while unimportant waveforms are reduced to zero, as they will have very little impact on the image quality. This is done by dividing the coefficients by a quantisation table and rounding the result to the nearest integer. The division will crush low coefficients to near-zero values and the rounding will reduce them to zero. The quantisation results in a coefficient table of mostly zeros. Different quantisation tables will result in different levels of compression and qualities of the decompressed image. As the grid of coefficients generally decreases diagonally from the top left, the final quantised values are flattened diagonally using the method shown in figure 4, this will produce a sequence which has large subsequences of repeating zeros. The flattened quantised values are encoded using Huffman encoding as the final step of compression as it will greatly benefit from the high frequency of repeating zeros in the flattened coefficient table [7].

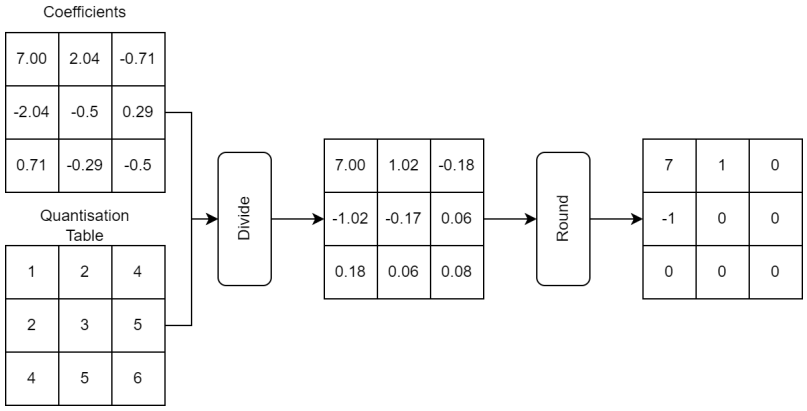


Figure 3: Quantisation of waveform coefficients

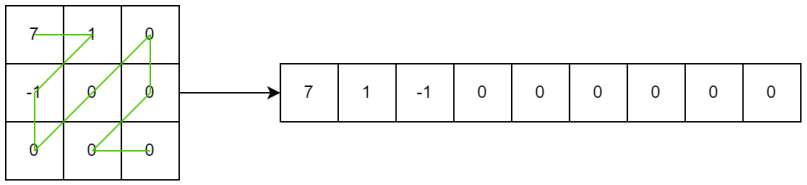


Figure 4: Flattening a coefficient grid

When the image needs to be decompressed the quantised coefficients will be reshaped into a grid and multiplied by the quantisation table to restore an approximation of the original coefficients (apart from the zeroed values). It is very important that the same quantisation table is used to quantise and de-quantise the coefficients as two different tables will provide two completely different results when passed through the inverse DCT.

5.1.4 Inverse Discrete Cosine Transform

The inverse of DCT can then be used to restore the original image by using the coefficients, the inverse of DCT is written as [8]:

$$f(x,y) = \sum_{u=0}^{W-1} \sum_{v=0}^{H-1} a(u) \cdot a(v) \cdot C(u,v) \cdot cos[\frac{\pi(2x+1)u}{2W}] \cdot cos[\frac{\pi(2y+1)v}{2H}]$$

Due to the smaller coefficients being crushed to zero, the output of this function will not be exactly the original image but will have satisfactory enough quality to look good to the human eye. While JPEG is

very successful at high compression ratios while maintaining image quality, it is strongly dependant on the assumption that neighbouring pixels are similar to each other, this means JPEG can quickly fall in image quality when posed with scenarios counter to this, such as images of text (where the pixel values will change sharply and frequently) [7].

5.2 Neural Network Models for Compression

5.2.1 Convolution based Approaches

CAS-CNN [9] is an artefact suppression network which takes a decompressed image and improves the quality of the image. CAS-CNN is a convolutional network with a total of 12 convolutional layers between the input and output layer. CAS-CNN addresses the issue of converging deep neural networks by adding multiple interim output layers which all have a loss function applied to them, this reduces the minimum input to output neural pathway from 12 layers to 9 layers. To further strengthen the backpropagation of the training gradient, skip layers are used throughout which concatenates low-level features, high-level features and output features together. The network uses channels of 128 when the tensor size is above (H/4,W/4) and 256 when the tensor size is below (H/4, W/4). Each convolutional layer is followed by a downsampling layer and there are a total of 3 stages resulting in a total reduction factor of 8 for the height and width of the image. While this is not a network used to compress images it is useful as an example to produce higher quality images after decompression.

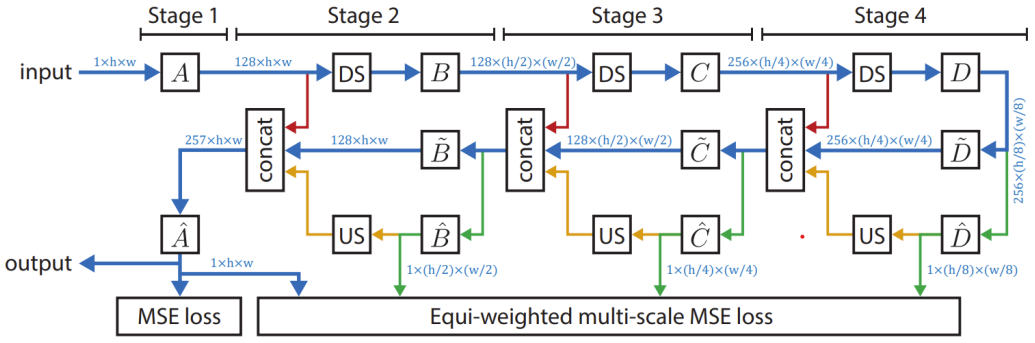


Fig. 1: Structure of the proposed ConvNet. The paths are color coded: main path (bold), concatenation of lower-level features, multi-scale output paths, re-use of multi-scale outputs.

Figure 5: Network Architecture of CAS-CNN [9]

Coarse-to-Fine Hyper-Prior Modeling for Learned Image Compression [10] builds a convolutional compression network to progressively extract latent features of different representations. The network is made up of 3 encoder and decoder layers, 2 probability estimation networks and an information aggregation network.

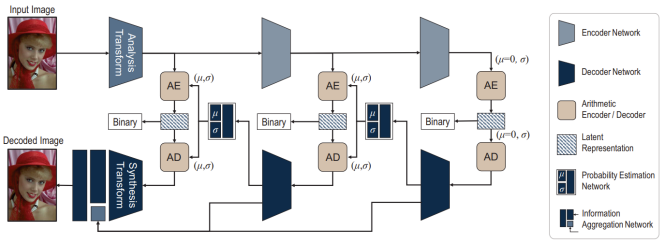


Figure 6: Full Network [10]

The encoder block takes input features, which in the case of the first encoder block would be the input image, and outputs a tensor with half the height and width. The kernel sizes of the convolutions in the

encoder are quite small as to retain as much important information about the image as possible. The space to depth module permutes the height and width of a grid of features into channelwise depth.

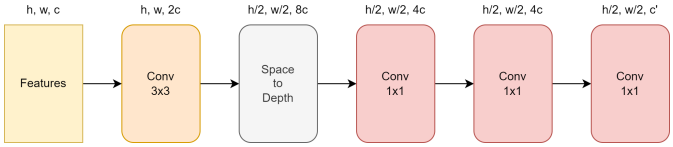


Figure 7: Internal structure of the encoder

The Decoder block takes dequantised latent features from the arithmetic decoder and outputs a tensor of double the height and width. The structure is kept very similar to that of the encoder and it uses a depth to space module, much like a space to depth but the inverse.

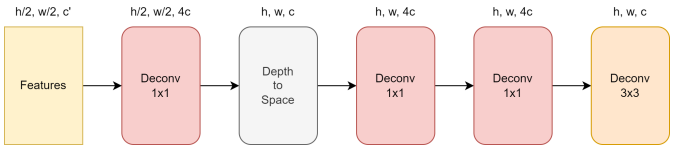


Figure 8: Internal structure of the decoder

The research paper's implementation of quantisation and arithmetic coding assumes the latent features follow a Gaussian (normal) distribution, $N(\mu, \sigma)$, where μ and σ are predicted using features from lower level representations and μ is assumed to be 0 for the lowest level of latent features.

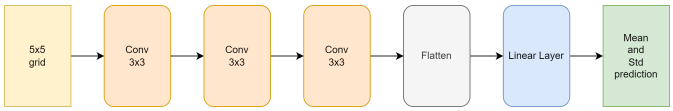


Figure 9: Internal structure of the probability estimation network

The last module in the network is an information aggregation network (figure 10), where all the latent features are aggregated and the final restored image is given as an output. The aggregation of latent features works by using transposed convolution layers (deconvolution) to bring all the features to the same shape then concatenating them channelwise and using convolution layers to change the number of channels back into the channels of the input image.

5.2.2 Transformer based Approaches

Transformer-based Image Compression [11] is a variational autoencoder architecture that makes use of transformers and convolutional layers to extract both short and long range spatial features. The encoder and decoder are made of Neural Transformation Units (NTUs). Each unit is comprised of a Swin transformer block and a convolution layer, in the case of the encoder the convolution acts to reduce the dimensionality of the data and in the case of the decoder a transposed convolution acts to increase the dimensionality of the data. The encoder outputs high-level latent features which are fed into the hyper-prior encoder, which will extract low-level latent features. The output features of the hyper-prior decoder are aggregated with the features from the main encoder by using windowed cross-attention (referred to as the casual attention module). The aggregated features are finally used to reconstruct the original image.

As TIC is a variational autoencoder, it combines two loss functions two optimise then encoder and decoder.

$$L = R(\hat{y}) + \lambda D(x, \hat{x})$$

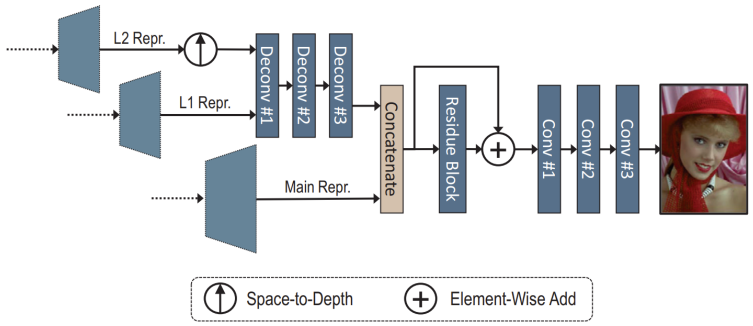


Figure 10: Aggregation of the 3 tensors of latent features [10]

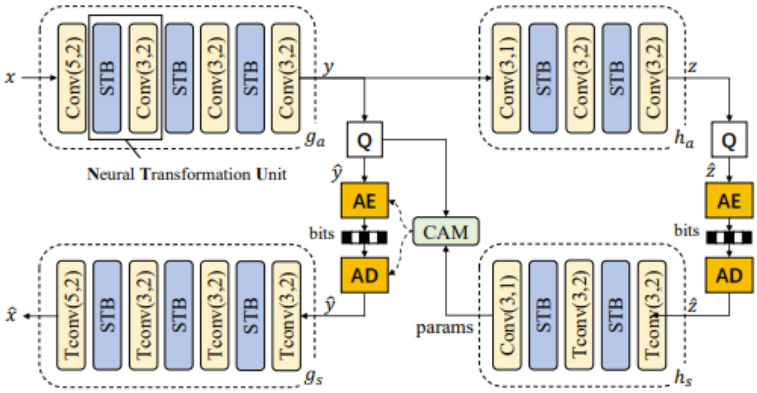


Figure 11: Architecture of TIC [11]

Where $R(\hat{y})$ is the compressed bit rate of the latent variables from the encoder, λ scales the rate distortion function to target different compression levels and $D(x, \hat{x})$ is the mean square error between the original and reconstructed image. The objective is to minimise both R and D, two functions which contradict each other as R requires a decrease in entropy to reduce the bit rate whereas D requires an increase in entropy in order to retain as much information as possible to reconstruct the original image.

TIC makes use of arithmetic coding for strong lossless compression of quantised latent variables from both the encoder and hyper-prior to create the final bit stream.

Towards End-to-End Image Compression and Analysis with Transformers [12] takes a look at using transformers in a bi-objective autoencoder architecture, with the objective to reconstruct the original image and correctly classify the subject of the image. As this architecture requires labelled data, the ImageNet dataset is used for training.

The encoder of this model is made from a set of convolutional layers which act to reduce the dimensionality of the image. The smaller dimension representation of the image is then quantised and sent through a ViT block. As the convolutional layers are reducing the dimensions of the image, the ViT block is not as memory intensive as it would be in the original ViT paper [13] (ViT explained further in section 6.3) as the height and width of the image is reduced by a factor of 16. The latent features returned by the encoder are arithmetically encoded for further lossless compression.

The decoder of this model arithmetically decodes the binary data to return the original latent features of the encoder. The latent features are passed through another ViT block before being passed through two separate output stems. The classification stem is a series of ViT blocks which lead to a classification head, this is used for training the model to classify the ImageNet images. The image reconstruction stem aggregates the output features of the first three ViT blocks from the classification stem with the original latent features. The aggregated features are then passed through a set of convolutional layers and transposed convolution layers which restore the height, width and channels back to the original dimensions of the input image.

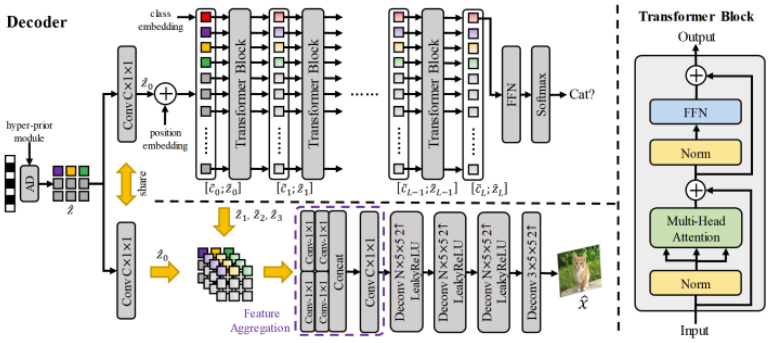


Figure 12: Model architecture proposed in 'Towards End-to-End Image Compression and Analysis with Transformers [12]

The loss function of this model is a combination of cross entropy loss, for the classification head, mean squared error, used for the image reconstruction head, and the rate loss, the approximate likelihood of the latent quantised variable $\hat{z}$.

$$L = \alpha \cdot CrossEntropy(y, \hat{y}) + \beta \cdot MSE(x, \hat{x}) - log(p(\hat{z}))$$

The paper explains that the mean squared error and cross entropy need to be scaled, using α and β as the mean squared error tends to return much larger loss values than cross entropy causing a bias towards reducing distortion and ignoring analysis error.

ViT blocks are used due to the findings of 'Learning Accurate Entropy Model with Global Reference for Image Compression' [14] which shows that global feature analysis can further improve image compression performance, a task that global self-attention does very well.

6 Architecture Design and Explanation

6.0.1 Basic Overview

Every compression technique I have looked at follows the same basic set of steps; transform, quantise and encode. My compression network will take on the role of transforming the image data. The encoder network (for compression) will reduce the dimensionality of the input data, promoting the learning of important information while removing any redundant or noisy information in the image [15]. Latent features returned by the encoder network will be quantised into 8 bit integer representations before being coded using arithmetic encoding. Quantisation is set to 8 bits as [16] shows that 8 bit quantisation of neural network activations and weights has little impact on network performance against top 5 accuracy with ImageNet. Furthermore 8 bit quantisation will allow a maximum of 256 unique symbols, whereas the output of my encoder will produce thousands of variables, therefore the arithmetic encoder will perform better as the final symbol set will have a symbols with a much higher probability of occurring [17]. Finally the data will be decoded using arithmetic decoding and de-quantised before being fed to the decoder network (for decompression) which will restore the data to the original dimensions of the input image.

I have chosen to build my network with a transformer centric approach as transformers are starting to show their dominance in other computer vision tasks such as classification and image segmentation [18] furthermore the two transformer based compression algorithms mentioned in section 5.2.2 use ViT blocks with successful results. Two transformer implementations will be used in my design; ViT (Vision Transformer) will be used for extracting long-range features from the image and Swin Transformer (Shifted Windows Transformer) will be used for extracting short-range features. Convolutional layers will be used for embedding data into specific dimensionalities at the start and end of the encoder and decoder. A common alternative to using transformers for an autoencoder architecture is convolutional layers, which are very good are extracting local features but do not consider, and can easily lose, long range features [19], both Swin and ViT supersede convolutions in this area.

6.1 Autoencoder

As the nature of my network is to return the same image it was passed, it follows the autoencoder design that is favourably used for learning features of data in an unsupervised approach. Autoencoders are made of an encoder, which reduces the dimensionality of the input data to promote the removal of redundant information and the learning of important information, a bottleneck, the name for the latent feature representation of the data at the midpoint of the autoencoder, and a decoder, which increases the dimensionality of the latent feature representation and returns an approximation of the input data [20]. The objective of an autoencoder is to adjust its parameters to minimise the loss between the input data and the output of the network

$$min_{\phi,\theta} L(x, D_\phi(E_\theta(x)))$$

$$L(x, \hat{x}) = \frac{1}{N} \cdot \Sigma_{i=0}^{N} (x_i - \hat{x}_i)^2$$

where $L(x, y)$ is a loss function (in the case of my network I am using mean squared error for distortion loss), E is an encoder function, D is a decoder function and θ denotes the parameters for either the encoder or decoder network.

My autoencoder architecture will use a convolutional layer to embed the input image, then pass it to the encoder made of feature extraction blocks. Feature extraction blocks transform the input variables using a swin transformer block and then reduce the dimensionality by halving the width and height and doubling the channel width. To stop the channel dimension of the bottleneck from becoming too large I will use a convolutional layer to collapse the final features of the encoder into a specific size, I call this the transfer dimension. The decoder will be composed a ViT block and feature restoration blocks to do the inverse of the features extraction blocks, increasing the dimensionality by doubling the height and width while halving the channels after every transformation. The output of the decoder will pass through a convolutional layer to return the number of channels to the original of the image (3 in the case of RGB images). The encoder and decoder will have the same number of blocks to ensure the same input and output shape. By splitting my encoder and decoder into blocks I can build my design easier by making my code take simple hyper-parameters to vary the amount of feature extraction and reduction blocks to find an optimal network depth.

The bottleneck of this design will also include quantisation and arithmetic coding to apply further compression after the dimensionality reduction.

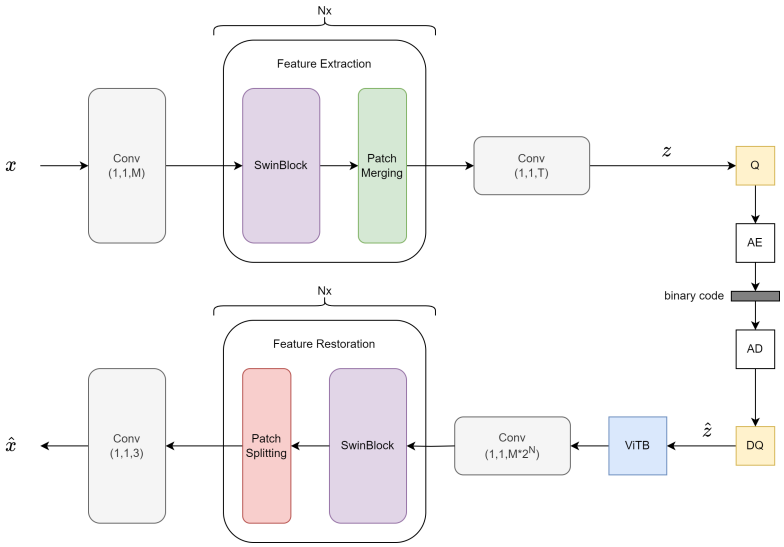


Figure 13: My proposed model architecture. Convolutional blocks are written as Conv (kernel size, stride, output channels). Q and DQ is quantisation and dequantisation. AE and AD is arithmetic encoding and decoding.

6.2 Transformers

Transformers are a neural network architecture originally made for natural language processing (NLP). Introduced in the paper 'Attention is all you need' [21], they are based around the self-attention function, which transforms a sequence of input tokens based on their attention scores of the other tokens. The input tokens are projected into 3 different representations before performing self-attention, these representations are known as the query, key and value, and are created by passing the tokens through a channelwise linear layer

$$k_i = W_k \cdot x_i, \quad q_i = W_q \cdot x_i, \quad v_i = W_v \cdot x_i$$

where $x \in \mathbb{R}^{n \times d}$ is a set of n tokens with dimensions d and $W \in \mathbb{R}^{d \times d}$ is a weight matrix for either the query, key or value.

Self-attention is then calculated as

$$a = v \cdot Softmax(\frac{q \cdot k^T}{\sqrt{d}})$$

Where softmax is applied long the rows of the $n \times n$ matrix. The transformer then applies a skip layer and layer normalisation when passing the transformed tokens through a final channelwise linear layer, finally including one last skip layer and normalisation layer. Skip layers reduce the vanishing gradient problem therefore helping training gradients backpropegate through the network.

$$z_i = (W_o \cdot (a_i \oplus x_i)) \oplus a_i$$

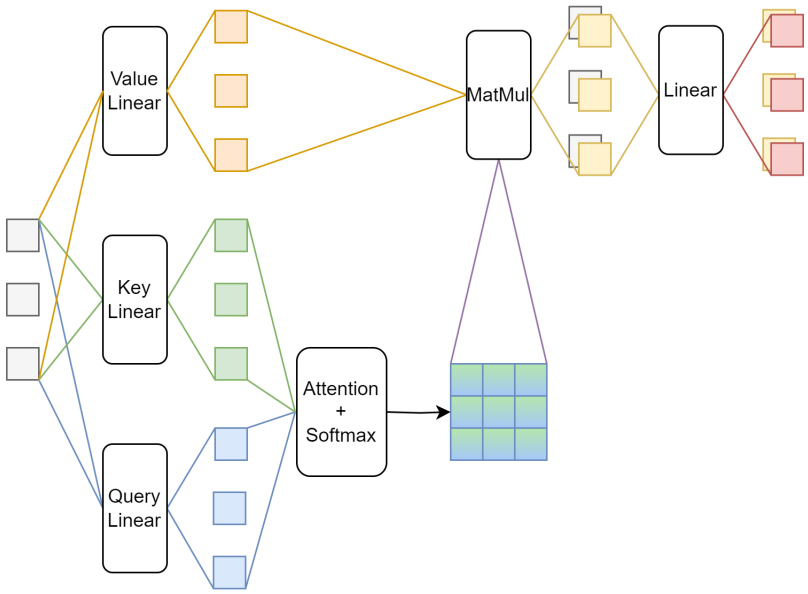


Figure 14: Diagram of transformer architecture. Each square is a input token or transformed token. Skip layers are shown as stacked tokens

6.3 Vision Transformer

The Vision Transformer (ViT) [13] is a deep neural network architecture that repurposes transformers for use with image data. Previous to ViT, the transformer was not a good architecture for computer vision as a large amount of tokens (in this case pixels) meant the attention mechanism was very costly in respect to memory consumption, this is due to the attention mechanism having a memory complexity of n^2. ViT addressed the memory complexity issue by using 16x16 embedded patches of the image as the input tokens rather than individual pixels, this significantly reduces the memory complexity to m^2 where $m = \frac{n}{16^2}$, although memory complexity will still be a problem as the resolution of the image increases. ViT performs very well with image classification but struggles where pixel level granularity is required such as semantic segmentation, this makes a purely ViT based compression network a poor pick. I will incorporate a ViT block before passing the compressed latent features to the decoder to enable the network to extract long-range spatial features. The ViT block is placed before the decoder as this is where the number of tokens will be at its lowest, due to the dimension reduction from the encoder, therefore the memory requirements of applying global self-attention on latent features will not be as severe as applying global self-attention on the full image.

While there are other vision transformers that apply global attention, they are all largely based on ViT with the most significant change being their training process. DeiT [22] is a vision transformer model that uses distillation from another pretrained network to train itself. While this approach is beneficial for reducing the amount of data needed to converge [23], it would require me to add a second output stem to my network and run a second network alongside it which would be create an increase in parameters and training time.

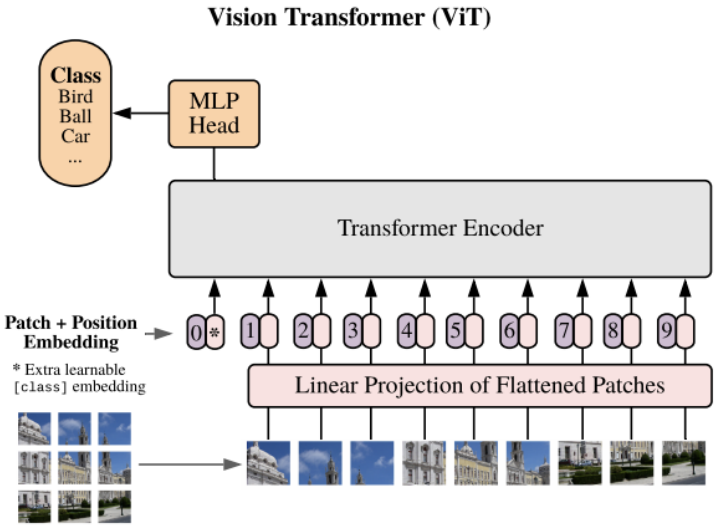


Figure 15: Diagram of ViT architecture, showing patches of images with positional embedding being used in a transformer architecture [13]

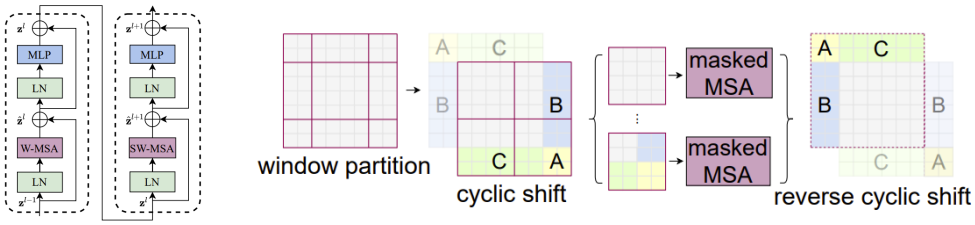
6.4 Shifted Windows Transformer

The Shifted Windows Transformer (Swin) [24] aims to address two major issues of ViT, the poor scalability due to m^2 memory complexity and the data granularity issue due to embedding large patches of an image into a single vector. Swin fixes these issues by applying windowed self-attention rather than global self-attention on images. Windowed self-attention splits the image into windows of $m \times m$ pixels and applies attention such that pixels can only see other pixels that are within their own window, this means you can get high levels of granularity due to pixel level attention but also limit the memory consumption of the attention operation as the complexity goes from n^2 to $\frac{n}{w} \cdot w^2$ where n is the number of pixels in an image and w is the flattened window size. Windowed self-attention alone cannot extract longer-range spatial features or features that happen to reside in between window boundaries, to address this Swin shifts the windows after applying windowed self-attention and applies attention once more. The windows shift by half their size and areas of the image that are no longer in the bounds of the windows will fold back around to the other corner of the image rather than using zero padding.

An alternative architecture to Swin for local attention would be localViT [25] which applies local attention without shifting windows. While localViT is more computationally efficient than Swin, it also is very dependant on using a good window size due to the very limited local scope. Swin is less dependant on using a good window size as the shifting windows create a longer range of spatial feature mapping.

6.4.1 Patch Splitting

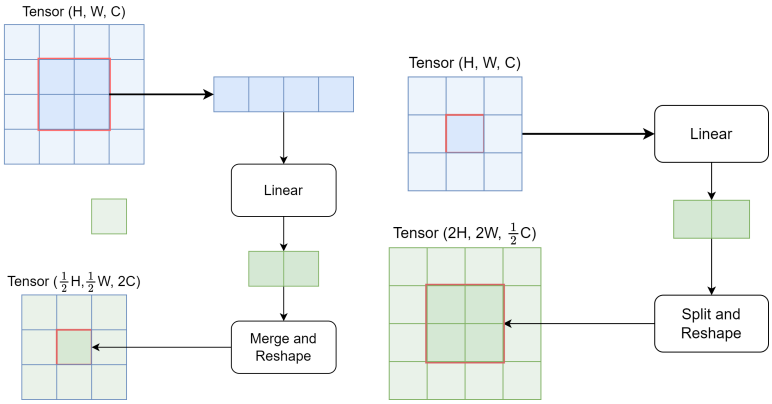
Swin uses a patch merging layer to reduce the dimensionality of an image, this works by concatenating 2×2 patches of features channelwise and passing them through a linear layer to half the channel size. I will largely use the Swin architecture as the backbone of the encoder and decoder due to its memory efficiency and high performance on granular data. I will make my own layer to be the inverse of patch merging called patch splitting (figure 17). Patch splitting will work by doubling the channel size of a feature and folding it into a grid of 2×2 features, each being made of $\frac{1}{4}$ of the single feature. While I came up with the idea of a patch splitting layer on my own, a few months into my project I discovered a paper that already created a patch splitting layer with the same functionality for a compression network



(a) Swin block architecture

(b) Window shifting example

Figure 16: Swin Diagrams [24]



(a) Patch merging

(b) Patch splitting

Figure 17: Swin Diagrams [24]

which was published a couple months before I started my project [26].

6.5 Quantisation

Quantisation is the movement from a continuous or discrete domain into a smaller discrete domain, this can reduce the amount of bits required to represent a number with the sacrifice of precision which can negatively effect the performance of a network. This makes quantisation a lossy form of compression. I plan to quantise the latent features returned by my encoder network from 32 bit floating point reals to 8 bit unsigned integers. Performing such drastic quantisation will not only reduce the storage size of the activations due to lower precision but will further reduce storage size due to the creating commonly occurring values which will result in more powerful arithmetic encoding as it has a reliance on frequently occurring values. Hard quantisation cannot be used during training as the gradient of the quantisation function commonly returns 0, and I will not be looking at approximation of quantisation functions such as soft quantisation [27], therefore quantisation will only be applied during testing with an expected performance loss. Quantisation of the latent features y is calculated as:

$$\hat{y}_i = round(255 \cdot \frac{y_i - min(y)}{max(y) - min(y)})$$

where $round()$ will round the result to the nearest integer. Once the decoder receives the quantised latent features it will dequantise them to an approximation of the original values:

$$\tilde{y}_i = min(y) + (max(y) - min(y)) \cdot \frac{\hat{y}_i}{255}$$

As neural networks are very good at generalising noisy data I expect the performance after quantisation to not degrade too much to the point of the compression network not working at all.

6.6 Arithmetic Coding

Arithmetic coding (also known as range coding) is a lossless compression method that encodes a sequence of data into a single decimal number ranging from $[0, 1]$. The algorithm makes use of a frequency table of the symbol set to encode larger probability symbols with less bits, much like Huffman coding. Arithmetic coding performs as good as, if not better than, Huffman coding but has the disadvantages of needing all the data in order to encode the final message, being very vulnerable to bit flips/message corruption as this would completely break the decoding step and the returned data would not be the original data, and needing high precision decimal numbers as more data is encoded [5]. The algorithm is split into an encoding and decoding step, although the calculations are very similar, as given by [28]. While I originally intended to build my own arithmetic coder, I found it much more productive to use a highly optimised C implementation for python [29].

6.6.1 Encoding

To encode a sequence of symbols in a given symbol set, the probability of each symbol occurring must first be calculated

$$p_i = \frac{f_i}{\sum_{j=0}^{n-1} f_j}$$

where f_i is the frequency of which symbol $0 \leq i < n$ occurs and n is the size of the set of symbols. The algorithm represents each symbol probability as an interval residing on a line that starts off bound by $[0, 1]$ represented as a start, s, and a width, w, of the interval. To calculate the interval of each symbol you need the cumulative probability

$$c_i = \sum_{j=0}^{i-1} p_j$$

where the base case is defined as $c_0 = 0$, and $0 \leq i \leq n$. As each value is encoded, the boundary is constricted to the interval of the encoded symbol. The updated start and width are calculated as

$$s(t+1) = s(t) + (w(t) \cdot c_i)$$

$$w(t+1) = w(t) \cdot p_i$$

This progressively constricts the boundary until all data has been encoded, the final encoding message is made up of the final start and width value given by

$$m = s + \frac{w}{2}$$

6.6.2 Decoding

Decoding a message reuses a lot of the methodology of encoding; it requires a cumulative probability distribution for each symbol and progressively constricts the boundary upon which those probabilities lie on. The symbols are decoded with first in first out, meaning the message ‘hello‘ encoded with ‘h‘ first will get ‘h‘ back as the first decoded letter. The difference between encoding and decoding is in the interval constricting step. The next decoded symbol is found by identifying which cumulative probability interval the encoded message resides within. The decoded symbol appended to the final decoded message and is used to update s and w using the same update equations as the encoder (constrict the probability distribution line).

7 Architecture Implementation

7.1 Encoder Implementation

The encoder was very simple to implement as it is functionally the same as the backbone of the Swin architecture with an added convolutional layer at the end to transform the output features to the correct transfer size. Due to this similarity my encoder simply needed to inherit the 'SwinTransformer' class included with PyTorch and overwrite the forward function to include the extra layer and a final normalisation layer to improve the training by keeping gradients smooth [30]. The testing version of the network contains the quantisation step after the encoder which enlists the equations shown in section 6.5.

7.2 Decoder Implementation

The decoder includes a ViT block and uses patch splitting which meant it made more sense to rewrite a lot of code from the swin transformer with the required changes instead of inheriting. The ViT block is implemented before the transformation from the transfer size to the embedded size. I made this decision as the transfer size is generally significantly smaller than the embedded size, therefore the computational time and amount of trainable parameters required of the global attention would be significantly reduced. The patch splitting layer is placed after each decoder swin block and is implemented as described in section 6.4.1. The output head of the decoder transforms the output of the Swin style backbone into a $H, W, 3$ shaped tensor and activates it with the hyperbolic tangent function to transform all values into the range of $(0, 1)$ as this domain can later be easily scaled to $[0, 255]$ for image visualisation.

7.3 Training and Testing

Training, Validation and Testing data was sourced from a subset of ImageNet, a dataset containing a wide variety of images categorised with 1000 classes labelling what the subject of the image is. I used ImageNet due to the large corpus of images with differing subjects and the security and accessibility on the AI@Surrey HPC. The full ImageNet dataset contains 1.3M images but training on this many images would take too long for the amount of models I would like to test with so I created subsets of the dataset. Training data was taken as the first 100,000 images, validation data was the next 25,000 images, giving me a 20% training-validation split; finally testing data was the subsequent 50,000 images. Each image was resized to a resolution of 224x224 and only the training images were randomly shuffled.

I used the Adam optimiser due to its fast convergence with an adaptive learning rate and momentum estimation [31]. I chose a learning rate of 1e-4 as it showed better convergence in early training in comparison to 1e-3 but not much more testing was done in this area. Loss was calculated using the mean square error between the output and the original input image.

$$MSE(y, \hat{y}) = \frac{1}{n} \sum_{i}^{n} (y_i - \hat{y}_i)^2$$

I also used the peak signal-to-noise ratio (PSNR) as a second metric to track the performance of my networks which compares the error between the output and target against the maximum possible error. This metric is a great way to measure the quality of a generated image in comparison to the target image and is commonly used in other compression network research papers. As the maximum possible error of my reconstructed image could be 1 (as all pixel values were scaled to be within 0-1, the PSNR can be calculated by

$$PSNR(y, \hat{y}) = 10 \cdot \log_{10}(\frac{1^2}{MSE(y, \hat{y})})$$

$$PSNR(y, \hat{y}) = 20 \cdot \log_{10}(1) - 10 \cdot \log_{10}(MSE(y, \hat{y}))$$

$$PSNR(y, \hat{y}) = -10 \cdot \log_{10}(MSE(y, \hat{y}))$$

All networks were trained with images in batches of 32 for 50 epochs.

7.4 Neural Architecture Search: Stage 1

In order to find the optimal architecture I needed to choose what could be changed about the architecture. I came up with 4 variables to effect the architecture: embedding dimension, transfer dimension, window size (for windowed attention in Swin) and depth. The embedding dimension effects the channel size of the input image tensor. Transforming data into a higher dimensionality can better separate features of the data but too high of a dimensionality can make the features too sparse. Transfer size, much like embedding dimension, changes the channel size of the output features of the encoder. The size of the output features will be very important as it will effect both the information available to describe the image and the final file size of the compressed image. The window size of the self attention mechanism in the Swin transformer was chosen as it will demonstrate the trade off between computational and memory complexity against final image quality improvements. Finally the depth of the network will change how many encoder and decoder blocks are used and what the height and width of the encoder output is. A deeper network would be able to output a larger set of channels than a shallower network as the total amount of bytes would remain the same for example, an output of $(28, 28, 1)$ and $(14, 14, 4)$ would have the same total of items (784) but may have different decompression performance. Depth will also greatly effect the amount of parameters that the model needs to train and this will be taken into account with final performance.

7.4.1 Grid Search

I had to make a small selection of hyper-parameters to test with due to time constraints and needing a quick set results to analyse so I limited myself to 11 models. The hyper-parameters I chose can be seen in figure 1. The choices for embedding size were chosen to find the effect on performance and memory consumption from having a small embedding size (24), medium size (48) and very large size (96). The choices for transfer size were chosen to see the effect of a very large feature size (256) to very small (32), the small features size was only chosen to be present on one network which represented a scenario of using a relatively resource light model (model ID 0, which only had 1M trainable parameters). Having a variety of different transfer sizes would demonstrate how well my model can compress information into smaller dimensions of latent variables. The window size was constrained to only 4 and 8 as an increase in window size quadratically increased the memory complexity during execution, for this reason I also only compared the larger window size of 8 in scenarios of larger models, specifically any model with a transfer size of 256 or embedding dimension of 96, equivalent models were made for a window size of 4 as to keep the other variables static. Finally the depth was changed between 3 and 4 encoding and decoding blocks. This was chosen as it meant the deeper models could comfortably fit into the GPUs I was using (3090s) and the other parameters would still be changed to a large variation. The models with an embedding dimension of 96 (model ID 9 and 10) were made as an experiment of the performance of a very wide network that wasn't as deep as others.

7.4.2 Results and Reflection

When looking at models that only change their embedding dimension, for example models 1, 5 and 9, there is a menial increase in image quality. Furthermore there is evidence that an increase in embedding dimension has less effect on the image as the dimension becomes larger; an increase from 24 to 48 produces a 0.9 dB increase but from 48 to 96 there is only a 0.5 dB increase, showing the improvement in image quality does not scale linearly. A drawback of larger embedding sizes is the significant increase in parameter count; the models generally follow the rule that doubling the embedding dimension quadruples the number of parameters which offsets the benefit of the performance increase.

Model ID	Embedding Size	Transfer Size	Window Size	Depth
0	24	32	4	3
1	24	64	4	3
2	24	64	4	4
3	24	256	4	4
4	24	256	8	4
5	48	64	4	3
6	48	64	4	4
7	48	256	4	4
8	48	256	8	4
9	96	64	4	3
10	96	64	8	3

Table 1: First wave of NAS hyper-parameters

Model ID	Test PSNR (dB)	Parameters
0	31.4	1.0M
1	36.1	1.0M
2	28.8	3.4M
3	34.2	4.2M
4	34.2	4.2M
5	37	3.7M
6	29.2	13.0M
7	12.1	13.8M
8	12.1	13.8M
9	37.5	14.1M
10	37.6	14.1M

Table 2: Performance of models in Table 1 and trainable parameter count

The transfer dimension is the bottleneck that decides how much information the encoder can send to the decoder to describe the image, therefore it makes sense that a larger transfer dimension would increase performance. Models 0 and 1 show that doubling a small transfer dimension results in a large increase of image quality going from 31.4 dB to 36.1 dB with a relatively small change in number of parameters. Models 2 and 3 also show a large increase in performance although the parameter count is also increased by roughly 0.8M parameters. It should be noted that the transfer dimension directly effects the compression ratio and therefore larger transfer dimensions may not be possible to achieve low bits per pixel.

The results with respect to window size may show a link to the transfer size. Models 3, 4, 7, and 8 demonstrate no change in image quality with a larger window size, but they all have a large transfer dimension of 256. When we look at 2 models with a smaller transfer dimension of 64, models 9 and 10 see a increase in PSNR of 0.1 dB when the window size doubles. A minimal increase such as this could easily be due to differently initialised weights or the shuffling of training data causing the model to converge differently so further investigation may be needed.

The final factor, depth, has a strong relationship to transfer dimension and parameter count. Model 1 and 2 show that increasing the depth of the network by 1 almost quadruples the parameter count while greatly decreasing the final image quality when transfer size is kept the same. A decrease in image quality like this should be expected as each depth halves the height and width of the features so not increasing the transfer dimension will decrease the overall amount of information passed to the decoder. When we look at an example of scaling the transfer dimension with depth, models 1 and 3 may show that a shallower network is more beneficial as model 1 outcompetes model 3 by 1.9 dB. More investigation into scaling transfer dimension with depth should be done to see the effects at different parameters.

Models 7 and 8 should be noted as anomalous results, greatly underperforming against all other networks. This underperformance may be due to a characteristic of the hyperbolic tangent function as the training gradient will move to 0 as a neuron becomes saturated, preventing convergence. More tests should be run to check if this problem persists.

7.5 Transfer Dimension and Compression Ratio

After my initial set of models there was a need to check that the output sizes I was testing would actually compress the image to the 0.5 bpp that I was targetting. I originally made my own version of arithmetic coding but when I found it ran far too slowly for large input sizes I decided to implemented arithmetic coding using another developers highly optimised C implementation [29]. The file sizes that were being output for tensors of size $(28, 28, 64)$ were far too big, proving that my next set of networks would need to target a much smaller transfer dimension. With further testing a maximum transfer size of 4 at a depth of 3 would hit the required levels of compression. Every time the depth was increased by 1 the maximum transfer size could be quadrupled, this was kept in mind for the next stage of hyper parameter tuning. My testing of the arithmetic encoder also found that a target of 0.5 bpp would require the neural network encoder to output 3136 elements or less, for more bits per pixel the amount of elements that can be output roughly scales linearly, for example to get 1 bpp the number of elements is equal to $2 \times 3136 = 6272$. This means I can calculate the transfer dimension for any of my networks with:

$$6272 = \frac{H \cdot W \cdot T}{bpp}$$

$$T = \frac{6272 \cdot bpp}{H \cdot W}$$

where H and W are the height and with of the latent features returned by the encoder, bpp is the bits per pixel target of the model and T is the transfer size of the model.

Model ID	Transfer Size	Test PSNR (dB)	Parameters
0	1	12.0	4.1M
1	2	12.0	4.1M
2	4	23.0	4.1M
3	8	26.1	4.1M

Table 3: Transfer size and performance of new set of models

Model ID	Embedding Size	Transfer Size	Window Size	Depth
0	24	4	4	3
1	24	4	8	3
2	24	16	4	4
3	24	16	8	4
4	24	64	4	5
5	24	64	8	5

Table 4: Third wave of NAS hyper-parameters

7.6 Neural Architecture Search: Stage 2

This NAS stage would concentrate on changing the transfer dimension only as I expected the effect of small increases to be much more pronounced during this training session. All models shared the same embedding dimension (48), window size (4) and depth (3). The transfer dimensions of the models can be found in Table 3.

7.6.1 Results and Reflection

All models reinforce the conclusion of stage 1 that increasing the transfer size creates a significant improvement with image quality. Unfortunately the drop in transfer size in comparison to the models in stage 1 has resulted in a large drop in image quality for example model 5 in stage 1 had a PSNR of 37 dB with a transfer size of 64 whereas model 3 in stage 3 had a PSNR of 26.1 dB.

Similar to models 7 and 8 in stage 1, models 0 and 1 seem to have had trouble converging. The model was very slightly changed for this training session to place the ViT block after the embedding step of the decoder, meaning the attention operation would handle larger token vectors. I believe this caused more issues and as I will not be using such small transfer sizes, as these model have, in the future I have changed the ViT block back to operate before the embedding.

7.7 Neural Architecture: Stage 3

Stage 3 will be a very important stage, we have identified the relationship between transfer size and compression ratio and further experimentation is needed for the relationship between depth, transfer size and performance. Furthermore stage 1 suggested that there may be an image quality increase when window size is larger for very small transfer dimensions. This means depth, transfer size and window size will be the main hyper parameters to focus on in this stage. I will also be focusing on an encoder output size that can be compressed to approximatly 0.5 bpp, specifically the flattened latent variable size must be $H \cdot W \cdot T = 3136$.

7.7.1 Results and Reflection

A strong and clear trend is shown with these results that scaling up the depth and transfer size increases image quality. As the depth is increased by 1 and transfer size is quadrupled, the subsequent PSNR

Model ID	Test PSNR (dB)	Parameters
0	22.7	1.1M
1	22.7	1.1M
2	24.5	3.8M
3	24.6	3.8M
4	12.1	14.4M
5	25.0	14.4M

Table 5: Performance of models in Table 4 and trainable parameter count

increases by 1.8 dB and 0.5 dB from model 0 to 2 and 2 to 5. As mentioned in stage 1 the models follow the rough rule of quadrupling the parameters for every extra encoding and decoding block added due to depth, this makes models 4 and 5 very large at 14.4M parameters. Window size has very little effect on the performance of the model. While model 1 did perform better than model 0 it was only an improvement of 0.03 dB. Model 3 also performed better than model 2 which could show that window size has a small but present improvement on image quality. When models are small enough to run with the extra window size it may be worth using for the small improvement. Window size also still has little effect on model size. Model 4 did not converge similar to the non-converging models in previous stages.

7.8 Final Model Choices

I decided to choose three different models to target three different compression levels; 0.5, 1 and 2 bits per pixel. I chose these three qualities as I could easily compare them with a jpeg equivalent and it would give me the opportunity to see how my network scaled with different levels of compression.

The first model will target 0.5 bpp so I will expect it to have the lowest PSNR of the three models. As the level of compression is much stronger than the other two models, requiring more powerful transformation of features, I will be making it the largest in terms of trainable parameters. The depth will also be large at 5 blocks for the encoder and decoder as I have shown more depth creates better image quality with a drawback of a larger model. As the depth will be large I will also need to keep the embedding dimension small due to the dimension scaling by $m \cdot 2^d$, where m is the embedding dimension and d is the depth, as it passes through the sequential encoding blocks. Finally I will use a window size of 8 as, although a weak connection, there is a connection between larger window size and higher image quality. I obtained the correct transfer dimension of 64 by using the previously mentioned equation:

$$T = \frac{6272 \cdot 0.5}{7 \cdot 7} = 64$$

The model targetting 1 bpp will have parameters that sit between the other two models. The embedding dimension will be 36 as, while there is little improvement for large embeddings, an increase in small embedding sizes does show a small improvement in image quality. The depth will be set at 4, using the idea that less compression should require less parameters means the depth needs to be reduced as the biggest contributor to model size. The window size is kept at 8 for the same reasons as the first model. Finally the required transfer size for a target of 1 bpp is 32.

$$T = \frac{6272 \cdot 1.0}{14 \cdot 14} = 32$$

The last model will provide the least compression, therefore it should also be the smallest of the three models. The embedding size is set to 48 as the small depth will limit the feature size as the image passes through the network. The depth is set to 3 as much less transformative power is required for the lesser compression and less depth cuts down on trainable parameters. The window size is dropped this time

as a size of 8 became too large when training, this is due to the larger feature sizes at the start of the network. Finally the transfer dimension of this model is calculated as:

$$T = \frac{6272 \cdot 2.0}{28 \cdot 28} = 16$$

8 Testing

8.1 Experimental Setup

The final models were trained with the same setup as mentioned in section 7.3, a summary can be seen in Table 6. Each Swin block had 4 attentions heads in and all blocks were kept shallow at a depth of 2 aside from the penultimate block which was given a depth of 6 for better low level feature extraction, this was kept the same in all models. I used the AI@Surrey HPC servers to get access to higher VRAM amounts needed to train my models. I used 1xRTX 3090 for each training session and a full 50 epochs took roughly 36-40 hours (time varying on model configuration). Quantisation is applied post training and the test dataset is passed through the model again for gathering metrics on quantisation effects. Random samples from the test set were extracted for qualitative visual analysis of each models performance, the 3 final photos were selected for their unique attributes (explained further in section 8.2.2). During the training,

Dataset	Optimiser	Learning Rate	Loss Metric	Batch Size	Epochs	Input Resolution
ImageNet	Adam	1e-4	MSE	32	50	224x224

Table 6: Experimental setup configuration

validation and testing phase the average PSNR and loss were tracked for later analysis of each models ability to learn.

8.2 Model Results

I trained 3 final models targetting the respective bits per pixel of 2, 1 and 0.5. The specific implementation of each model is talked about in section 7.8, each model has a different amount of parameters and layers adjusted to the different levels of compression power that they are expected to provide.

During the training phase, the average PSNR of the training and validation set was recorded to ensure that my models were not over-fitting or under-fitting. The results of the PSNR scores gave a very useful insight into the learning of each model. All models displayed a higher average PSNR for the validation dataset over the training dataset, despite having a lower average loss. Furthermore, both the training and validation PSNR scores kept increasing even during the later epochs, this suggests either more epochs of training is needed (since 50 epochs is relatively low in comparison to similar research papers, for example [11] uses 400 epochs) or a higher learning rate should have been used.

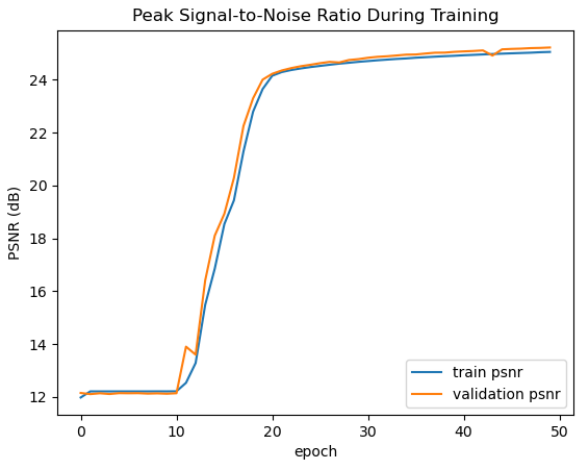
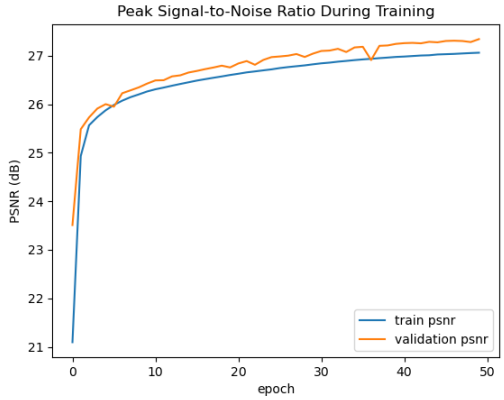
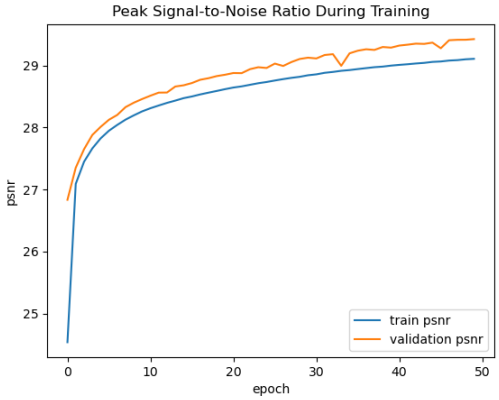


Figure 18: Training and validation PSNR of 0.5 bpp model



(a) Training and validation PSNR of 1 bpp model



(b) Training and validation PSNR of 2 bpp model

Figure 19: 2 bpp and 1 bpp model results

The largest model targetting 0.5 bpp (Figure 18) stayed at a very low plateau of 12 dB for the first 10 epochs, this lack of learning may be due to a low learning rate problem causing the model to take a long time to find a optimal minima, alternatively the activation function, hyperbolic tangent, could have struggled to learn at first due to the training gradient being very close to zero at very large or very small input values. The model quickly increased to 24 dB in the next 10 epochs and learning slowed but did not stop for the last 30 epochs. The final average validation PSNR was 25.2 dB.

The smaller models, targetting 1 bpp (Figure 19 (a)) and 2 bpp (Figure 19 (b)) both converged much faster than the larger model (Figure 18) but still followed the trend on continuous learning all the way to 50 epochs and having the validation set performing better PSNR results than the training data. The performance gap is generally larger when a model has less trainable parameters, suggesting that the model may be under-fitting.. A larger model could capture the complexity of the images better and fit the data better. The validation PSNR score of the 1 bpp model ended at 27.3 dB and the 2 bpp ended at 29.4 dB.

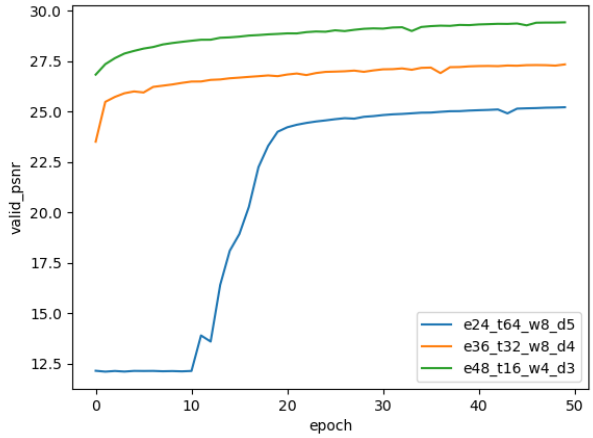


Figure 20: Validation PSNR of 2 (green), 1 (orange) and 0.5 (blue) bpp models

By comparing the three final models it can be seen that, despite less trainable parameters (Table 7) and less layers, the networks which apply less compression are able to rebuild the original image much better. The better performance of the higher bit per pixel networks can largely be accredited to their

Model bpp	Test PSNR (dB)	Parameters
0.5	25.3	12.7M
1	27.2	7.4M
2	29.2	3.6M

Table 7: Performance of final models and parameter count

(a) Image without bottleneck quantisation (25.3 dB) (b) Image with bottleneck quantisation (25.2 dB)

Figure 21: Example of quantisation effect (2 bpp model used for images)

ability to retain much more information about the image in their latent representation in comparison to the lower bpp networks; for example the 0.5 bpp network has a latent representation of (7, 7, 64) totalling 3136 elements, whereas the 2 bpp has a latent representation of (28, 28, 16) totalling 12544 elements. It can also be seen that the 0.5 bpp network took longer to converge to an optimal solution in comparison to the other networks, this could be due to the larger parameter count combined with the smaller latent representation forcing the network to have to search more for an optimal solution.

8.2.1 Effect of Quantisation

In section 6.5 I theorised that quantisation will have a negative effect on the quality of the output image due to to loss of information when moving from a large discrete domain into a small one. Despite these negative effects I still assumed that the quality reduction would not be too much as neural networks are very good at generalisation and are therefore very robust to noise. As can be seen in Table 8 my assumption of a small drop in image quality was correct for the 2 and 1 bpp models but the loss decreased and PSNR increased for the 0.5 bpp model. The performance of the 0.5 bpp model after bottleneck activation quantisation may be higher as large neural networks tend to respond much better to quantisation than small neural networks [32] and can perform as good as if not better than the un-quantised models in some cases.

A visual analysis of the effects of quantisation show, while the image is slightly different, it is very difficult to say whether one version of the image is better than the other. The example (Figure 21) of the two slugs shows the effect of bottleneck quantisation on the 2 bpp model; this model was selected as it shows the largest negative effect due to quantisation when referring to loss and PSNR. The quantised

Model bpp	Loss 32-bit	Loss 8-bit	PSNR 32-bit	PSNR 8-bit
0.5	3.28	3.25	25.13	25.28
1	2.06	2.07	27.19	27.15
2	1.29	1.31	29.22	29.15

Table 8: Effect of bottleneck quantisation on average loss ($\times 10^{-3}$) and PSNR (dB)

image shows some very small changes such as a slight change in the shade of orange around the edge and eyes of the slug.

8.2.2 Comparison Against JPEG

The photos I chose to compare against JPEG were selected to show scenarios where both JPEG and my networks perform well. The first photo of a sea slug was selected as it shows a strong suit of JPEGs ability to represent images as low frequency cosine waveforms, a characteristic which will map the pattern on the slug's back very well. The second photo of a man holding a crab was chosen to have features that benefit both compression schemes, specifically the texture of the crab for JPEG and the wide background with the blurred face for my models. Finally the 3rd model was chosen to show the strong abilities of my models, the image has clearly been digitally edited and will require a good understanding of natural shape and colour to maintain a sharp boarder between the coloured and greyscale aspects of the photos.

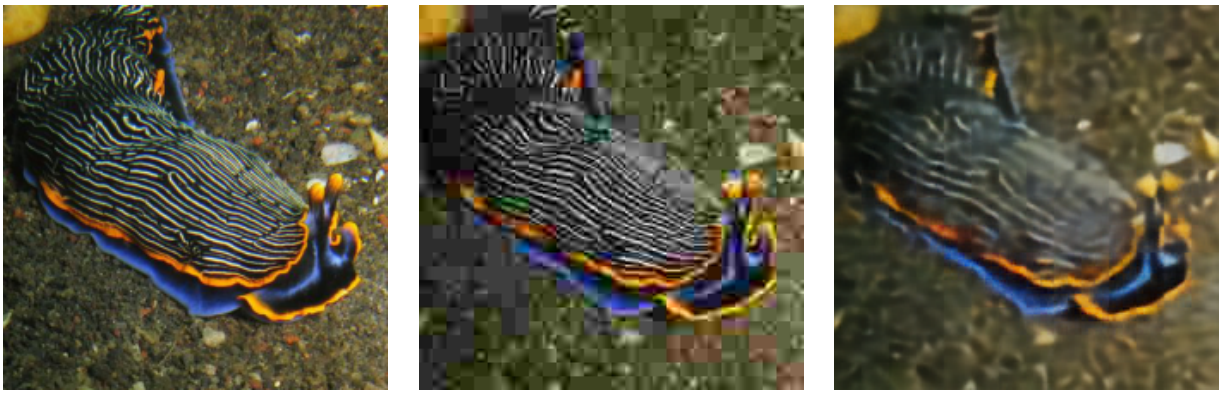
To compare my 0.5 bpp model to JPEG, I used a JPEG quality of 6 when saving the original image as this resulted in a very similar file size ($\pm 3\%$). In all cases my network outcompeted JPEG compression quality from 0.1-2.8 dB.

JPEG retains a lot of quality from the slugs back, this makes sense as the pattern on its back is very similar to the patterns produced by the lower frequencies of the discrete cosine transform, my model does better than JPEG in the colour accuracy and background of the image. The background of the JPEG image is very low quality in both sharpness and colour accuracy, with blocks of green, grey and red. My model produces a much more natural background that is much more colour accurate, furthermore my model has a smoother look, as opposed to the blocky nature of jpeg, which works very well with the background. The overall PSNR values of the slug photo were very close at 21.0 dB with JPEG and 21.1 dB with my model.

The photo of the crab displays the strengths of both compression schemes with JPEG retaining a lot of detail of the crabs shell but struggling with the blurred face and background of the image. My model loses a lot of the crabs detail, displaying a very blurry render without any of the small bumps of the shell. My model shows its strengths with the blurred face and background, as it displays a much smoother and more natural image and the background colours are much closer to the original than the JPEG background. My model beats the JPEG version in PSNR with 25.4 dB against 24.2 dB.

Finally JPEG struggles with the image of the bird where the colours sharply change between vibrant and grey, this results in a very blocky photo with pink colouring unable to follow the complex shape of the wings. My model greatly outperforms JPEG in this scenario due to its high colour accuracy and ability to generate more natural shapes rather than blocks of colours. The network still struggles to fill in the subtle colour of the beak and legs and loses a lot of granular detail resulting in a blurry photo. My model scores 29.6 dB against JPEGs 26.8 dB for the recreation of the bird.

A JPEG quality of 21 was used for 1 bpp comparisons against my network. The PSNR values for all images were closely matched within less than 1 dB difference.



(a) original (b) jpeg: 21.0 dB (c) my model: 21.1 dB

(d) original (e) jpeg: 24.2 dB (f) my model: 25.4 dB

(g) original (h) jpeg: 26.8 dB (i) my model: 29.6 dB

Figure 22: Results of 0.5 bpp compression network against original and comparative jpeg image of same file size

(a) original (b) jpeg: 24.0 dB (c) my model: 23.1 dB

(d) original (e) jpeg: 28.9 dB (f) my model: 28.1 dB

(g) original (h) jpeg: 32.1 dB (i) my model: 32.8 dB

Figure 23: Results of 1 bpp compression network against original and comparative jpeg image of same file size

Model	Bits per Pixel	Input Resolution	PSNR (dB)	Parameters (M)
TICT	0.55	224x224	30.4	25.6
Mine	0.5	224x224	25.3	12.7
TICT	0.71	224x224	31.7	25.6
Mine	1	224x224	27.2	7.4

Table 9: My model compared to the model built in 'Towards End-to-End Image Compression and Analysis with Transformers'

The visual quality of the slug is much better with JPEG compression than my model retraining more detail of the slugs back and displaying a much sharper background. The JPEG image still outputs artefacts of discoloured blocks of pixels whereas my model creates artefacts of heavily blurred portions of the image (most notable on the slug's back left side).

The image of the crab has the same strengths and weakness as the lower quality JPEG and model scheme but much less pronounced. JPEG has displays the texture of the crab very well and has much less pronounced discolouration of the background than the lower quality, but still has blocky artefacts most notable in the blurred face. The network has managed to retain much more detail in the crab than the 0.5 bpp model but still has significantly less detail than the JPEG image.

The image of the bird is the only case where my model does better in PSNR performance than JPEG. My model still does much better at reproducing the colours with a much smoother tone, this also negatively effects the image as the eye of the bird is much less present than in the JPEG version.

To target 2 bits per pixel, a JPEG quality of 66 was chosen. At this higher quality, JPEG produces a much better photo than my model in all cases.

My model reproduces the features of the slugs back much better than the lower bpp versions of the model but still has a lack of sharpness, especially in the reflection of the light on the slug's back. The JPEG image of the slug is very close to the original with only minor artefacts.

The JPEG photo of the crab has visually no differences besides small artefacts whereas the image generated by my network struggles to reproduce the crab, rope and tube features to a high level of granularity, resulting in blurry looking objects.

The image of the bird is high quality in both the JPEG and network case with the network only struggling on the small features of the image; for example the eye, legs and feet of the bird are all desaturated in comparison to the original and JPEG versions of the image. The upper wing also does not hit the vibrant pink colour of the original image, instead producing a much darker shade of pink. My model achieves much lower dB than JPEG between 1.2-2.5 dB less across the three examples.

8.2.3 Comparison Against Another Compression Neural Network

I chose to compare my network against the model built in 'Towards End-to End Image Compression and Analysis with Transformers' [12] (referred to as TICT for simplicity) as it is trained and evaluated on the same dataset (ImageNet) as my approach. I am comparing my model in similar bits per pixel and at the same image resolution. All results, found in Table 9, were taken from the original research paper. When looking at 0.5 bits per pixel quality, TICT out performs my model by an average of 4.8 dB but my model is significantly smaller, with around half the amount of trainable parameters. TICT did not report on less powerful compression ratios such as 1 bpp so my second comparison looks at roughly 0.7 bpp against 1bpp. TICT, despite achieving a greater level of compression, also achieved a better image quality at 31.7

(a) original (b) jpeg: 26.4 dB (c) my model: 25.2 dB

(d) original (e) jpeg: 32.9 dB (f) my model: 30.4 dB

(g) original (h) jpeg: 36.4 dB (i) my model: 35.0 dB

Figure 24: Results of 2 bpp compression network against original and comparative jpeg image of same file size

dB against my models 27.2 dB however my model is significantly less complex at only 7.4M parameters against their 25.6M parameters.

TICT trained their network over the entire ImageNet dataset with 300 epochs and a batch size of 1024, giving their network more time and data to learn the features of the dataset. Furthermore TICT uses a hyper-prior in their model to extract lower level features. A hyper-prior, in the context of learned image compression, is a secondary VAE that learns lower level features of an image. The low level features are sent to a hyper-prior decoder which will restore them to higher level features, finally both sets of features are aggregated to strengthen the quality of the rebuilt image. As mentioned in the model results section 8.2, a model with more parameters may have been able to map the complex features of the data better, the results of TICT reinforce this theory. TICT also made strong use of both transformers and convolutional layers, this may have helped the network as convolutional layers are very good at understanding local features of an image which could help the blurriness of my models images, as I mentioned in my JPEG comparison (section 8.2.2), by synthesising sharp pixel-granularity features.

8.3 Final Analysis

As predicted, quantisation of the bottleneck activations results in a minimal change in image quality, with the PSNR of my largest model increasing after quantisation. My network produces high quality compressed images at high compression ratios, particularly 0.5 bpp, beating a JPEG equivalent by 0.1-2.8 dB signal distortion. As the compression ratio increases my network starts to be out competed by traditional compression (JPEG) both quantitatively and qualitatively with JPEG beating my model at 2 bpp by 1.2-2.5 dB signal distortion. In comparison to another neural network, my model achieves a lower PSNR in both high and low compression ratios. However my model is also significantly smaller, between 50-25% of the size of the comparison model, depending on compression ratio.

9 Ethics

9.1 IP and Copyright Law

ImageNet contains a dataset of images from public web sources. ImageNet does not own the copyright to the images, however, UK law states that use of limited extracts of content for non-commercial use, in this case research, is exempt from copyright without requiring permission from the owners of the images [33], furthermore as my research paper will not impact the financial status the the original copyright owners I am respecting the copyright of their work. The Swin transformer [24] and ViT [13] architectures come with a license agreement for commercial use, modification, distribution and private use; as I am modifying Swin for my autoencoder I am working within the license agreement.

9.2 Do No Harm

Image compression algorithms have a responsibility to reliably decompress data to a sufficient quality such that users will not lose important data, such as memories or images of important documents. I have been open that my compression method is lossy in nature and as such will lose a level of detail in each image. Neural networks can only perform as well as the data they are given, bias in the images can result in the network performing worse in specific case scenarios. To prove that my networks can perform in unseen scenarios without losing a significant level of detail I have tested both quantitatively and qualitatively that the output is representative of the original images before compression, with decompression qualities better than or similar to JPEG compression. As mentioned in section 9.1 I have followed the legal requirements for all third party data and algorithm use. As all human faces within that dataset I have used were blurred, a deployed version of these model would need to be further trained on a dataset with human

faces taking care to securely store this data and provide equal representation of ethnicities and races to ensure high quality data restoration for all users.

9.3 Informed Consent

No participants were used during the development of this paper, therefore informed consent was not required from any third parties.

9.4 Confidentiality of Data

The image data used in this project for training came from the ImageNet dataset with obfuscated faces. The Data Protection Act [34] requires data to be used for explicit purposes that are necessary for my project. I used ImageNet for the specific requirements of training and testing data with my image compression networks, therefore I did not use any of the image labels and did not access any other data. All data should be handled with appropriate security, for this reason I accessed the images through the AI@Surrey servers, which can only be accessed by users with correct permissions and a unique private key for each user which I have safely stored on a password protected device. The image data from ImageNet does not contain personal or identifiable data as the current version of the data has blurred all faces of people found in the images.

9.5 Social Responsibility

All research should strive to benefit society. My project looks at using machine learning as a better compression algorithm than traditional approaches as a solution to the ever growing volume and velocity of data that is being generated today. My strongest compression model proves that machine learning provides better decompression quality than JPEG at low bit per pixel rates, this could be used to reduce that amount of data storage required in enterprise servers, which would subsequently reduce hardware and energy use, making data storage more environmental. Neural networks are largely black box systems with and researchers have little understanding of what conclusions the networks have made from the data they're trained with, therefore my compression system has the potential to be misused as a ransomware system as only the user with the decoding network could restore the original images and it would be challenging to approximate the necessary parameters needed to correctly decompress the data.

9.6 BCS Code of Conduct

9.6.1 Public Interest

To have due regard for public health, privacy, security and wellbeing of others and the environment. My compression programs do not not store any images in locations other than specified from the user as to keep all images private. ImageNet images were accessed from a secure server with private key access. My model does not effect public health, the environment or wellbeing.

To have due regard for the legitimate rights of third parties. As mentioned in section 9.1 the ImageNet images, while under copyright by their original owners, are used within legal requirements as my project is research based and not using the images for commercial reasons. All work used for research and implementation during the entire report has been credited.

To conduct your activities without discrimination. All people involved in the creation of this report, including opinions, assistance and further help; were treated equally, fairly and with respect.

To promote equal access to the benefits of IT. My work uses all accessible, open source software that is easy to install with the provided environment files. My work will be public upon completion of my course.

9.6.2 Professional Competence and Integrity

Only undertake work that is within your professional competence, do not claim competence that you do not possess. As a Computer Science student that has studied AI and deep learning modules I have competent ability to make a neural network of this type in a research context. I have only covered content that I have studied either in or out of University and have provided a clear explanation and understanding of any concepts that I have not built myself (e.g. arithmetic coding).

Develop you knowledge on a continuing basis. I have built upon my understanding of deep learning throughout my final year and have implemented these new ideas upon my original work (e.g. quantisation). Work that I have learned about nearer the end of my dissertation has been mentioned in section 10.3 about future prospects.

Ensure knowledge of legislation. Refer to section 9.1 on detail of IP, licencing and correct use of copyrighted images for training and testing.

Respect alternate viewpoints and offer honest criticisms of work. I have written about multiple approaches to image compression including a traditional method and a set of neural network methods. I have evaluated my compression scheme against JPEG and have been honest about my success and my failings and have analysed why my network may perform better and worse in different scenarios. I have been open about my comparisons of a different network against my ones.

9.6.3 Duty to Relevant Authority

Carry out professional responsibilities within the authority's requirements while exercising professional judgement at all times. I have made sure to get permission from my supervisor for my project idea. Where I have used my judgement to change my idea, due to better plans or reduced scope to meet deadlines, I have also made sure to receive agreement from my supervisor.

Accept personal responsibility of your work All work is my own and any work used from outside sources has been credited.

Do not misrepresent or withhold information on the performance of products I have extensively and honestly evaluated the performance of my models, in cases where my model is both better and worse than comparative systems.

9.6.4 Duty to the Profession

Uphold the reputation of the profession All actions taken during the course of this research were done with respect and honesty, following the ethical and legal guidelines that University of Surrey requires.

Seek to improve professional standards In this project I have tried to add to the field of machine learning based compression by making a transformer based auto encoder system. I have followed the standards of compression evaluation and used forefront standard neural architectures.

10 Conclusion

10.1 Overview

This research looks at using a transformer centric approach to image compression with neural networks, using both the Swin and ViT architectures. By developing a modular autoencoder architecture, a neural architecture search method called grid search was enlisted to identify the relationship between different hyper-parameters. Results from the NAS were used to create 3 compression models which target different compression ratios, which also have varying levels of image reconstruction performance. Comparisons against JPEG compression show the neural networks' ability to construct much more naturally shaped features along with better colour accuracy, in high compression ratios, than JPEG.

10.2 Achievements

Enlisting an auto-encoder design along with forefront Computer Vision architectures (Swin, ViT), I made a compression architecture which can easily scale given a set of four different hyper-parameters. Adjusting the hyper-parameters at multiple parameter tuning stages has helped me find three strong networks for different compression ratios by using my gained understanding of how each hyper-parameter effects the final model performance.

By using quantisation and arithmetic coding I was able to further compress the transformations made by the encoder network with little effect on the quality of the decompressed image.

My three final networks target different compression ratios at 2, 1, and 0.5 bits per pixel. My stronger compression networks outcompeted traditional JPEG compression between 0.1-2.8 dB rate distortion, however my weaker compression networks fail to provide a better decompressed image than JPEG, with JPEG producing between 1.2-2.5 dB better image quality than my comparative network. The promising strong compression performance from my network suggests that machine learning based compression may have its place in targetting much lower bits per pixel while JPEG can maintain its dominance in high bit per pixel compression ratios. My networks were not able to achieve the image quality produced by a comparative network but were significantly smaller in parameter count, ranging from 2-4 times smaller.

10.3 Future Prospects

Mentioned in section 8.2, further investigation into a significant increase in parameter count for each model could provide better fitting to the image data. Furthermore training my models on the entire ImageNet dataset with much more epochs could improve the PSNR performance of each network. While I used forefront architectures in this research (Swin, ViT and general autoencoder design) I would like to extend the research to look at variational autoencoders which have been found to deal with unseen images much better than normal autoencoders by learning the posterior distribution of the latent space in which the images are encoded. Finally, as other research papers have done, the development of a hyper-prior network would be a strong next step for extracting lower-level features to aid the reconstruction of the image.

References

[1] K. Arai and S. Kapoor, Eds., *Advances in Computer Vision*. Springer International Publishing, 2020. [Online]. Available: https://doi.org/10.1007%2F978-3-030-17795-9

[2] Google, "Copyright transparency report," 2022. [Online]. Available: https://storage.googleapis.com/transparencyreport/report-downloads/pdf-report-22_2022-1-1_2022-6-30_en_v1.pdf

[3] R. Khosla, "Auto-encoders for computer vision: An endless world of possibilities," 2021. [Online]. Available: https://www.analyticsvidhya.com/blog/2021/01/auto-encoders-for-computer-vision-an-endless-world-of-possibilities/

[4] R. Gray and D. Neuhoff, "Quantization," *IEEE Transactions on Information Theory*, vol. 44, no. 6, pp. 2325–2383, 1998.

[5] I. H. Witten, R. M. Neal, and J. G. Cleary, "Arithmetic coding for data compression," *Commun. ACM*, vol. 30, no. 6, p. 520–540, jun 1987. [Online]. Available: https://doi.org/10.1145/214762.214771

[6] D. A. Kerr, "Chrominance subsampling in digital images," *The Pumpkin*, vol. 2012, no. 3, pp. 1–15, 2012.

[7] C. Mike Pound. Jpeg dct, discrete cosine transform (jpeg pt2). Youtube. [Online]. Available: https://www.youtube.com/watch?v=Q2aEzeMDHMA

[8] S. A. Khayam, "The discrete cosine transform (dct): theory and application," *Michigan State University*, vol. 114, no. 1, p. 31, 2003.

[9] L. Cavigelli, P. Hager, and L. Benini, "Cas-cnn: A deep convolutional neural network for image compression artifact suppression," in *2017 International Joint Conference on Neural Networks (IJCNN)*, 2017, pp. 752–759.

[10] Y. Hu, W. Yang, and J. Liu, "Coarse-to-fine hyper-prior modeling for learned image compression," *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 34, no. 07, pp. 11 013–11 020, Apr. 2020. [Online]. Available: https://ojs.aaai.org/index.php/AAAI/article/view/6736

[11] M. Lu, P. Guo, H. Shi, C. Cao, and Z. Ma, "Transformer-based image compression," 2021. [Online]. Available: https://arxiv.org/abs/2111.06707

[12] Y. Bai, X. Yang, X. Liu, J. Jiang, Y. Wang, X. Ji, and W. Gao, "Towards end-to-end image compression and analysis with transformers," 2021. [Online]. Available: https://arxiv.org/abs/2112.09300

[13] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly, J. Uszkoreit, and N. Houlsby, "An image is worth 16x16 words: Transformers for image recognition at scale," *CoRR*, vol. abs/2010.11929, 2020. [Online]. Available: https://arxiv.org/abs/2010.11929

[14] Y. Qian, Z. Tan, X. Sun, M. Lin, D. Li, Z. Sun, H. Li, and R. Jin, "Learning accurate entropy model with global reference for image compression," 2022.

[15] S. Velliangiri, S. Alagumuthukrishnan, and S. I. Thankumar joseph, "A review of dimensionality reduction techniques for efficient computation," *Procedia Computer Science*, vol. 165, pp. 104–111, 2019, 2nd International Conference on Recent Trends in Advanced Computing ICRTAC -DISRUP - TIV INNOVATION , 2019 November 11-12, 2019. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1877050920300879

[16] M. Nagel, M. Fournarakis, R. A. Amjad, Y. Bondarenko, M. van Baalen, and T. Blankevoort, "A white paper on neural network quantization," 2021.

[17] A. Singh and R. Gilhotra, "Data security using private key encryption system based on arithmetic coding," *International Journal of Network Security & Its Applications (IJNSA)*, vol. 3, no. 3, pp. 58–67, 2011.

[18] S. Jamil, M. J. Piran, and O.-J. Kwon, "A comprehensive survey of transformers for computer vision," 2022.

[19] Z. Chao, W. Wei-zhi, Z. Chen, F. Bin, W. Jian-guo, F. Gu, and Y. Xue, "Extraction of local and global features by a convolutional neural network–long short-term memory network for diagnosing bearing faults," *Proceedings of the Institution of Mechanical Engineers, Part C: Journal of Mechanical Engineering Science*, vol. 236, no. 3, pp. 1877–1887, 2022. [Online]. Available: https://doi.org/10.1177/09544062211016505

[20] D. Bank, N. Koenigstein, and R. Giryes, "Autoencoders," 2021.

[21] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," 2017.

[22] H. Touvron, M. Cord, M. Douze, F. Massa, A. Sablayrolles, and H. Jégou, "Training data-efficient image transformers & distillation through attention," 2021.

[23] S. Khan, M. Naseer, M. Hayat, S. W. Zamir, F. S. Khan, and M. Shah, "Transformers in vision: A survey," *ACM Comput. Surv.*, vol. 54, no. 10s, sep 2022. [Online]. Available: https://doi.org/10.1145/3505244

[24] Z. Liu, Y. Lin, Y. Cao, H. Hu, Y. Wei, Z. Zhang, S. Lin, and B. Guo, "Swin transformer: Hierarchical vision transformer using shifted windows," *CoRR*, vol. abs/2103.14030, 2021. [Online]. Available: https://arxiv.org/abs/2103.14030

[25] Y. Li, K. Zhang, J. Cao, R. Timofte, and L. V. Gool, "Localvit: Bringing locality to vision transformers," 2021.

[26] R. Zou, C. Song, and Z. Zhang, "The devil is in the details: Window-based attention for image compression," 2022.

[27] R. Gong, X. Liu, S. Jiang, T. Li, P. Hu, J. Lin, F. Yu, and J. Yan, "Differentiable soft quantization: Bridging full-precision and low-bit neural networks," 2019.

[28] G. G. Langdon, "An introduction to arithmetic coding," *IBM Journal of Research and Development*, vol. 28, no. 2, pp. 135–149, 1984.

[29] L. Theis, "Range coder," 2020. [Online]. Available: https://github.com/lucastheis/rangecoder

[30] Z. M. Chng, "Using normalization layers to improve deep learning models," 2019. [Online]. Available: https://machinelearningmastery.com/using-normalization-layers-to-improve-deep-learning-models/

[31] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," 2017.

[32] R. Krishnamoorthi, "Quantizing deep convolutional networks for efficient inference: A whitepaper," 2018.

[33] "Exceptions to copyright," 2021. [Online]. Available: https://www.gov.uk/guidance/exceptions-to-copyright

[34] "Data protection act," 2018. [Online]. Available: https://www.legislation.gov.uk/ukpga/2018/12/contents/enacted