University of BRISTOL

Learned Image Compression with Transformers

Charlie Tan

April 29, 2022

Final year project thesis submitted in support of the degree of Bachelor of Engineering in Electrical & Electronic Engineering

Department of Electrical & Electronic Engineering

University of Bristol

Page left blank

DECLARATION AND DISCLAIMER

Unless otherwise acknowledged, the content of this thesis is the original work of the author. None of the work in this thesis has been submitted by the author in support of an application for another degree or qualification at this or any other university or institute of learning.

The views in this document are those of the author and do not in any way represent those of the University.

The author confirms that the printed copy and electronic version of this thesis are identical.

Signed: 

Dated: April 29, 2022

Abstract

Image compression is used to reduce the size of images, making them more suitable for communication and storage. Due to statistical redundancy, moderate compression can be achieved losslessly. However, heavy compression implies losing information and hence incurring distortion in the decoded image. The challenge of lossy compression is selecting which information to retain such that distortion is minimised. Machine learning has been successfully and extensively applied to image coding, to both enhance conventional methods and form end-to-end learned codecs. Transformers are a type of machine learning model with state-of-the-art performance in a variety of task domains, but with limited work applying them to image processing.

This project applies a novel transformer-based architecture to the task of learned image compression; it further proposes a reformulation of the existing learned compression framework. The proposed framework imposes constraints on both dimensionality and symbol entropy, avoiding the need to entropy code entirely. The proposed framework and architecture is evaluated on the IEEE-CVPR Challenge on Learned Image Compression 2022 Dataset, demonstrating the codec to be both functional and effective. The latent symbols are shown to have entropy of approximately equal to their bit depth, indicating entropy coding would not significantly reduce the rate achieved. As measured by MS-SSIM the proposed framework is demonstrated to exceed the performance of Better Portable Graphics but is found to be inferior when measured by PSNR.

Acknowledgements

I most sincerely thank Dr. Aaron Zhang for his extensive support during this project, and for the opportunities he has provided me to learn and grow academically. I further thank Prof. David Bull for the computing resources he has provided access to, as well as the University of Bristol High Performance Computing Team. I am additionally grateful for the encouragement of Dr. Simon Armour, and the advice and guidance of Duolikun Danier and Chen Feng. I lastly thank my family and friends for their kindness and support.

Contents

Chapter 1

Introduction

This chapter details the motivation, aims, objectives and main contributions of this project. It further outlines the thesis organisation.

1.1 Motivation

In 2014 an estimated 1.8 billion images were uploaded to the Internet every day [1]. Without compression, and assuming 1080×1080 24-bit images, this amounts to in excess of 6 petabytes of data per day. This volume of data would add significant load to constrained communication networks, without even considering data-intensive moving images. Image compression allows images to be represented at a fraction of their original size; this is essential for enabling the storage and communication of image data at scale [2]. Image compression algorithms are often measured by their compression performance; the quality of the decoded images for a given bitrate. Further advances in conventional image compression are becoming increasingly difficult to achieve, leading researchers to investigate novel approaches.

Machine learning refers to algorithms that 'learn' to complete tasks implicitly from data [3]. Machine learning has been applied extensively to image coding, both to enhance conventional codecs and to form end-to-end learned schemes. Learned image compression is an active area of research that has exceeded the performance of conventional compression [4, 5]. Whilst learned codecs currently suffer from high complexity, as both hardware and codec efficiency advance they may become widespread, particularly in cases where ultra-high performance is required. Research in learned image compression therefore carries significant academic, commercial and economic motivation.

Transformers are a family of machine learning model ubiquitous in natural language processing [6]. Research applying transformers to computer vision has also proven highly successful, with state-of-the-art performance in image classification [7], amongst other tasks. However, there is limited work applying transformers to

image coding. Computer vision and image processing share many common themes, making this an exciting area of novel work likely with significant unrealised potential.

1.2 Aims and Objectives

The aim of this work is to develop and evaluate a novel learned image compression framework, based on the transformer neural architecture. It is hoped the evaluation performance will be competitive with the state-of-the-art in conventional and learned image compression. This work will represent an early investigation of the relatively unexplored transformer-based image processing, and it is hoped the methodology and results can inform future work in this area. Towards the project aims, there are several objectives:

1. Implement and debug a training and evaluation workflow using Python and PyTorch.
2. Refine the proposed methodology through prototyping, seeking optimal compression performance.
    - Train neural networks on high performance computing clusters.
    - Interpret validation results.
    - Incorporate features of relevant research.
3. Benchmark the proposed method against existing codecs.
4. Critically evaluate the results relative to existing methods, making suggestions for future work.

1.3 Main Contributions

The main contributions of this work are:

1. Reformulating the learned compression workflow with constrained entropy, removing the need to optimise rate or entropy code.
2. Proposing a novel architecture for learned image compression, denoted the Bottleneck Transformer, that uses a transformer for both the encoding and decoding of images.
3. Proposing a novel encoding algorithm; training iterations during encoding, allowing a learned codec to optimise itself for the image to be encoded.
4. Implementing, training and evaluating the proposed framework, comparing to existing methods.

1.4 Thesis Organisation

The remainder of this thesis is organised as follows:

- Chapter 2 details the underlying theory behind image compression and machine learning, as well as relevant work on which the proposed method is based.
- Chapter 3 defines the proposed compression framework and neural architecture. It further includes the training methodology utilised.
- Chapter 4 reports training results, details the evaluation procedure and presents evaluation results with distortion metrics and image samples.
- Chapter 5 builds on the previous to discuss and interpret the results in relation to the proposed methodology.
- Chapter 6 draws conclusions from the work, recognises limitations and proposes suggestions for future work.

Chapter 2

Background

The background of this project is provided in this chapter. Section 2.1 defines the objectives of image compression, methods for evaluating different codecs as well as conventional approaches to compression. Section 2.2 builds from early machine learning to modern methods such as transformers, giving a comprehensive overview of modern deep learning techniques. Lastly, section 2.3 details how machine learning has been applied to image processing, with a particular focus on compression.

2.1 Image Compression

The objective of image compression is to communicate images at a reduced bitrate whilst incurring minimal distortion. Bitrate (or just rate) is the number of bits in the bitstream, often measured relative to the spatial dimensions of the image as bits-per-pixel (bpp). An uncompressed 24-bit RGB image has 24 bits for each pixel (an 8-bit integer for each of red, green and blue). The ratio of bits in the original image to those in the bitstream gives the compression ratio. Distortion is the difference, as measured by a given metric, between the original and decoded image. This work will focus on *perceptual* quality: the visual similarity between the original and decoded images as visible to a human observer.

2.1.1 Lossless and Lossy Compression

Image compression algorithms can be divided into two categories: lossless and lossy [2]. Lossless compression exploits only the statistical redundancy within images; the decoded images contain no distortion, but low rates are not achievable. In contrast lossy compression also exploits the visual redundancy within images; information with little visual significance is discarded to greatly reduce the rate. The challenge of lossy compression is the tension between rate minimisation and distortion minimisation, known as the rate-distortion trade-off. A high rate can represent an image with less distortion, whereas a low rate will require losing more information and

subsequently produce greater distortion. High-rate lossy compression can be imperceptibly similar to the original image, at a significantly lower rate than lossless compression. Lossy compression algorithms often include entropy coding, a form of lossless compression. Lossy image compression is the focus of this work, and references to 'image compression' or 'image coding' hereafter can be assumed to refer to this.

2.1.2 Image Coding Standards

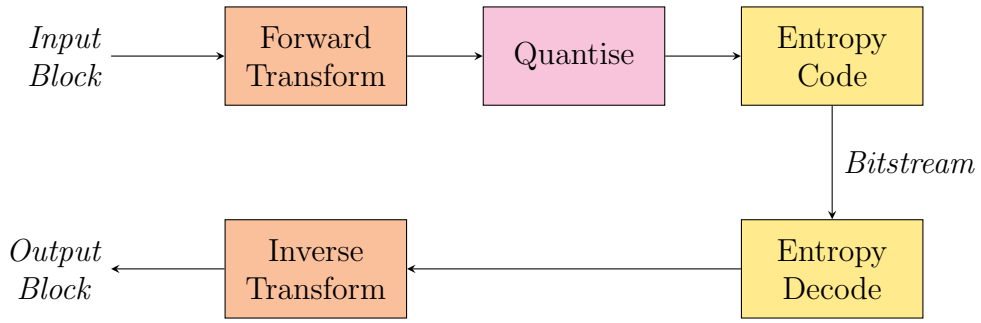


Figure 2.1: Transform coding framework.

JPEG is an image coding standard released in 1992 by the Joint Pictures Expert Group [8], and remains the most commonly used form of image compression [9]. JPEG employs the transform coding workflow for image compression, outlined in Figure 2.1. Encoding begins by splitting the image into blocks, to which the discrete cosine transform (DCT) is applied [10]. The DCT is chosen over the discrete Fourier transform (DFT) due to the DFT having complex coefficients and thus requiring more data to represent, as well as the DFT being more susceptible to blocking artefacts. Next, the DCT coefficients are quantised by an elementwise product with a quantisation matrix. The quantised coefficients are then entropy coded with run-length encoding and Huffman coding [11] or Arithmetic coding [12] to produce the bitstream. Decoding proceeds by entropy decoding the bitstream to recover the quantised coefficients, performing the inverse transformation and finally reconstructing the image from the decoded blocks.

The principal source of distortion in the transform coding workflow is the quantisation operation; the extent of quantisation is used to define the compression ratio. The purpose of the transformation is to decorrelate the image block and produce a representation that approximately separates information by visual significance [2]. In the case of the DCT the image block is transformed into the discrete frequency domain; each coefficient corresponds to a sinusoid of different frequency [8]. The first element is the DC component, with the vertical and horizontal frequency increasing along each axis. The quantisation matrices are defined such that the higher frequency components are reduced to 0 whilst the DC and low frequency components

are retained. These matrices are experimentally verified to align with subjective opinion. After decoding, the image will have reduced high-frequency information.

Numerous image coding standards have been released since JPEG. JPEG2000 was the successor to JPEG, released by the Joint Pictures Expert Group in 2000 [13]. JPEG2000 exceeded the performance of JPEG by 10-20%, with superiority increasing with decreasing bitrate. A major contribution of JPEG2000 is the use of discrete wavelet transforms (DWT) instead of the DCT, and the facility to define a region-of-interest within the image.

Video compression shares many common features with image compression; image codecs can be used to encode video frame-by-frame. However, this would be far from optimal due to the high temporal redundancy between frames, hence specific video codecs are employed. High-efficiency Video Coding (HEVC) is one such standard, released in 2013 by the ITU-T Video Coding Experts Group [14], achieving up to 36% performance gains over its predecessor Advanced Video Coding. In addition to video, HEVC is also able to compress still images using its intra-coding profile [15]. The HEVC intra-profile employs intra-prediction methods to exploit spatial redundancy within images, before encoding the residuals; it further makes use of variable sized coding blocks. HEVC intra-coding was shown to achieve average bitrate saving of 44% and 23% over JPEG and JPEG2000 respectively [16]. Better Portable Graphics (BPG) is a wrapper for HEVC intra-coding as an image codec. HEVC now has a successor itself, Versatile Video Coding (VVC) with improved video and image coding capabilities.

2.1.3 Image Quality Metrics

Full-reference image quality metrics are functions of both the distorted image under evaluation and the undistorted original image. Pixel-wise metrics measure distortion between the images pixel-by-pixel. Examples include mean absolute error (MAE) and mean squared error (MSE). Peak signal-to-noise ratio (PSNR) is derived from MSE and is the most commonly used quality metric. MAE, MSE and PSNR are shown in equations (2.1), (2.2) and (2.3). In equation (2.3) MAX refers to the largest possible integer value in the image format (255 for the unsigned 8-bit integers of 24-bit RGB). One reason PSNR is preferred over MSE is the logarithmic scale values are easier to interpret. MSE and PSNR have been shown to correlate poorly with subjective opinion [17]. One explanation for this is the insensitivity to small errors: values of error less than 1 become smaller when squared and values of error greater than 1 become larger and consequently have a greater effect on the mean. MAE does not share this limitation and is considered a superior image quality metric to MSE / PSNR [18], despite being less commonly used.

$$\text{MAE}(x, y) = \frac{1}{n}\sum_{i=0}^{n}|x_i - y_i| \tag{2.1}$$

$$\text{MSE}(x, y) = \frac{1}{n}\sum_{i=0}^{n}(x_i - y_i)^2 \tag{2.2}$$

$$\text{PSNR}(x, y) = 20\log_{10}\left(\frac{MAX}{\text{MSE}(x, y)}\right) \tag{2.3}$$

Owing to the poor correlation between MSE and subjective opinion, alternative quality metrics have been developed. Perceptual metrics consider more than pixel-wise errors and aim to more closely correspond with subjective opinion. Structural similarity index metric (SSIM) [17] operates on the assumption that the human visual system is highly sensitive to variation in structure, and aims to quantify the structural variation between the original and distorted image. SSIM applies an 11×11 circular sliding window to the original and distorted images; at each window position the similarities in luminance, contrast and structure are computed as per equations (2.4 - 2.6). Here μ_x, σ_x and σ_{xy} are the mean of x, variance of x and covariance of x and y. The mean, variance and covariance are employed as estimations of the luminance, contrast and structural similarity respectively. The constants defined in equation (2.7) are included for numerical stability. The local similarity is computed as the product of these local statistics (2.8), the mean across all positions gives the final metric.

$$l(x, y) = \frac{2\mu_x\mu_y + C_1}{\mu_x^2 + \mu_y^2 + C_1} \tag{2.4}$$

$$c(x, y) = \frac{2\sigma_x\sigma_y + C_2}{\sigma_x^2 + \sigma_y^2 + C_2} \tag{2.5}$$

$$s(x, y) = \frac{\sigma_{xy} + C_3}{\sigma_x\sigma_y + C_3} \tag{2.6}$$

$$C_1 = (MAX \cdot 0.01)^2 \quad C_2 = (MAX \cdot 0.03)^2 \quad C_3 = C_2/2 \tag{2.7}$$

$$\text{SSIM}(x, y) = [l(x, y)] \cdot [c(x, y)] \cdot [s(x, y)] \tag{2.8}$$

SSIM was later extended into multi-scale structural similarity index metric (MS-SSIM) [19]. SSIM was limited in operating at the original image scale, MS-SSIM used

multiple scales to quantify structural similarity more generally. This contribution was made on the observation that the perception of image details is dependent on both viewing distance and sampling density. MS-SSIM iteratively low-pass filters and downsamples the images by a ratio of 2. At each of these scales, similarity in contrast and structure is measured. At the final (smallest) scale M, luminance similarity is also measured. Each scale is weighted by a constant α_j. The local similarity for MS-SSIM is shown in equation (2.9), as in SSIM the final metric is the mean across the spatial positions.

$$\text{MS-SSIM}(x, y) = [l_M(x, y)]^{\alpha_M} \cdot \prod_{j=1}^{M} [c_j(x, y)]^{\alpha_j} \cdot [s_j(x, y)]^{\alpha_j} \tag{2.9}$$

2.2 Machine Learning

Machine learning (ML) is the study of algorithms that 'learn' to complete tasks implicitly through exposure to data [3]. This contrasts with non-learning algorithms whose behaviour must be explicitly programmed. Machine learning is often considered as three major paradigms; supervised, unsupervised and reinforcement. In supervised learning, the input data is provided with labelled target outputs; the objective is to learn a mapping from input to output. Examples include classification and regression. In contrast, unsupervised learning algorithms identify patterns in unlabelled input data. Clustering and dimensionality reduction are classic examples of unsupervised learning. Lastly, in reinforcement learning an agent takes actions within an environment. The agent is 'rewarded' for completing desirable actions, and learns a policy to seek maximal reward.

2.2.1 From Perceptron to Deep Learning

The perceptron algorithm is a supervised learning algorithm for binary classification [20]. The classifier is a vector of learnable weights $\boldsymbol{\theta}$, the input is a vector of equal length $\boldsymbol{x}$, and the labels are scalars $y \in \{-1, 1\}$. The learnable weights represent the normal to a linear hyperplane through the vector space, this hyperplane classifies outputs $\hat{y}$ into two sets. A diagram of this configuration is shown in Figure 2.2. Classification is performed by taking the vector product of the weights with the input before applying a threshold function, as in equation (2.10). Training proceeds by iterating through the dataset. For each data-point, if $\hat{y} \neq y$ misclassification has occurred and the weights are updated with equation (2.11). If the data-points are linearly separable the perceptron algorithm will converge to correctly classify the dataset. A perceptron, without the use of the later developed kernel-trick, is unable to correctly classify non-linearly separable data [21].

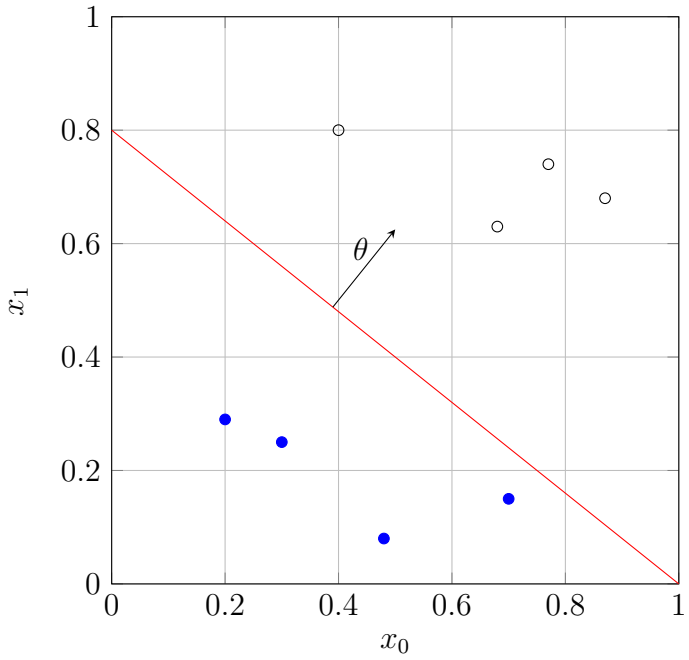


Figure 2.2: Perceptron decision boundary (figure generated artificially).

$$\hat{y} = \begin{cases} 1 & \boldsymbol{\theta}^T \boldsymbol{x} > 0 \\ -1 & \text{otherwise} \end{cases} \tag{2.10}$$

$$\boldsymbol{\theta} \coloneqq \boldsymbol{\theta} + y\boldsymbol{x} \tag{2.11}$$

The multi-layer perceptron (MLP) generalises the perceptron to sequential layers of learnable weights, as illustrated in Figure 2.3. However, the lack of direct connection between the intermediate layers and the output means it is not possible to use the perceptron learning algorithm. The backpropagation algorithm overcomes this incompatibility, and was first applied to MLPs in [22]. The insight of backpropagation was that the derivative chain rule can be used to compute the gradient with respect to the output for all layers in the network. With the gradients computed, it is then possible to minimise a given cost function using an optimisation algorithm such as gradient descent. A further contribution of [22] was the introduction of non-linearities to the network, applied after each layer of weights. This crucially permits the MLP to learn non-linear mappings between input and output. Without the inclusion of non-linearities an MLP with an arbitrary number of layers can be reduced to a single linear operation.

MLPs are an early example of what would eventually become known as deep learning; machine learning with deep neural networks. The progress and popularity of neural networks stagnated somewhat until the early-mid 2010s, when computational power had increased dramatically, and it was feasible to train large models using large datasets. Deep learning has achieved state-of-the-art performance in a

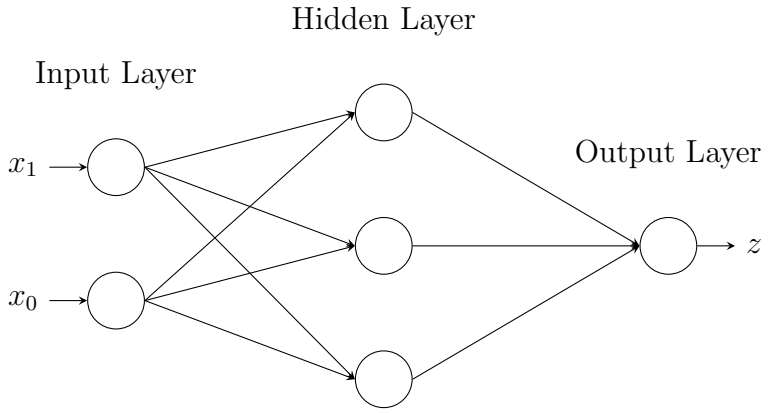


Figure 2.3: Multi-layer perceptron containing three layers.

diverse range of tasks such as object detection, document translation, and game playing. Interestingly despite its ubiquitous success, deep learning is not theoretically well-grounded. In [23], it was proven that an MLP with a single hidden layer of arbitrary width could approximate any continuous function. Originally specific to the logistic non-linearity, this result was later generalised to any non-polynomial function (with suitable gradients) in [24]. More recent work proves a similar result for the case of bounded-width and arbitrary depth [25]. In practice the arbitrary-sizing conditions of these universal approximation theorems are violated, and the strong performance of deep learning remains an active area of research.

2.2.2 Stochastic Optimisation

Stochastic gradient descent (SGD) [26] and its variations are (by far) the most common optimisation algorithms used to train neural networks. In SGD a random subset of the training data is observed at each iteration. The subset is presented to the network as input, and a loss function (the function to be minimised) is applied to the resulting output. The backpropagation algorithm is used to compute gradients for all learnable parameters. SGD then updates each parameter by a scalar value (the learning rate) in the direction of steepest negative gradient. This process repeats for a given number of iterations or epochs (a complete pass through the training data). Importantly, when updating the weights the gradient is not the true gradient but a stochastic approximation of it. Computing the true gradient would require observing the full dataset at once, and require significantly more computing power and memory.

Interestingly, in most cases SGD (with its stochastic gradient approximation) is often a better choice of optimisation strategy than true gradient descent [27]. More specifically, this is true where the loss function is non-convex and thus has local minima. True gradient descent will always step perfectly in the direction of steepest

negative gradient, for a convex function this is ideal, but in the non-convex case this makes it susceptible to converging to poor minima. SGD on the other hand takes noisier steps and is subsequently more likely to converge to a better minima. For this reason SGD is described as having implicit regularising properties.

A number of modifications of SGD have been developed, of which Adaptive Moment Estimation (ADAM) [28] is most relevant. ADAM is an adaptive optimiser: the learning rate is adapted per-parameter during training. This is achieved by maintaining estimates for the first and second order moments of the gradients for each parameter.

During training, it is typical to validate the generalisation performance at regular intervals. This is done by evaluating the model on data that the model has not been trained on. Ultimately the goal of machine learning is to have highly generalisable performance, not just perform well on data that has already been observed. As training begins the training loss and validation loss will both decrease. However, it is possible that at some stage the validation loss will begin to increase again despite the training loss continuing to decrease. This is known as overfitting, the network has learned the noise of the training data and cannot perform well on other data. Several factors influence the presence or absence of overfitting in a training experiment, but models with large numbers of parameters are known to be susceptible. Regularisation refers to strategies for the mitigation of overfitting. Dropout [29] and weight decay [30] are two forms of regularisation. Dropout randomly zeros activations during training. The probability of any given activation being zeroed is a Bernoulli distribution with parameter p. At each training iteration, each activation has a probability of p of being zeroed, and a probability of $1 - p$ of being retained. Weight decay applies a penalty to the l_2 norm of the network parameters $||\theta||$ This penalty is applied at each training iteration and leads to smaller parameters. This penalty means the l_2 norm of the parameters is subject to minimisation leading to lower complexity models. A modification of ADAM to improve its performance with weight decay (ADAMw) was introduced in [31].

2.2.3 Activation Functions

The non-linear functions applied after network layers are referred to as activation functions. In [22], the logistic function was used. The logistic function and hyperbolic tangent (Tanh) function are both examples of sigmoid functions. Despite their similarities, Tanh is considered to lead to faster convergence than the logistic function [32]. This is due to the symmetry about the origin of Tanh, whereas the logistic function is symmetric about 0.5. As discussed in [32], activations will ideally have zero mean and unit variance, making Tanh superior to the logistic function.

The vanishing gradient problem [33] is the phenomenon where backpropagated

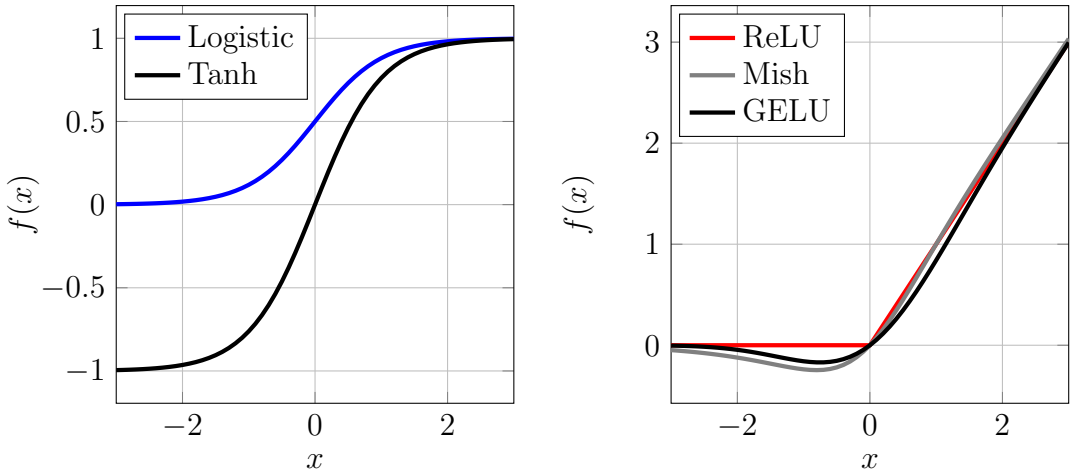


Figure 2.4: Activation Functions.

gradients become small in magnitude due to the depth of the network. At best, this leads to slow convergence due to the updates being proportional to the gradient magnitudes. At worse, the layers in the early stages of the network receive no useful gradient information and are unable to learn at all. Both the logistic and hyperbolic tangent functions are prone to cause vanishing gradients due to their gradients being in the range [0, 1]. The rectified linear unit (ReLU) activation function [34], was found to be much less susceptible to vanishing gradients [35]. Further modifications to the ReLU function have been proposed, such as Mish [36] and Gaussian Error Linear Unit GELU [37]. Mish and GELU both have a 'bump' in the negative region, and are both special cases of the same function. Mish is often used in computer vision research, whereas GELU is more typical of natural language processing.

2.2.4 Convolutional Neural Networks

An MLP contains dense connections between layers, each neuron is connected to all neurons in the preceding and following layers. Accordingly, one name for MLP layers is fully connected layers. Each of the inter-layer connections corresponds to a parameter defining the weight of the connection, this is a problem for high-dimensionality input such as images. An image of just 128×128 has a dimensionality of 16384; two fully connected layers of this size will have over 260 million connections between them - each with an associated learnable parameter. This large number of parameters makes MLPs susceptible to overfitting, and is also computationally expensive. Convolutional neural networks (CNN) replace fully connected layers with convolutional layers [38, 39]. Convolutional layers apply kernels of learnable weights to input through the convolution operation. This means the same parameters are applied across different input positions, regularising the model through reduced complexity as well as being significantly more computationally efficient. Applying

the same parameters across the entire image / feature map gives CNNs the property of translation equivariance; a circular shift of the input results in the same shift of the output. In contrast an MLPs neurons are fixed in position; different weights will be applied to different parts of the image / feature maps resulting in different output. Convolutional neural networks are now ubiquitous within computer vision and image processing.

2.2.5 Recurrent Neural Networks

MLPs belong to the family of feed-forward neural networks; their outputs are a function of only the current input. The lack of dependence on past outputs implies MLPs are finite impulse response systems. This presents challenges for processing sequential information such as natural language or time-series data. Recurrent neural networks (RNN) [40] are another family of neural network wherein the outputs depend on both the current input and the previous outputs. Recurrent neural networks are able to process sequences of arbitrary length by maintaining a hidden state. At each iteration of the sequence, the hidden state is updated as a function of the input and the previous hidden state. This dependence of current input and previous outputs makes RNNs infinite impulse response systems [41]. Due to their exponential dependency on the parameter weights, the gradients of RNNs tend to 'explode' to infinity or 'vanish' to 0 as they are propagated backwards through the sequence [33]. This can lead training to be unstable or prohibitively slow, particularly for long sequences.

Many variations of RNNs have been proposed, of which two are particularly relevant to this work. Long Short-term Memory [42] was proposed to mitigate the exploding / vanishing gradient problems. This permitted much longer sequences to be processed, and LSTMs found significant success in natural language processing. LSTMs were then extended with learnable attention mechanisms for the task of machine translation [43]. Attention within machine learning refers to mechanisms used to weight the relative importance of information, akin to the attention of biological systems. The key contribution of attention in [43] was to overcome the fixed dimension of the hidden state passed from encoder to decoder, allowing sequences of longer length to be accurately translated.

2.2.6 Transformers

Despite their success, RNNs and their variations are severely limited in throughput by their sequential processing; this makes training slow, and discourages the use of large datasets. Transformers were introduced for the machine translation task [6], and rapidly superseded RNNs as the state-of-the-art in natural-language-processing.

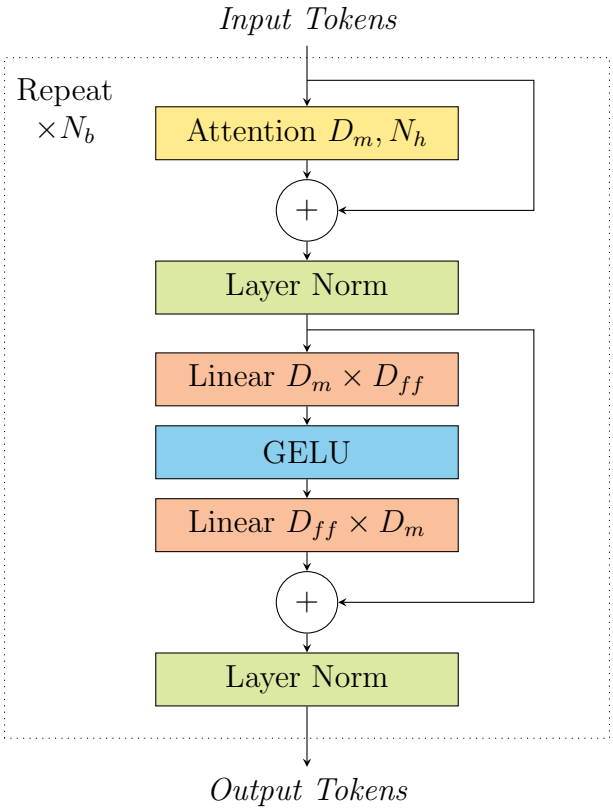


Figure 2.5: Transformer architecture. N_b = number of blocks, D_m = model dimension, N_h = number of attention heads and D_{ff} = feed-forward dimension. Linear layer parameters are denoted as $D_{in} \times D_{out}$.

A diagram of a transformer is provided in Figure 2.5. The main contribution of the transformer was the use of attention mechanisms without recurrence. This overcame the sequential processing limitation of RNNs allowing a significant increase in parallelisation and throughput, permitting larger models to be trained using larger datasets. The transformer specifically used the scaled-dot product attention mechanism given in equation (2.12). In this equation Q, K, V are the outputs of three learnable linear transformations applied to input X, and d_K is the row length of K.

$$Z = \text{softmax}\left(\frac{QK^T}{\sqrt{d_K}}\right)V \tag{2.12}$$

Transformers are typically pre-trained for a general language task using a large dataset and then fine-tuned for a specific task with a smaller dataset. Bi-directional encoder representations from transformers (BERT) was a methodology for pre-training that achieved excellent results [44]. One drawback to transformers is the complexity of the attention mechanism; for sequence length N, the time and memory complexity is $\mathcal{O}(N^2)$. The linear transformer used a kernelisation of the attention mechanism to reduce both time and memory complexity to $\mathcal{O}(N)$ [45].

Owing to their widespread success in NLP, there is research interest applying

transformers to other domains. Notably, [7] applied transformers to object detection. This work was the first to apply transformers to a computer vision task without the inclusion of convolutional layers, essentially applying the same architecture as [44] to images. An important contribution of this work was the patch embedding methodology. Transformers operate on sequences of tokens whereas images are two dimensional. In order to process an image with a transformer it is necessary to produce tokens from the image. The methodology used was to split the image into non-overlapping patches before flattening into a sequence of vectors. The tokens are then linearly transformed to produce tokens. As in [6] a learnable additive positional encoding is applied to provide the transformer with positional information. Further work has applied transformers to auto-regressive image generation [46] and image completion [47].

2.3 Machine Learning-based Image Processing

Machine learning has been applied extensively to image processing. Image processing tasks include denoising, super-resolution and learned image compression.

2.3.1 Loss Functions

All of MAE, MSE / PSNR, SSIM and MS-SSIM are differentiable operations, they can therefore all be used as loss functions for neural network optimisation. It is noteworthy that MSE is used in extensive applications in statistics and optimisation outside of image processing. MAE and MSE are much simpler operations and less computationally expensive. Comparing the two, MAE is preferred in image processing due to its piecewise constant gradient [18]. In contrast, the gradient of MSE tends to zero as the error tends to zero; this leads to blurry images due to the small gradients limiting the learning process. SSIM and MS-SSIM are more computationally intensive. Some image processing works combine metrics to propose perceptual loss functions. One such example is [48] who propose the loss function given in equation (2.13), derived using a cross-validation method.

$$\begin{aligned} \mathrm{PLF}(x,y) = {} & 0.3 \cdot \ln[\mathrm{MAE}(x,y)] + 0.1 \cdot \ln[\mathrm{MSE}(x,y)] \\ & + 0.2 \cdot \ln[\mathrm{SSIM}(x,y)] + 0.4 \cdot \ln[\text{MS- SSIM}(x,y)] \end{aligned} \tag{2.13}$$

2.3.2 Learned Image Compression

The first framework in which the transform coding framework was learned end-to-end was introduced in [4]. The network architecture was a convolutional autoen-

coder, with the encoder and decoder tasked with learning the analysis and synthesis functions respectively. Ideally the objective function would be as stated in equation (2.14), where R is the rate, D the distortion and λ is a Lagrange multiplier used to weight the relative importance of the two. However, there are two obstacles to optimising this in the transform coding framework. Firstly, the scalar quantisation operation (applied to give integer values for entropy coding) has a gradient of zero almost everywhere which would prevent the encoder from learning. Secondly, entropy coding nor the computation of discrete entropy are differentiable operations preventing the rate from being optimised. The work of [4] overcame these challenges with a form of quantisation-aware training, wherein the scalar quantisation operation was replaced with an additive uniform noise source during training. This simulated the quantisation operation, and allowed the distortion gradients to propagate backwards from decoder to encoder. Furthermore, the differentiable entropy of the latent vector + noise can be used as an approximation of the discrete entropy of the quantised symbols, thus giving a differentiable value for rate. This method achieved superior performance to JPEG and JPEG2000 as measured by PSNR and MS-SSIM.

$$L = R + \lambda D \tag{2.14}$$

One limitation of [4] was the lack of entropy coding; the entropy was reported as an estimation of the rate. In [5] the same authors extend their previous work with arithmetic coding and a hyperprior. Arithmetic coding relies on knowledge of symbol probability distribution. However, the coded symbols produced by each image are not identically distributed. The hyperprior is communicated as side information alongside the image code symbols such that the arithmetic coder can achieve superior lossless compression. This method achieved marginally inferior performance to BPG, and significantly superior to JPEG and JPEG2000.

Chapter 3

Method

This chapter concerns the project methodology. Section 3.1 defines the proposed compression framework relative to existing methods. Section 3.2 details the novel neural architecture employed within the framework. Section 3.3 reports the training configuration and dataset used. Lastly, implementation details are included in section 3.4.

3.1 Compression Framework

3.1.1 Overview

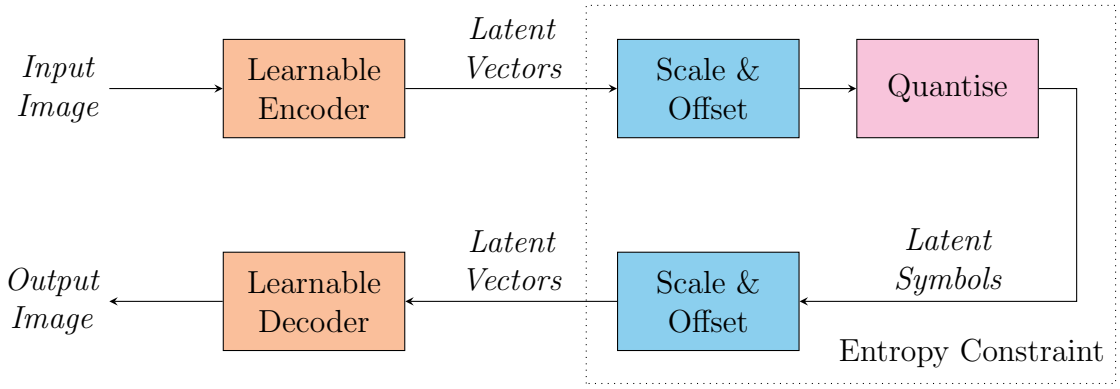


Figure 3.1: Proposed compression framework.

The proposed compression framework is outlined in Figure 3.1. This framework is a variation of that introduced in [4]; an important distinction lies in the use of entropy coding. In [4] the use of entropy coding was assumed, and the discrete entropy of the latent symbols was employed as an estimation of the rate. A continuous relaxation of the discrete entropy was then included in the objective function and subject to minimisation, permitting the learning of a lower 'rate'. In contrast, the proposed framework forgoes the use of entropy coding in favour of imposing an upper-bound on the discrete entropy. This is achieved by constraining the quantised symbols S to

a range of integers as in equation (3.1), where M is a scalar parameter. The entropy upper-bound H_{max} can then be derived as per equation (3.2). The upper-bound will be realised in the case that the symbols are uniformly distributed, as shown in equation (3.3). Under this formulation, the rate no longer requires optimisation and the objective is simply to minimise distortion subject to this constraint.

$$\{S \in \mathbb{Z} : 0 \leq S < M\} \tag{3.1}$$

$$H_{max} = \log_2 M \tag{3.2}$$

$$P_S \sim \text{unif}(0, M-1) \longrightarrow H = H_{max} \tag{3.3}$$

To implement the entropy constraint, the latent vectors (which are bounded to range [-1, 1]) are scaled and offset to the range [0, M - 1], before quantisation to the set given in equation (3.1). Larger values of M will result in quantisation to a wider range of values and a greater H_{max}. Before decoding the values are scaled and offset back to [-1 1]. As in [4], additive uniform noise was used to simulate the quantisation during training. The noise is independent of M with range [-0.5, 0.5]. The influence of noise therefore decreases with increases to M, corresponding to the fineness of quantisation increasing with M.

3.1.2 Compression Ratios

As in [4], learned models are trained for each compression ratio. However, the method for achieving different compression ratios differs in this work. In [4] the learned encoder applies a fixed dimensionality reduction of 1/64 for all models. Different compression ratios are achieved by modifying the λ hyperparameter to adjust the relative weighting between rate and distortion. In contrast, in this work the compression ratio is controlled by two hyperparameters: the aforementioned M (controlling the symbol entropy) and the dimension of the symbol space D_S (controlling the number of symbols). For this thesis, M has been fixed to 8 to give to $H_{max} = 3$ bits, a decision discussed further in subsection 5.3. Compression ratio is therefore controlled using D_b, where smaller values lead to more aggressive dimensionality reduction and higher compression ratios.

3.1.3 TIDE

A further contribution of this work is the use of training iterations during encoding (TIDE). TIDE is a proposed method for learned image encoding where the compression framework is fine-tuned end-to-end before encoding each image. This is executed for a limited number of iterations using the image to be encoded as training material. Crucially, the decoder is 'frozen' such that its parameters are not

updated; only the encoder is permitted to learn. The decoder cannot be modified during encoding as this would render the bitstream incompatible with the original decoder. However, using TIDE the encoder is able to adapt to the specific image to be encoded and produce a superior compressed representation whilst maintaining compatibility. The learned parameters are reset to the saved values for each image, discarding the fine-tuned parameters after encoding. This algorithm can be understood as overfitting the encoder on the image to be encoded before producing the latent symbols. The number of TIDE iterations was set to 100 for this work to limit the computational cost of encoding. The complexity of decoding is unaffected by TIDE.

3.2 Network Architecture

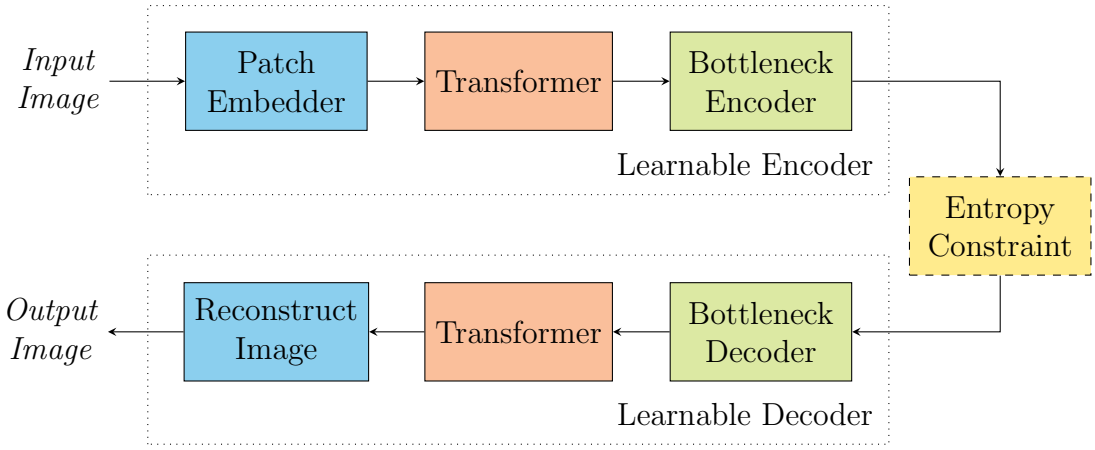


Figure 3.2: Learnable encoder and decoder.

Together the learned encoder and decoder make up an autoencoder. A novel transformer-based architecture, denoted the Bottleneck Transformer (BT) was developed for this work. A flow diagram of this architecture is presented in Figure 3.2. As transformers operate on one-dimensional tokens the image is patch embedded before processing, using a process similar to [7]. The output of the encoder transformer is then subjected to an information bottleneck. As discussed, the bottleneck is constrained in both dimensionality and symbol entropy. Constrained dimensionality is achieved with a learnable linear projection into a lower-dimension space. At the decoder the symbols are projected back to the transformer model dimension and processed by a second transformer. An image is then reconstructed from the transformer output. This image is then either post-processed using a simple CNN, or immediately output.

3.2.1 Transformers

Both the encoder and decoder contain a stack of transformer layers, which are illustrated in Figure 2.5. The transformer layers are structurally the same in both parts of the codec (unlike in [6] where the decoders layer contain encoder-decoder attention). To reduce the complexity of the transformer, kernelised linear attention [49] is used instead of the original attention formulation. Key hyperparameters defining a transformer model are; the number of sequential transformer blocks N_b, the model dimension D_m, the number of heads in each multi-headed attention layer N_h and the dimension of the feed-forward neural network sub-blocks D_{ff}. For this work $N_b = 4$ (for each of encoder and decoder), $D_m = 1024$, $N_h = 16$ and $D_{ff} = 2048$. Without positional encoding, transformers can process arbitrary sequence lengths. Despite the linear attention significantly reducing the memory complexity for large sequence lengths, it was found to be necessary to train on images blocks (as opposed to full images).

3.2.2 Patch Embedder

Figure 3.3: Patch embedding. Linear layer parameters are denoted as $D_{in} \times D_{out}$.

Transformers operate on sequences of one-dimensional tokens, to process images tokens must be produced from the image data. The procedure for this is detailed in Figure 3.3. Let C, H and W refer to the channels, height and width of an image block, and P refer to the side length of a square patch size. Following [7], the image ($C \times H \times W$) is first split into non-overlapping patches ($N_t \times C \times P \times P$) using a sliding window. N_t is related to C, H and W through equation (3.4). Next, these patches are flattened into vectors ($N_t \times CP^2$). A learnable linear projection is then applied to produce tokens ($N_t \times D_m$). In this work $P = 18$, and therefore $CP^2 = 864$. D_m was chosen as 1024 to be moderately larger than this, and to align with transformer sizing in existing work. Based on the observation that activation functions are not present between each transformer block, there is no activation function between this linear layer and the first transformer block. The principal difference between this patch embedding and that of [7] is the omission of additive positional encoding applied to the tokens. Preliminary experiments indicated positional encoding did not improve performance, and presented complications for processing arbitrary sized image blocks.

$$N_t = \frac{HW}{P^2} \tag{3.4}$$

3.2.3 Learnable Bottleneck

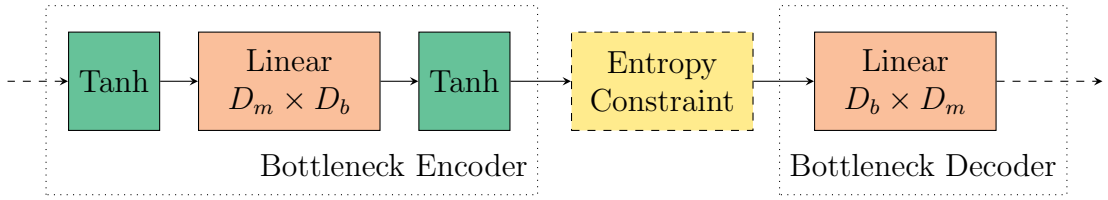


Figure 3.4: Learnable bottleneck. Linear layer parameters are denoted as $D_{in} \times D_{out}$.

As the last learnable stage of the encoder, the bottleneck encoder applies dimensionality reduction to the feature space, as illustrated in Figure 3.4. The tokens $(N_t \times D_m)$ output by the final transformer layer of the encoder are Tanh-activated and linearly projected into a low-dimensional space $(N_t \times D_b)$. The dimension of this bottleneck D_b is the number of elements in each latent vector. Another Tanh activation follows the linear layer, leading to latent vectors bounded to [-1, 1]. The latent vectors are then quantised (or perturbed by noise for training) into latent symbols before being passed to the decoder. On the decoder side of the bottleneck there is a single linear layer that projects from D_b to D_m. As in the patch embedder, there is no activation between this layer and the following transformer stack.

3.2.4 Image Reconstruction

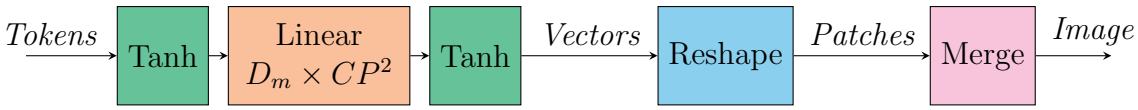


Figure 3.5: Image reconstruction. Linear layer parameters are denoted as $D_{in} \times D_{out}$.

There is limited work applying transformers to image processing and none in which the transformer outputs full images simultaneously; image output is typically achieved pixel-by-pixel auto-regressively [46]. In order to reconstruct the image from the output tokens, the process shown in Figure 3.5 was defined to mirror the patch embedding. The output tokens $(N_t \times D_m)$ are processed by a learnable linear transformation to produce vectors $(N_t \times CP^2)$; this transformation is both preceded and followed by Tanh activation. The penultimate Tanh introduces a non-linearity between the last linear layer in the transformer and final linear layer. As discussed in subsection 2.2.1, sequential linear transformations without non-linear activation functions are redundant and are no more expressive than a single linear layer. Anecdotally, preliminary experiments indicated the greater expressiveness of

including a non-linearity improved performance. The use of the final Tanh activation was inspired by work applying CNNs to image processing, where it is often used as the final stage of processing to give bounded output. After the final Tanh the element values are no longer modified, and the vectors are reshaped into patches ($N_t \times C \times P \times P$) before merging into the reconstructed image block ($C \times H \times W$).

3.2.5 Convolutional Post-processing

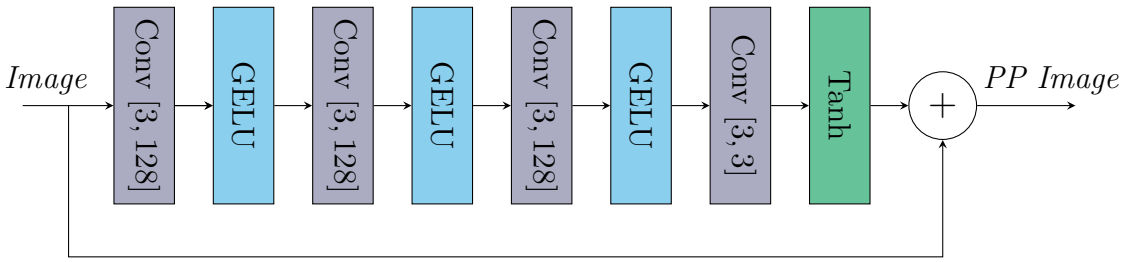


Figure 3.6: Convolutional post-processing. Convolutional layer parameters are denoted by [kernel size, channels]. In all cases stride = 1.

During development visible discontinuities between adjacent patches were observed in the decoded images, particularly at low bitrates. For this reason, the framework was also trained with a convolutional post-processor for artefact suppression. CNNs are well established for post-processing [50]. The simple architecture illustrated in Figure 3.6 was used; all convolutional layers have a kernel size of 3, stride of 1 and 128 channels (except the final which produces the 3 channel RGB output). The use of the GELU activation function [37] is atypical for CNNs, however it is highly similar to the common Mish activation [36] and was hence chosen for consistency with the transformer. As in other post-processing works the output of the final convolutional layer is summed with the input [50]. This means the task of the network is to learn the difference between the input and ideal output, as opposed to approximating the ideal output directly. Convolutional pre-processing was additionally experimented with but was not found to offer significant performance gains with or without a post-processor.

3.3 Training

3.3.1 Data

A subset of the Unsplash Full Dataset 1.2.0 [51] was used as training material. The full dataset contains in excess of 3 million images available in the lossless Portable Network Graphics (PNG) format. Lossless images are strongly preferable to those having undergone lossy compression to avoid distributional differences

between training and evaluation data which could negatively impact generalisation performance. Both the evaluation and validation datasets were also derived from Unsplash and thus will have similar distributions. A training subset of 100,000 images was downloaded. The images were resized using a Lanczos filter such that their smaller edge was within the range [1024, 2048]. This resizing was done because the evaluation images had also been resized to have their longest edge equal to 2048, and it is desirable to have a similar density of features and texture. Any original images smaller than this range were discarded, as were images not in 24-bit RGB format.

3.3.2 Configuration

The training methodology was informed by existing work training transformers [44, 7] as well as CNNs for image processing [50]. The training is conducted in two stages; pre-training and fine-tuning. In both stages, the objective is to minimise distortion in the decoded images. Image blocks of 288 were used for training. The perceptual loss function defined in [50] was used as the distortion measure due to its high performance at balancing perceptual detail. During pre-training, the network is regularised with dropout [29] and weight decay [30]. The regularisation applied leads the network to learn robust parameters and allows the training to proceed at length without over-fitting. For fine-tuning, the network is trained for significantly fewer iterations with neither dropout nor weight decay applied. This two-stage procedure was found to achieve superior performance to a single longer stage with or without regularisation. Both stages of training used ADAMw [31] for optimisation with a batch size of 16. During each training iteration a mini-batch of image blocks is used as input. The blocks are produced for each mini-batch by randomly cropping images from the training dataset, and further augmented with random rotations and flipping. The rotation is restricted to 0, 90, 180 and 270 degrees to avoid interpolation, and the distortion this incurs.

Pre-training was carried out for 500,000 iterations. The initial learning rate was 4×10^{-6} which was increased linearly to a maximum of 1×10^{-4} over 25,000 iterations. This warm-up period is considered to improve convergence by limiting the learning rate during the initial stages of adaptive learning [52]. The learning rate was then cosine annealed down to 1×10^{-6} over the remaining iterations. Dropout was defined based on the compression ratio being trained; the ratios 80, 160 and 320 had dropout probabilities of 0.1, 0.2 and 0.4 respectively. This configuration was decided based on preliminary experiments, and follows that the dropout probability doubles for every halving of the rate. Intuitively, since all models had the same number of parameters, the ratio 320 model has much greater redundancy than the ratio 80 and will therefore be less influenced by small values of dropout. All models

used weight decay of 0.01. Fine-tuning was carried out for 100,000 iterations. The learning rate followed a similar schedule to pre-training with an initial learning rate of 2×10^{-6}, increased to 5×10^{-5} over 10,000 iterations and cosine annealed to 5×10^{-7} at the end of training. The maximum learning rate for fine-tuning is lower than pre-training to aid stability (some models were found to diverge at 1×10^{-4}). Where used, the CNN post-processor was added at the start of fine-tuning (only transformers were present for pre-training).

3.4 Implementation

All training, evaluation and utility code was written in Python 3.9. Extensive use was made of the PyTorch 1.9.1 library for GPU-enabled machine learning. Whilst PyTorch natively supports MAE and MSE, the PyTorch Image Quality Assessment (PIQA) library was used for SSIM and MS-SSIM. All training and evaluation was conducted on the University of Bristol BluePebble1 Cluster, using NVIDIA RTX-2080Ti GPUs. Due to GPU memory constraints gradient accumulation was used during training to achieve the batch size of 16.

Chapter 4

Results

The evaluation results of the proposed codec are presented and discussed in this chapter. To provide a point of comparison, JPEG and BPG were also evaluated on the same dataset. JPEG was chosen due to being the most commonly used image compression algorithm: the Independent JPEG Group implementation was used with 4:2:0 chroma subsampling [53]. BPG was selected for being the highest performing conventional image codec: the original implementation was used without chroma subsampling [16].

4.1 Evaluation Procedure

The proposed framework was trained at three compression ratios: 80, 160 and 320. The framework was then evaluated on an unseen dataset. The IEEE Computer Vision and Pattern Recognition (CVPR) Challenge on Learned Image Compression (CLIC) 2022 dataset was used for evaluation. The dataset, ratios and anchors were chosen to align with the CLIC 2022 evaluation procedure. This dataset contains 30 24-bit RGB images sourced from Unsplash, resized such that their longer edge is 2048 pixels long.

To evaluate the framework, it is necessary to divide the image into non-overlapping blocks. To generate the blocks, the image is first padded such at an integer number of patches fit in both dimensions (the patch size is not variable). The padded image is then split into blocks of defined size, with a border of smaller blocks where the full block size was unable to fit evenly. The input blocks are then processed by the network shown in Figure 3.2, with scalar quantisation applied instead of additive uniform noise. The decoded image blocks are reassembled into a full image, before cropping any padding that was applied. Lastly, distortion is measured between the original and decoded image using both PSNR and MS-SSIM. Note that alongside the latent symbols, the bitstream contains an 8-bit integer for the bottleneck dimension D_b and three 16-bit integers for the image height, image width and block size

respectively. This gives a total of 56 bytes of side-information. All other quantities, such as padding or border sizes can be inferred from the provided information.

Four configurations of the proposed codec will be presented in this section, for clarity these are defined below. The motivation for training and evaluating multiple variants was to conduct an ablation study; this permits the discussion of the performance and complexity contributions of the different components. All configurations of the framework were evaluated using the same block size as training (288).

- Bottleneck Transformer (BT) is simply as in Figure 3.2.
- Bottleneck Transformer with Post-processing (BT-PP) is as above with the post-processing shown in Figure 3.6.
- Bottleneck Transformer with TIDE (BT-TIDE) applies the TIDE algorithm described in subsection 3.1.3 to BT.
- Bottleneck Transformer with Post-processing and TIDE (BT-PP-TIDE) does the same for BT-PP.

4.2 Training Results

A plot of training results, including pre-training and fine-tuning, is provided in Figure 4.1. As described in subsection 3.3.2, the same pre-trained model is used for both BT and BT-PP, hence there is only one train-validation pair for pre-training but two for fine-tuning. Observing the pre-training results, validation loss decreases quickly at first but the rate of change then slows, whereas training loss decreases more steadily. As expected, there is no indication of over-fitting due to the dropout and weight decay applied. Fine-tuning removes this regularisation and leads to a sharp decrease in both training and validation loss. Intuitively the regularisation makes the task more difficult, and once removed the network is able to increase performance. As discussed, this two-stage procedure was found to achieve superior validation performance to simply training without regularisation; potentially due to the regularised pre-training preventing the network from getting stuck in a poor local optimum.

4.3 Image Samples

Three sets of image samples have been extracted from the evaluation dataset for visual comparison. The samples in Figures 4.2, 4.3 and 4.4 were compressed to 0.3, 0.15 and 0.075 bpp respectively. The samples have been selected to show a variety of content and textures: Figure 4.2 is densely detailed with various colours, Figure

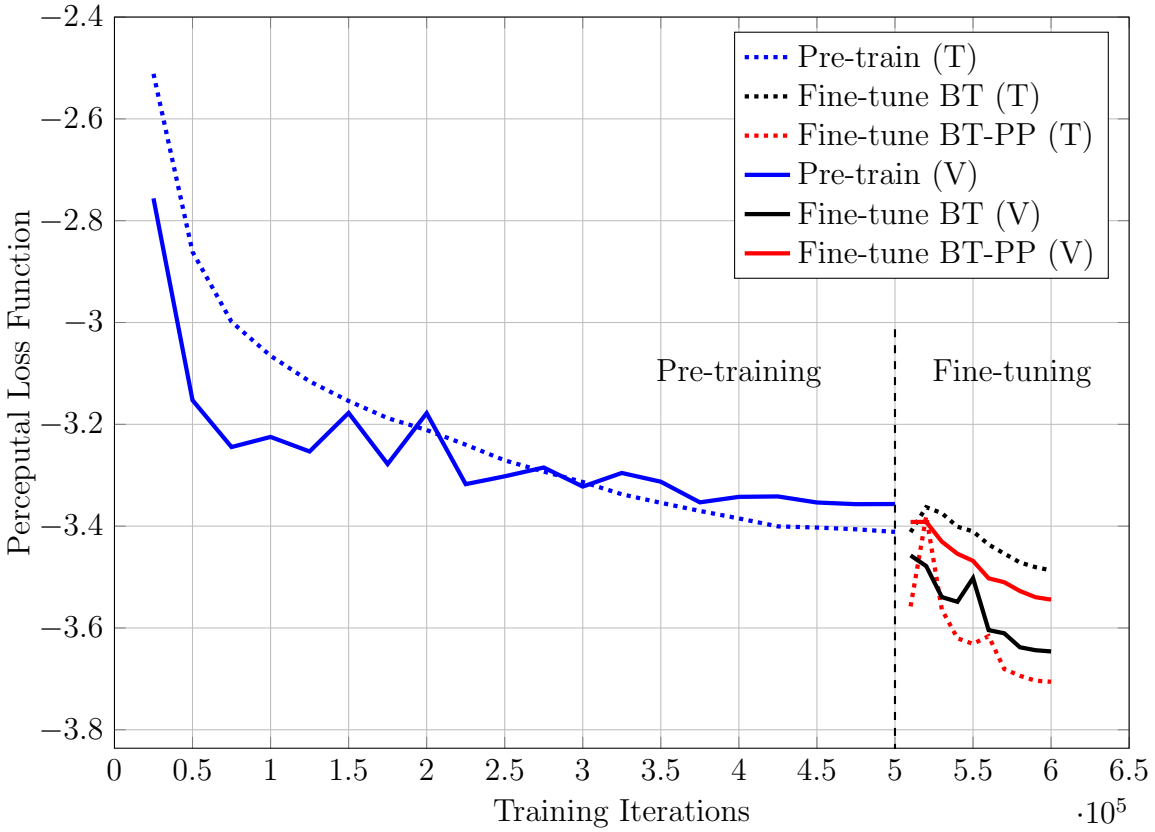
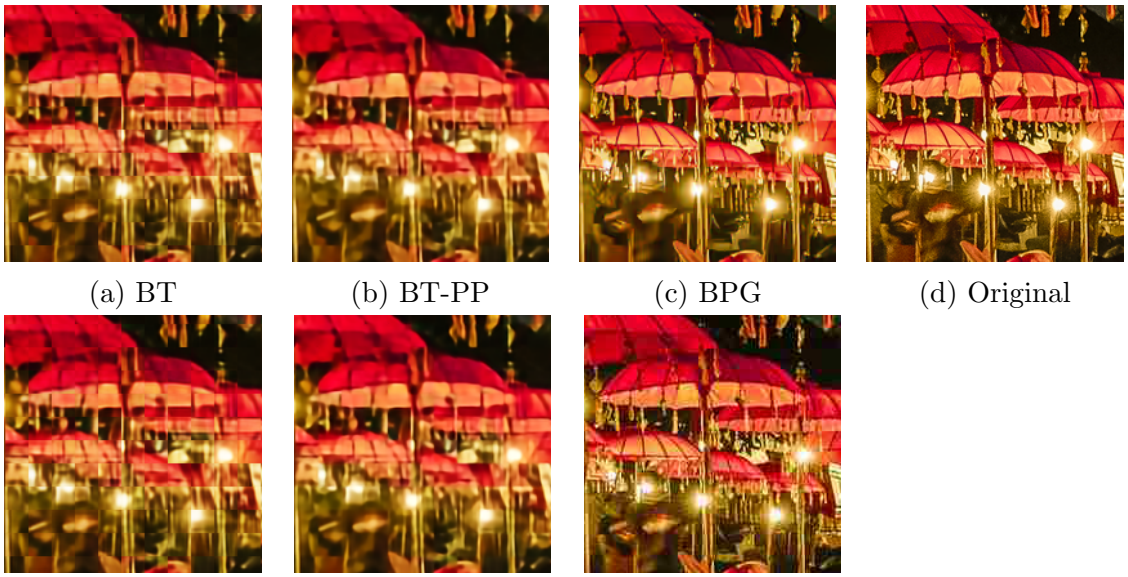


Figure 4.1: Training (T) and validation (V) results throughout the 0.15 bpp pre-training and fine-tuning.

4.3 contains the dynamic texture of water and Figure 4.4 displays an intricate rigid structure.

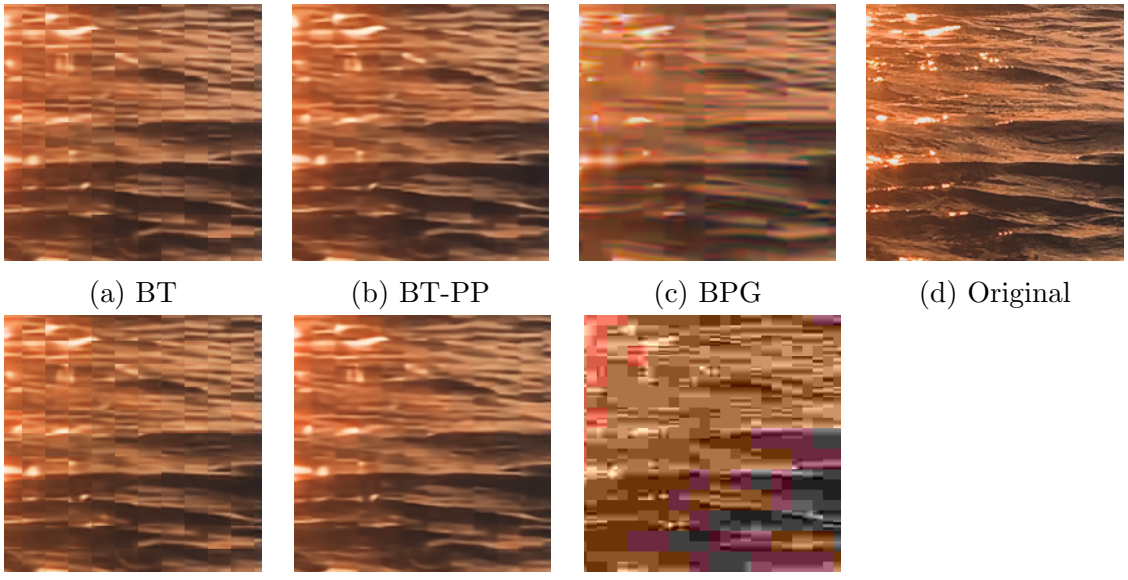
Observing these samples, patch artefacts are clearly visible in BT and BT-TIDE at all tested rates. The inclusion of post-processing in BT-PP and BT-PP-TIDE is mostly successful at suppressing these artefacts, making these the best performing variants of the proposed codec. There is little visible difference between the variants with and without TIDE. In Figure 4.2, BPG is considerably sharper than the proposed codec, with BT-PP and BT-PP-TIDE producing soft edges. On this sample JPEG is superior at preserving detail than BT-PP and BT-PP-TIDE but introduces some artefacts itself. For Figure 4.3, BT-PP and BT-PP-TIDE preserve more detail than BPG, with BPG also producing minor colour artefacts. JPEG produces severe artefacts in both colour and structure at this rate. Lastly in Figure 4.4, BPG contains more detail and more sharply defined edges than the proposed codec. JPEG was unable to compress the evaluation dataset to this rate so has been omitted from this comparison.



(a) BT (b) BT-PP (c) BPG (d) Original

(e) BT-TIDE (f) BT-PP-TIDE (g) JPEG

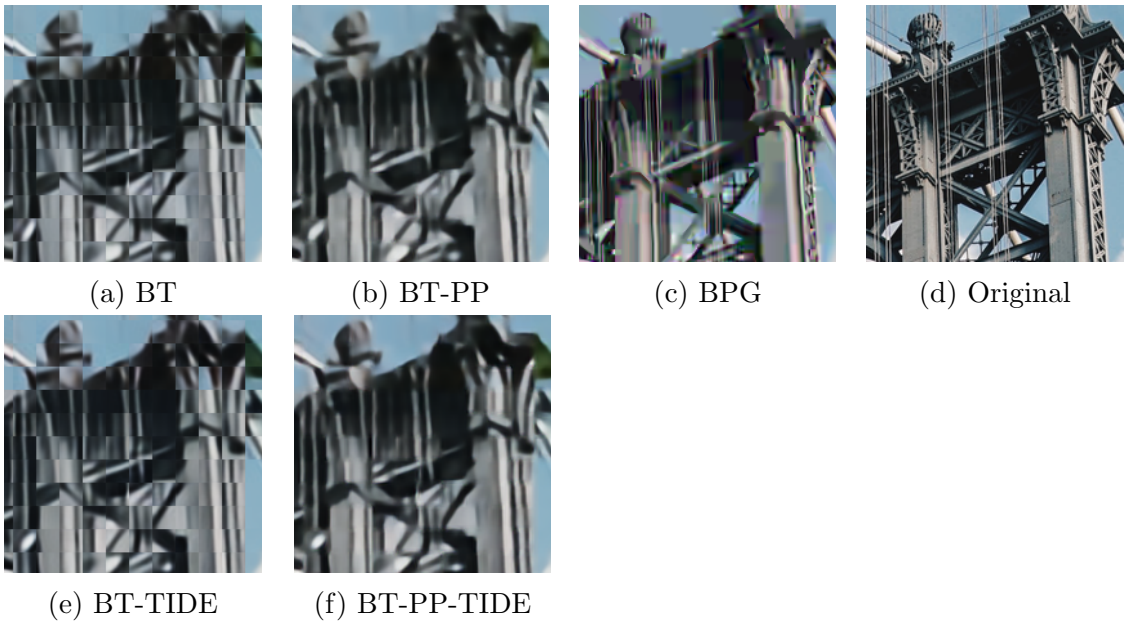
Figure 4.2: Crop of image from CLIC 2022 evaluation dataset compressed to 0.3 bpp.



(a) BT (b) BT-PP (c) BPG (d) Original

(e) BT-TIDE (f) BT-PP-TIDE (g) JPEG

Figure 4.3: Crop of image from CLIC 2022 evaluation dataset compressed to 0.15 bpp.



(a) BT (b) BT-PP (c) BPG (d) Original

(e) BT-TIDE (f) BT-PP-TIDE

Figure 4.4: Crop of image from CLIC 2022 evaluation dataset compressed to 0.075 bpp. Note that JPEG could not achieve this compression ratio so has been omitted.

4.4 Distortion Metrics

The rate-distortion performance of the codecs under evaluation, as measured by PSNR, is presented in Figure 4.5. It is apparent that BPG is consistently the superior codec by a large margin. Next is BT-PP-TIDE, making it the best performing variant of the proposed codec, at 0.5dB and 1.5dB inferior to BPG at 0.075 bpp and 0.3 bpp. The BT variants in descending order of performance are BT-PP-TIDE, BT-PP, BT-TIDE and BT, with approximately 0.5dB between best and worse. Compared to BT, JPEG is 1dB inferior at 0.15 bpp but marginally exceeds the performance at 0.3 bpp.

The rate-distortion plots for MS-SSIM are provided in Figure 4.5. In contrast to PSNR, BT-PP-TIDE has the highest MS-SSIM performance of the tested codecs, this superiority is consistent across the evaluated rates. The BT variants are in the same performance order as when measured by PSNR, with BPG approximately 0.02 MS-SSIM inferior to BT at all rates. Lastly, JPEG has very poor performance at 0.15 bpp, rising to 0.2 MS-SSIM lower than BPG at 0.3 bpp.

4.5 Complexity

The encoding and decoding time was recorded for all codecs. The per-image average across all rates is presented in Table 4.1. It is observed that BT and BT-PP have similar encoding time and that BT-TIDE and BT-PP-TIDE are 18 and 27 times

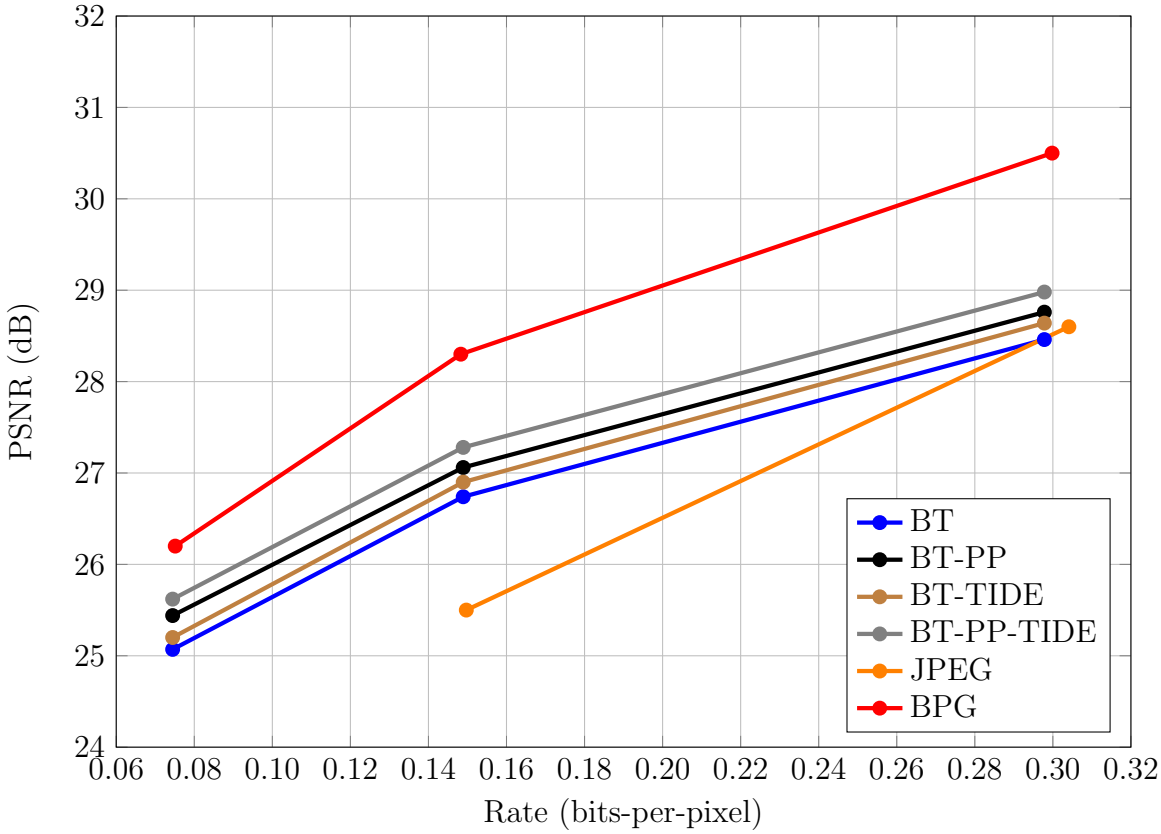


Figure 4.5: Rate distortion performance as measured by PSNR.

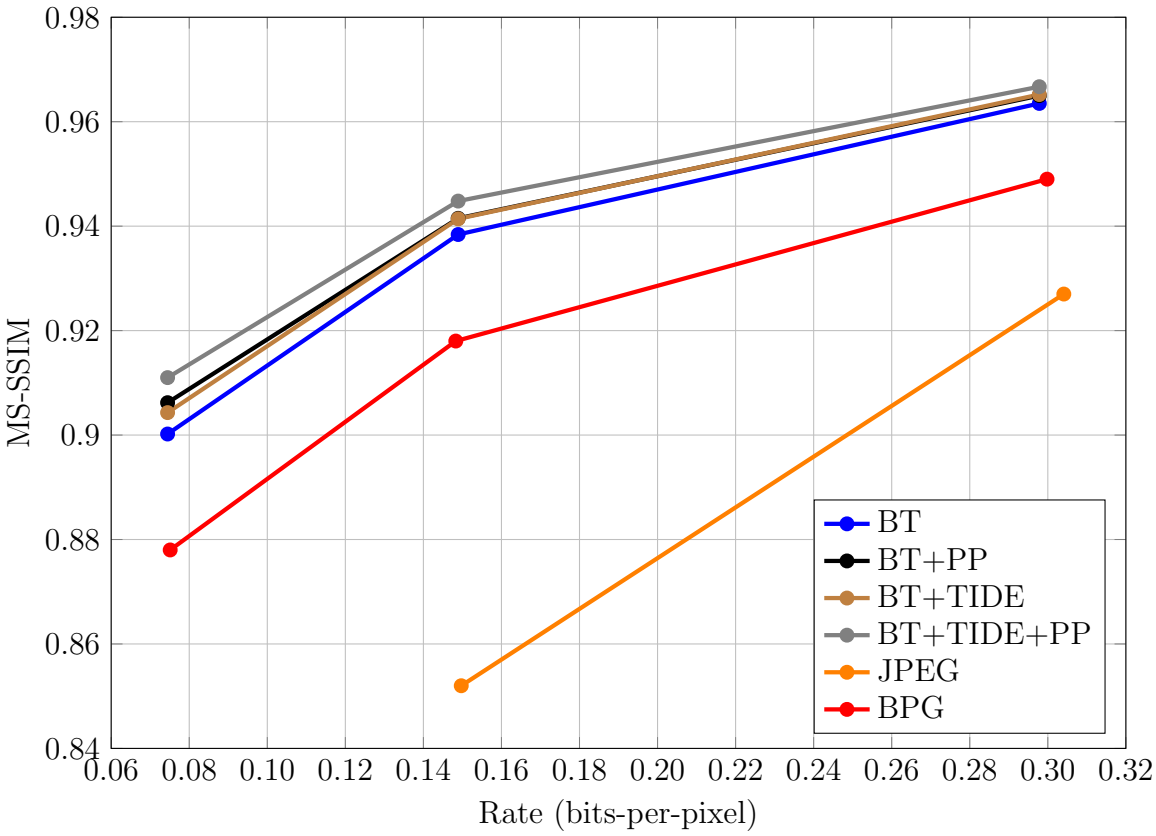


Figure 4.6: Rate distortion performance as measured by MS-SSIM.

Codec	BT	BT-PP	BT-TIDE	BT-TIDE-PP	JPEG	BPG
Encoding (s)	4.50	4.53	98.0	124	0.0664	0.937
Decoding (s)	3.70	3.63	3.67	3.74	0.00716	0.811

Table 4.1: Per-image encoding and decoding time in seconds.

slower at encoding than the variants without TIDE. All BT codecs have similar decoding time. JPEG is an order of magnitude lower in complexity than BPG, and even more notably lower in complexity than the BT codecs. BPG is around 4 times faster than BT at encoding and decoding.

4.6 Entropy

The distribution of the encoded latent symbols was recorded for the evaluation dataset. This distribution was used to compute the latent symbol entropy for BT (the other variants were found to be similar) presented in Table 4.2. It is seen that the entropy of the symbols is close to the symbol bit depth of 3. Statistical redundancy was computed as the percentage difference between the symbol bit depth and the entropy. The PMF of the latent symbols is given in Figure 4.7.

Rate (bpp)	0.075	0.150	0.300	mean
Entropy (bits)	2.995	2.992	2.975	2.987
Statistical Redundancy (%)	0.167	0.267	0.833	0.433

Table 4.2: Entropy and statistical redundancy for BT.

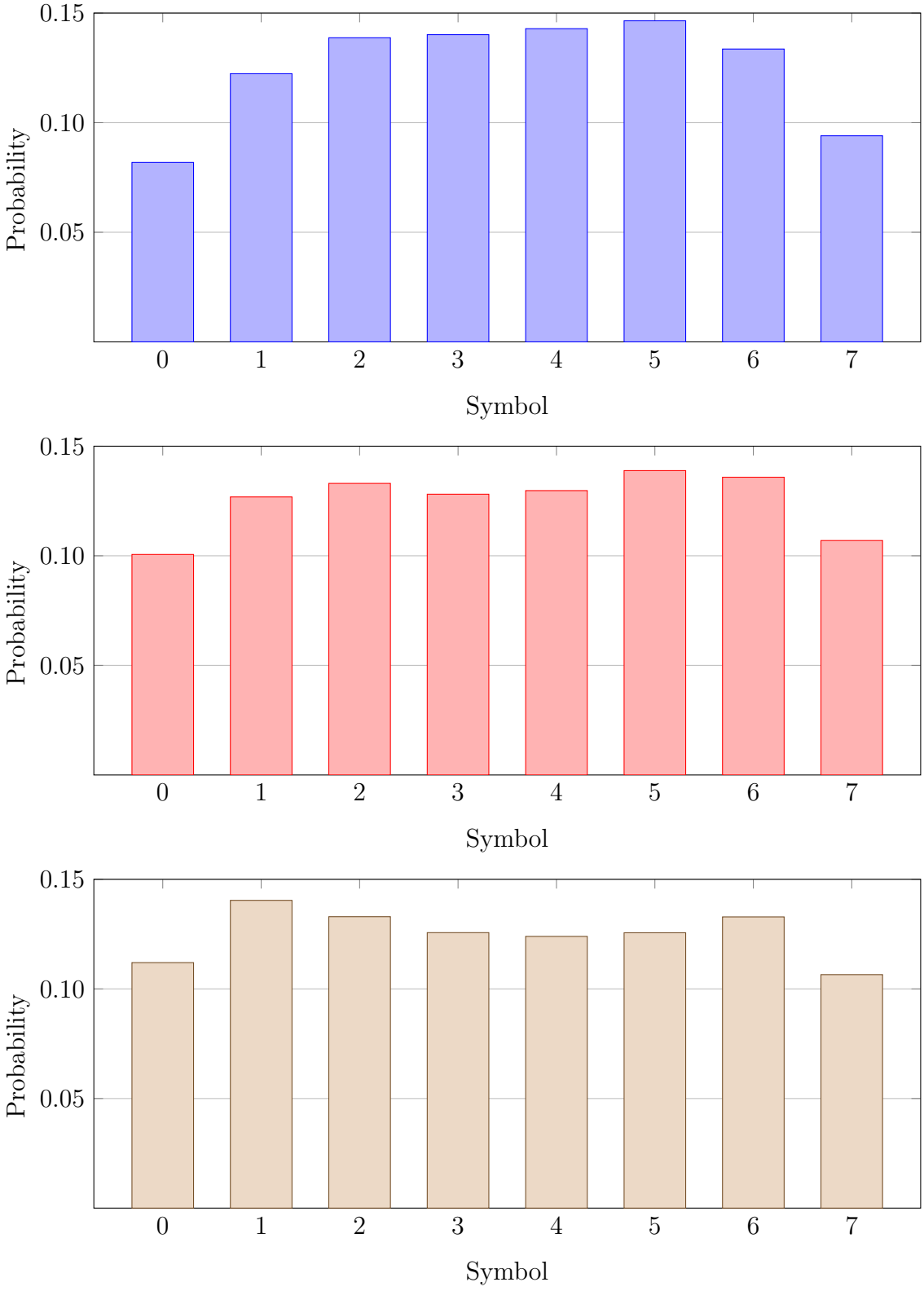


Figure 4.7: Latent symbol probability mass functions for BT at 0.3 bpp (top), 0.15 bpp (middle) and 0.075 bpp (bottom).

Chapter 5

Discussion

5.1 Compression Performance

In Figures 4.5 and 4.6, BT-PP-TIDE is shown to be the superior proposed codec variant for both PSNR and MS-SSIM. Both with and without post-processing TIDE is seen to provide a moderate improvement consistently across rates. However, there is little visible difference between the TIDE and non-TIDE codecs in the image samples provided in Figures 4.2, 4.3 and 4.4. Post-processing in contrast improves performance numerically and gives clear improvements in perceptual quality by suppressing patch artefacts. The presence of patch artefacts can be understood from the perceptive that the bottleneck between encoder and decoder applies dimensionality reduction to the latent vectors, which themselves represent patches. Observing the samples for BT closely, the patches themselves appear to have been reduced into dominant frequency domain components leading to discontinuities between the patches. Whilst successful at suppressing patch artefacts, both BT-PP and BT-PP-TIDE suffer from soft edges and poorly defined features. A factor contributing to this could be that in learning to suppress the patch artefacts produced by the transformer, the post-processor is required to blend between patches by learning a soft filter.

Compared to BPG, the BT codecs are inferior when measured by PSNR, but superior when measured by MS-SSIM. The inferior performance with respect to PSNR aligns with the image samples provided, particularly those at 0.3 bpp and 0.075 bpp. In these samples the patch artefacts produced by BT are clearly visible, as are the soft edges of BT-PP. The BT codecs performed well on the 0.15 bpp sample, perhaps due to the soft texture of water suiting the properties of the codec. Given these images samples, the superior performance on MS-SSIM is surprising. One explanation could be that the BT variants have performed poorly on the 0.3 bpp and 0.075 bpp samples due to their structural density but have superior MS-SSIM than BPG on less dense regions, potentially due to the intra-prediction of

BPG assigning bits non-uniformly across the images. The BT codecs are seen to strongly outperform JPEG at 0.15 bpp both numerically and in the provided sample, although JPEG is competitive on PSNR at 0.3 bpp and has a more sharply defined sample at this rate.

It is further indicative to consider the performance of the BT codecs in relation to the objective function used for training. As shown in equation (2.13) the perceptual loss function scales MS-SSIM with a weight of 0.4 but MSE by only 0.1. This could lead the network to prioritise the minimisation of MS-SSIM over MSE, providing a potential explanation for the superior MS-SSIM but inferior PSNR observed relative to BPG. To fully understand this relationship it would be necessary to train networks separately with each of MSE and MS-SSIM.

5.2 Complexity

As discussed in section 4.5, the proposed codec has greater time complexity than either of the conventional codecs, although BPG is only 4 times faster than the non-TIDE variants (it is however worth noting that all BT models were evaluated using a high-performance GPU). Whilst JPEG and BPG exhibited a trend between rate and time complexity, this was deemed insignificant for presentation relative to the complexity of the BT codecs. The BT codecs require an almost equivalent number of operations between rates, with the only variation being the dimensionality of the bottleneck transformations. The number of operations in the bottleneck is minimal compared to those in the transformers so no variation in time complexity relative to rate was anticipated or observed. Interestingly the inclusion of post-processing does not definitively increase the decoding time. Post-processing increases the number of computational operations but these may be too few to make a noticeable difference to the run-times, particularity on a heterogeneous computing cluster with nodes shared between compute jobs. It is likely that if the encoding and decoding were repeated many times a statistical difference would emerge. Using TIDE does not increase the decoding time but increases the encoding time by a factor of 20, with BT-PP-TIDE taking the longest to encode due to the additional complexity of training with the CNN.

5.3 Entropy

The proposed codec achieves compression by constraining both the dimensionality and entropy of latent symbols. Significantly it does not use entropy coding, the latent vectors are simply scaled and quantised to latent symbols. Referring to Table 4.2 it is evident that the symbol entropy closely approximates the defined upper-

bound $H_{max} = 3$. This low statistical redundancy demonstrates that entropy coding could not yield a significant reduction in rate, as the theoretical lower-bound is the entropy itself. Intuitively it can be understood that given the task of minimising distortion, the network has sought to maximise information transfer between encoder and decoder leading to entropic symbols. Figure 4.7 displays the PMF of the symbols at different rates, at the lower rates of 0.075 bpp and 0.15 bpp the symbols are more uniformly distributed than at 0.3 bpp. As the uniform distribution is the maximal entropy discrete distribution, this corresponds with the computed entropy values where the entropy of 0.3 bpp was lowest. It is plausible that given the higher rate there is less incentive for the network to maximise entropy, whereas at low rates the network must 'try harder' to get information from encoder to decoder.

The relationship between entropy, dimensionality and compression performance was explored during preliminary experiments. The values 8, 4, 2 and 1 were trailed for H_{max}, indicating empirically that 3 consistently performed highly. Speaking relatively, larger values of H_{max} introduce less noise at the cost of higher entropy. Therefore, for a given compression ratio using a larger value of H_{max} implies the use of a smaller bottleneck dimension D_b. I hypothesise that when using a value of H_{max} greater than optimal the severe dimensionality reduction hurts performance without the low noise providing a comparable benefit. In contrast using a value of H_{max} less than optimal causes the network to suffer from the excessive noise and this is not offset by the large number of symbols used. At the optimal value of H_{max} the network is resistant to the noise power and benefits from the moderate dimensionality. Due to the computational costs of training the networks, the optimal values of H_{max} and D_b for a given compression ratio have not been confirmed.

Chapter 6

Conclusion

6.1 Contributions

The proposed framework was demonstrated as a valid methodology for learned image compression. The main novelty of the framework was the reformulation of learned compression without explicit rate minimisation, and the use of constrained entropy. The latent symbol entropy was shown to be almost equivalent to the symbol bit depth used, confirming entropy coding to be unnecessary. Alongside this framework a novel architecture for learned image compression was proposed based on a transformer autoencoder. The novel architecture was demonstrated as compatible with learned image compression.

Despite promising results on MS-SSIM, the proposed codec was significantly inferior on PSNR than BPG. Visual samples support the conclusion that the performance of BPG has not been exceeded, although confirming this would require full subjective testing. Whilst other learned image compression codecs were not evaluated in this work, BPG is already known to have been surpassed by learned image compression. The performance of JPEG was exceeded at low rates but not at the higher rate evaluated when measured by PSNR.

The TIDE algorithm indicated numerical improvements although this was not clear from the image samples. TIDE dramatically increases the encoding complexity, without modifying the decoding process. There are however use-cases where this may be tolerable, such as image-hosting servers where an image is encoded once but downloaded and decoded many times. One advantage of TIDE is that a separate model does not need to be trained, so it could be provided as an optional codec feature. Once again subjective testing would be required to confirm the performance benefit provided. This work limited the number of TIDE iterations to a fixed value; the relationship between distortion and the number of TIDE iterations used was not explored. It is possible fewer iterations could be used for comparable benefit, which could reduce the computational complexity of encoding.

6.2 Limitations and Future Work

One limitation of this work is that a novel framework and architecture have been trained and evaluated together. It would be indicative to both train an existing learned compression architecture under the novel formulation, and to train the proposed architecture using the existing learned compression workflow. This would aid in drawing conclusions regarding the efficacy of both propositions. Additionally, training the Bottleneck Transformer on larger block sizes was not investigated, it is possible that processing full images would lead to superior performance due to the greater visual and spatial redundancy present. A further limitation is the few rates at which training and evaluation was conducted, as well as the lack of subjective testing. Lastly this work did not provide evaluation results for other learned image codecs, which would enable direct comparisons of distortion metrics and subjective quality. Future work would seek to overcome these limitations.

Appendix A

Software Traceability

A.1 Packages and Languages

Name	Supplier/Source/Author/Website	Use
Python 3.9	https://www.python.org/	Programming language.
NumPy 1.22	https://numpy.org/	Numerical computing library.
SciPy 1.7	https://scipy.org/	Computing discrete entropy.
PyTorch 1.9	https://pytorch.org/	Deep learning library.
PIQA 1.1	https://francois-rozet.github.io/piqa/piqa.html	Image quality metric library for PyTorch.
PILLOW 8.3	https://python-pillow.org/	Python image i/o, JPEG coding.
Fast-transformers	https://fast-transformers.github.io/	Linear transformer attention.
TorchInfo 1.5	https://github.com/tyleryep/torchinfo	PyTorch model summaries.
BPG 0.9	https://bellard.org/bpg/	BPG codec.

A.2 Python Files

Filename	Supplier/Source/Author/ Website	Use
datasets.py	Student written	Student code for loading training data with PyTorch.
evaluate.py	Student written	Implementation of proposed codec evaluation, used for results in chapter 4.
evaluate_bpg.py	Student written	Student code for evaluating BPG, used for results in chapter 4.
evaluate_jpeg.py	Student written	Student code for evaluating JPEG, used for results in chapter 4.
evaluate_TIDE.py	Student written	Implementation of proposed codec evaluation with TIDE, used for results in chapter 4.
loss.py	Student written	Student implementations of PSNR and PLF [50].
train.py	Student written	Student code for training proposed codec, as in section 3.3.
transform_io.py	Student written	Implementation of patch embedder and image reconstruction, as in section 3.2.
transform_models.py	Student written	Implementation of proposed models, as in section 3.2.
utils.py	Student written	Student utility functions.

Bibliography

[1] M. Meeker, "The state of the web." https://www.bondcap.com/report/it14/, 2014.

[2] F. Zhang and D. R. Bull, *Intelligent Image and Video Compression: Communicating Pictures*. Academic Press, Apr. 2021. Google-Books-ID: KiMZEAAAQBAJ.

[3] Z. Ghahramani, "Unsupervised Learning," Lecture Notes in Computer Science, pp. 72–112, Berlin, Heidelberg: Springer, 2004.

[4] J. Ballé, V. Laparra, and E. P. Simoncelli, "End-to-end Optimized Image Compression," *arXiv:1611.01704 [cs, math]*, Mar. 2017. arXiv: 1611.01704.

[5] J. Ballé, D. Minnen, S. Singh, S. J. Hwang, and N. Johnston, "Variational image compression with a scale hyperprior," *arXiv:1802.01436 [cs, eess, math]*, May 2018. arXiv: 1802.01436.

[6] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," in *Advances in Neural Information Processing Systems* (I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, eds.), vol. 30, Curran Associates, Inc., 2017.

[7] 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," in *International Conference on Learning Representations*, 2021.

[8] G. Wallace, "Overview of the JPEG (ISO/CCITT) still image compression standard," in *Other Conferences*, 1990.

[9] G. Hudson, A. Léger, B. Niss, I. Sebestyén, and J. Vaaben, "JPEG-1 standard 25 years: past, present, and future reasons for a success," *Journal of Electronic Imaging*, vol. 27, no. 4, pp. 1 – 19, 2018.

[10] N. Ahmed, T. Natarajan, and K. Rao, "Discrete Cosine Transform," *IEEE Transactions on Computers*, vol. C-23, pp. 90–93, Jan. 1974.

[11] D. A. Huffman, "A Method for the Construction of Minimum-Redundancy Codes," *Proceedings of the IRE*, vol. 40, pp. 1098–1101, Sept. 1952.

[12] P. Howard and J. Vitter, "Arithmetic coding for data compression," *Proceedings of the IEEE*, vol. 82, pp. 857–865, June 1994.

[13] C. Christopoulos, A. Skodras, and T. Ebrahimi, "The JPEG2000 still image coding system: an overview," *IEEE Transactions on Consumer Electronics*, vol. 46, pp. 1103–1127, Nov. 2000.

[14] G. J. Sullivan, J.-R. Ohm, W.-J. Han, and T. Wiegand, "Overview of the High Efficiency Video Coding (HEVC) Standard," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 22, pp. 1649–1668, Dec. 2012.

[15] J. Lainema, F. Bossen, W. Han, J. Min, and K. Ugur, "Intra Coding of the HEVC Standard," *IEEE Transactions on Circuits and Systems for Video Technology*, 2012.

[16] T. Nguyen and D. Marpe, "Objective Performance Evaluation of the HEVC Main Still Picture Profile," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 25, pp. 790–797, May 2015.

[17] Z. Wang, A. Bovik, H. Sheikh, and E. Simoncelli, "Image quality assessment: from error visibility to structural similarity," *IEEE Transactions on Image Processing*, vol. 13, pp. 600–612, Apr. 2004.

[18] H. Zhao, O. Gallo, I. Frosio, and J. Kautz, "Loss functions for image restoration with neural networks," *IEEE Transactions on computational imaging*, vol. 3, no. 1, pp. 47–57, 2016.

[19] Z. Wang, E. P. Simoncelli, and A. C. Bovik, "Multi-Scale Structural Similarity for Image Quality Assessment," in *in Proc. IEEE Asilomar Conf. on Signals, Systems, and Computers, (Asilomar*, pp. 1398–1402, 2003.

[20] F. Rosenblatt, *The perceptron, a perceiving and recognizing automaton Project Para*. Cornell Aeronautical Laboratory, 1957.

[21] M. Minsky and S. Papert, *Perceptrons*. Perceptrons, Oxford, England: M.I.T. Press, 1969.

[22] D. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors," *Nature*, 1986.

[23] G. Cybenko, "Approximation by superpositions of a sigmoidal function," *Mathematics of Control, Signals and Systems*, vol. 2, pp. 303–314, Dec. 1989.

[24] M. Leshno, V. Y. Lin, A. Pinkus, and S. Schocken, "Multilayer feedforward networks with a nonpolynomial activation function can approximate any function," *Neural Networks*, vol. 6, pp. 861–867, Jan. 1993.

[25] Z. Lu, H. Pu, F. Wang, Z. Hu, and L. Wang, "The Expressive Power of Neural Networks: A View from the Width," in *Advances in Neural Information Processing Systems*, vol. 30, Curran Associates, Inc., 2017.

[26] H. Robbins and S. Monro, "A Stochastic Approximation Method," *The Annals of Mathematical Statistics*, vol. 22, pp. 400–407, Sept. 1951.

[27] N. S. Keskar, D. Mudigere, J. Nocedal, M. Smelyanskiy, and P. T. P. Tang, "On Large-Batch Training for Deep Learning: Generalization Gap and Sharp Minima," *arXiv:1609.04836 [cs, math]*, Feb. 2017. arXiv: 1609.04836.

[28] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[29] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: a simple way to prevent neural networks from overfitting," *The journal of machine learning research*, vol. 15, no. 1, pp. 1929–1958, 2014.

[30] A. Krogh and J. Hertz, "A simple weight decay can improve generalization," *Advances in neural information processing systems*, vol. 4, 1991.

[31] I. Loshchilov and F. Hutter, "Decoupled weight decay regularization," *arXiv preprint arXiv:1711.05101*, 2017.

[32] Y. A. LeCun, L. Bottou, G. B. Orr, and K.-R. Müller, "Efficient BackProp," Lecture Notes in Computer Science, pp. 9–48, Berlin, Heidelberg: Springer, 2012.

[33] S. Hochreiter, "The vanishing gradient problem during learning recurrent neural nets and problem solutions," *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, vol. 6, no. 02, pp. 107–116, 1998.

[34] V. Nair and G. E. Hinton, "Rectified linear units improve restricted boltzmann machines," in *Proceedings of the 27th International Conference on International Conference on Machine Learning*, ICML'10, (Madison, WI, USA), pp. 807–814, Omnipress, June 2010.

[35] X. Glorot, A. Bordes, and Y. Bengio, "Deep sparse rectifier neural networks," in *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics* (G. Gordon, D. Dunson, and M. Dudík, eds.), vol. 15 of *Proceedings of Machine Learning Research*, (Fort Lauderdale, FL, USA), pp. 315–323, PMLR, 11–13 Apr 2011.

[36] M. Diganta, "Mish: A self regularized non-monotonic activation function," *British Machine Vision Conference*, 2020.

[37] D. Hendrycks and K. Gimpel, "Gaussian error linear units (gelus)," *arXiv preprint arXiv:1606.08415*, 2016.

[38] K. Fukushima, "Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position," *Biological Cybernetics*, vol. 36, pp. 193–202, Apr. 1980.

[39] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel, "Backpropagation Applied to Handwritten Zip Code Recognition," *Neural Computation*, vol. 1, pp. 541–551, Dec. 1989.

[40] J. J. Hopfield, "Neural networks and physical systems with emergent collective computational abilities.," *Proceedings of the National Academy of Sciences of the United States of America*, vol. 79, pp. 2554–2558, Apr. 1982.

[41] A. Sherstinsky, "Fundamentals of Recurrent Neural Network (RNN) and Long Short-Term Memory (LSTM) network," *Physica D: Nonlinear Phenomena*, vol. 404, p. 132306, Mar. 2020.

[42] S. Hochreiter and J. Schmidhuber, "Long Short-Term Memory," *Neural Computation*, vol. 9, pp. 1735–1780, Nov. 1997.

[43] D. Bahdanau, K. Cho, and Y. Bengio, "Neural Machine Translation by Jointly Learning to Align and Translate," *arXiv:1409.0473 [cs, stat]*, May 2016. arXiv: 1409.0473.

[44] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding," in *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, (Minneapolis, Minnesota), pp. 4171–4186, Association for Computational Linguistics, June 2019.

[45] A. Katharopoulos, A. Vyas, N. Pappas, and F. Fleuret, "Transformers are RNNs: Fast Autoregressive Transformers with Linear Attention," pp. 5156–5165, PMLR, Nov. 2020.

[46] N. Parmar, A. Vaswani, J. Uszkoreit, Kaiser, N. Shazeer, A. Ku, and D. Tran, "Image Transformer," *arXiv:1802.05751 [cs]*, June 2018. arXiv: 1802.05751 version: 3.

[47] Z. Wan, J. Zhang, D. Chen, and J. Liao, "High-Fidelity Pluralistic Image Completion with Transformers," *arXiv:2103.14031 [cs]*, Mar. 2021. arXiv: 2103.14031.

[48] D. Ma, F. Zhang, and D. R. Bull, "CVEGAN: A Perceptually-inspired GAN for Compressed Video Enhancement," *arXiv:2011.09190 [cs, eess]*, Nov. 2020. arXiv: 2011.09190.

[49] A. Katharopoulos, A. Vyas, N. Pappas, and F. Fleuret, "Transformers are RNNs: Fast Autoregressive Transformers with Linear Attention," *arXiv:2006.16236 [cs, stat]*, Aug. 2020. arXiv: 2006.16236.

[50] D. Ma, F. Zhang, and D. R. Bull, "Video compression with low complexity CNN-based spatial resolution adaptation," *Applications of Digital Image Processing XLIII*, p. 9, Aug. 2020. arXiv: 2007.14726.

[51] Unsplash, "Unsplash dataset." `https://unsplash.com/data`, 2022.

[52] L. Liu, H. Jiang, P. He, W. Chen, X. Liu, J. Gao, and J. Han, "On the variance of the adaptive learning rate and beyond," in *International Conference on Learning Representations*, 2020.

[53] IJG, "Reference sources." https://jpegclub.org/reference/reference-sources.