Data Compression via Nonlinear Transform Coding using Artificial Neural Networks

MTHE 493 Thesis

Spencer Hill, Wyllie Schenkman, Jordan Curnew, Mark Benhamu

April 10, 2023

Abstract

This thesis presents an implementation of nonlinear transform coding (NTC) that uses artificial neural networks (ANNs) for lossy data compression. In today's digital age, it is not difficult to motivate the need for efficient data compression. There is no end to society's reliance on multi-media information within every industry, including healthcare. This project focuses in particular on the compression of computerized tomography (CT) scans and examines the social, economic, and environmental impacts that improvements in their compression would have.

Today's standard image compression techniques commonly rely on linear transform coding, a method that leverages orthogonal transformations to decorrelate and compress a source. NTC offers a more sophisticated approach but has been historically limited by the intractability of determining appropriate nonlinear transforms in high dimensions. However, recent advancements in computer hardware and the emergence of ANNs have provided a method to implement general nonlinear transforms, motivating significant work on the topic of NTC.

We provide a mathematical formulation of an NTC system, which comprises nonlinear analysis and synthesis transforms implemented using ANNs. A differentiable cost function is derived, necessitating approximations of the source entropy model and a proxy for scalar quantization. This cost function can be optimized using stochastic gradient descent, and by incorporating Lagrangian optimization can be used for rate-distortion traversal. An iterative design process is employed to enhance the model's compression capabilities, incorporating techniques such as image tiling, regularization techniques and more sophisticated distortion measures and activation functions, namely the structural similarity index measure and generative divisive normalization, respectively.

The NTC model is tested using increasingly complex data sources, including a one-dimensional Laplacian distribution, images of handwritten digits, and finally the target application of CT scans. For image compression, the analysis and synthesis transforms are implemented using convolutional neural networks. The success of the project is measured by the achieved rate-distortion function and perceptual quality of the reconstructed images. A comparison of the model with the JPEG standard is presented, and although JPEG outperforms NTC in this application we give an outline of future work that would allow this system to surpass current industry standards.

In writing this thesis, we extend a special thank you to the Queen's School of Computing for generously providing us with access to their GPU cluster. Without these resources, our model implementation and testing would have been severely limited.

Contents

List of Tables

List of Figures

1 Problem Definition

As the world continues to rely increasingly on multi-media communication, the need for efficient compression, storage, and transmission of data is higher than ever. Although transform coding has been the universal compression method for the past decades, a potential shift from linear to nonlinear transform codes has been a topic of interest in recent years. The emergence of artificial neural networks (ANNs) and massively parallel computational resources has produced new strategies for solving the previously intractable problem of determining optimal nonlinear transforms in high dimensions. With these developments, nonlinear transform coding (NTC) has the potential to leverage the theoretical advantages in learning non-Gaussian source distributions [1]. Still, practical questions remain surrounding nonlinear transform coding, most notably how to perform stochastic gradient optimization of an ANN-based system and whether it can compare to industry standards such as JPEG.

The problem being addressed in this thesis is the development of an NTC model for the compression of computerized tomography (CT) scan images used to detect cancer lesions. Specifically, we aim to implement ANN analysis and synthesis transforms and design a custom, differentiable cost function to jointly optimize compression rate and distortion. The primary challenge, as in most data compression, is achieving practically useful compression rates with distortion sufficiently low for our application of CT scans. This requires a thorough analysis of relevant stakeholders to understand all facets of the problem and the relevant constraints.

The design of the neural network-based compression system must align with stakeholder needs, including patients, practitioners, and the healthcare industry. Patients require extremely accurate compression, as any misdiagnoses resulting from the distortion of a reconstructed CT scan can have negative health effects. Practitioners need fast access to high-quality images from any clinic location, including remote areas lacking in internet bandwidth. The healthcare industry has requirements for data storage and transmission and compliance with regulatory standards, as well as a significant environmental and economic interest in compressing CT scans. The needs of these stakeholders and how they impact the project direction are further detailed in Section 2.2. Further, the deployment of any system must consider computational power and speed. The ability to reconstruct images must be immediately available to patients and practitioners through most laptops and smartphones, limiting the complexity of neural networks used.

The successful implementation of this project will contribute to the development of more efficient and effective compression techniques for medical imaging, particularly in the detection of cancer lesions. It will also provide insights into the application of artificial neural networks in image compression and the challenges associated with balancing compression rate and distortion.

2 Engineering Impact

In this section, we identify relevant stakeholders and conduct a triple-bottom-line analysis to understand the social, economic, and environmental impact that this project has. Regulatory concerns are also addressed, as well as a thorough economic analysis.

2.1 Standards and Regulation

The CT scans used to train the models in this project were obtained from the National Institute of Health (NIH). While sensitive patient data was used, all personally identifiable information was removed by NIH to protect the privacy of the individuals whose CT scans were used [2]. By using this public data set, the team did not infringe on the privacy rights of any study participants.

We note again the damning negative impact of over-compression on patients and practitioners. Although no legal standards regarding the compression of CT scans were found, papers such as the *Pan-Canadian Evaluation of Irreversible Compression Ratios for Development of National Guidelines* give recommended compression ratios for specific anatomical regions [3]. The recommended compression ratios from that paper are summarized in Table 1. These compression ratios were produced from a qualitative analysis of images compressed using current industry standards. It is therefore important to consider them in the context of the distortion that compression produces, which can be translated to our system to determine similar guidelines for compression rates. As system deployment is outside the scope of the project, it is not necessary to establish or follow compression standards. Nevertheless, we continue to emphasize the negative effects of over-compression and these constraints' influence on project direction.

Table 1: Recommended Compression Ratios for CT by Anatomical Region [3]

Anatomical Region	CT
Angio	16–24
Body	JPEG 10–15; J2K 10
Breast	8–12
Chest	10–15
MSK	JPEG 20–30; J2K 20
Neuro	16–24
Pediatrics	10–15

2.2 Stakeholders and Triple Bottom Line

An in-depth stakeholder analysis is a critical part of a successful engineering design project. For our application of compressing CT scans, we identified important stakeholders and ensured the needs of each are met. These stakeholders' needs were then used to inform the constraints, goals, and design choices specific to our project. To identify the stakeholders of the project, the group first created a brainstorm map, as seen in Figure 1, to understand the organizations and individuals affected by the project. For *organizations*, there was only one major stakeholder identified, that being hospitals and, more broadly, the healthcare system. The *individuals* category was divided into two subcategories: patients who are having CT scans performed and health practitioners taking or using the CT scans. Thus, the stakeholder analysis was performed on the following key stakeholders: the healthcare system, patients, and practitioners.

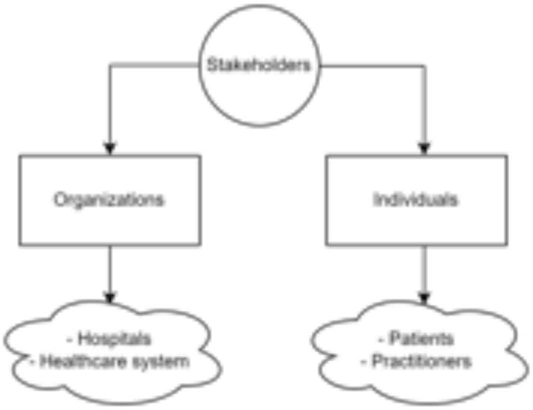


Figure 1: Brainstorm of Stakeholder selection.

The stakeholder analysis was conducted by doing further background research on each stakeholder and their involvement in our project, then performing the triple-bottom-line analysis. This analysis will be used to understand the social, environmental, and economic impacts of the project.

2.2.1 Patients

Two studies from the *World Journal of Radiology* and the *Journal of Thoracic Imaging* found that low-quality CT images resulted in higher rates of misdiagnoses [4]. Along with this, close to 30% of CT scans taken in the United States are classified as low-quality [5]. Due to this, one of our success metrics for our project is to maintain image quality, with a goal to reduce the percentage of low-quality images upon the implementation of our solution. This is crucial to the first key stakeholder, patients, as a misdiagnosis as a result of an over-compressed CT scan could have significant negative impacts, including delayed or incorrect treatment. Delayed treatment can lead to tumour growth and even prove fatal to a patient [6]. Additionally, an incorrect treatment could lead to unnecessary surgery, which in the United States can have financial repercussions or even fatal injuries. In 2019, between 80,000 and 160,000 misdiagnoses led to medical-related injury [7]. For this reason, ensuring image quality can reduce the amount of these misdiagnoses each year, improving patient safety.

Social Impact: The social impact on the patients is the reduction in misdiagnoses as a result of an over-compressed CT scan. This is beneficial as fewer patients will be mistreated or have delayed treatment, leading to fewer medical-related injuries or fatalities [6].

Economic Impact: The economic impact on patients arises from false negative diagnoses, which can severely complicate the necessary treatment. Further, unclear scans may lead to unnecessary additional testing. For patients without health insurance, both scenarios represent significant added costs.

Environmental Impact: No notable environmental impacts have been identified for the patients.

2.2.2 Practitioners

In rural parts of Canada, less than 15% of emergency departments (ED) have access to CT scanners [8]. For this reason, practitioners assess CT scans sent to them by radiologists. This can create a bottleneck in the process if CT scan image sizes are impractically large for practitioners to open or view. Due to this, one of our success metrics for our project is to reduce the image size, with a goal to increase the availability of CT images for rural practitioners upon solution implementation. For remote practitioners, long wait times for the download and upload of the images cause inefficiencies in their work. Along with this, maintaining a high-quality CT image scan is beneficial to practitioners, as a low-quality image may necessitate resending or even retesting, further delaying their ability to perform their job [9].

Social Impact: The social impact on the practitioners is increased accessibility of high-quality CT scans for rural office workers. This improvement would mean practitioners are able to expand treatment to patients who need immediate scan results in rural locations.

Economic Impact: The economic impact on practitioners also comes from increased access to high-quality CT scans for rural office workers. This is economically beneficial as practitioners will be able to treat more patients, increasing their pay. This is especially beneficial in the United States, where healthcare is privatized.

Environmental Impact: No notable environmental impacts have been identified for the practitioners.

2.2.3 Healthcare System

In the United States alone 80 million CT scans are taken every year, and medical records are required by law to be retained for 10 years [10][11]. For this reason, more than 800 million CT scans are stored at a given time. Modern CT machines take upwards of 640 slices per CT scan, with each slice being approximately 0.5 MB [12][13]. This results in 320 MB of data being produced for a CT scan. In total, there are at least 256,000 terabytes of CT images are stored at any given time. Due to this, one of the success metrics for our project is reducing image size, specifically aiming to decrease the cost of storing CT images for hospitals. This is extremely important for the healthcare system, as more funding can be allocated to patient care and other important resources [14].

Social Impact: The social impact on the healthcare system is the reallocation of funding to patient care and more medical resources. With the cost of CT scan storage being reduced (as the compression size is decreased), then upon implementation of the project, social resources for hospitals can be improved.

Economic Impact: The economic impact on the healthcare system comes in the form of savings from CT scan storage. Developed further in Section 2.3, the cost of total yearly CT scan storage would decrease significantly with an increased compression ratio. For this reason, the healthcare system will economically benefit upon implementation of the project.

Environmental Impact: The environmental impact on the healthcare system comes from the power saving per CT scan. Reducing the size of CT scans will also reduce the healthcare system's power required for the processing, transmission, and storage of these scans. This will have a positive impact on the environment by lowering the carbon footprint of CT scans.

2.3 Economic Analysis

As seen in the stakeholder analysis, CT scans play a crucial role in medical imaging. However, the large file sizes pose significant challenges to the storage and transfer of medical images, resulting in high storage costs and slow transfer times. Effective compression of these images while maintaining high levels of quality and diagnostic accuracy would be economically lucrative.

This analysis assesses the economic impact of implementing the developed model to compress CT scan images in the healthcare system. The team analyzed the potential cost savings for storing and accessing patient records at varying compression rates. As can be seen in the results, Section 6, the developed model achieves different compression rates with associated reconstruction distortions. Table 2 shows the total storage required in TB, total yearly storage cost in USD, and rural area download time per CT-scan for compression ratios of 1:1, 2:1, 5:1, and 10:1.

Table 2: Storage Cost & Access Times for Varying Compression Ratios

Compression Ratio	Storage Required (TB)	Yearly Storage Cost (USD)	Rural Access Times per CT scan (min)
1:1	250,000	30,000,000	7.23
2:1	125,000	15,000,000	3.62
5:1	50,000	6,000,000	1.45
10:1	25,000	3,000,000	0.72

The following assumptions were made for calculating yearly storage costs and upload/download times for hospitals in rural areas:

- 80 million CT scans are conducted in the USA each year [10].
- The storage cost is $0.01 per month per GB [15].
- The file size of a full CT scan is 320MB [12][13].
- The yearly number of scans is equal, assuming equivalent turnover and entry rates of stored data for consistent year-on-year comparisons.
- CT Scans must be stored for 10 years [11].
- The connection speed for hospitals in rural areas is 5.9 Mbps [16].

The analysis demonstrates that effective compression of CT scans can reduce storage costs by millions. In terms of access times, saving practitioners three to six or more minutes per access significantly improves operational efficiency. It remains crucial to consider the impact of image quality on practitioners and patients. As mentioned in Section 2.2.3, poor image quality has been found to decrease diagnostic accuracy. Misdiagnoses can incur economic costs from additional treatment, but the impact of fatalities from misdiagnoses cannot be measured economically. As such, it is essential to carefully balance the benefits of compression against the potential risks of misdiagnoses due to poor image quality.

3 Background

The following sections describe background information necessary to understand the project. A review of relevant data compression topics are discussed, followed by an overview of neural networks and a literature review of related work.

3.1 Data Compression

Data compression broadly aims to reduce the size of a data source for either transmission or storage. In particular, by removing the statistical redundancies of the data source, it can be represented in a lower number of bits while maintaining either all or most of the key information. Data compression can be categorized into two types: lossless and lossy compression. Lossless data compression uses an invertible transform that removes statistical redundancy such that the reconstructed data $\hat{X}$ is identical to X. The rate of lossless data compression is lower bounded by the entropy of a source, which for discrete random variables is given in bits by

$$H(X) = -\sum_{x \in X} P(x) \log_2 P(x). \tag{3.1}$$

for its probability distribution P. Conversely, lossy data compression aims to minimize the distortion between X and $\hat{X}$ without the requirement of exact reconstruction. Lossy compression can achieve lower compression rates, with the tradeoff that the distortion between X and $\hat{X}$ increases inversely proportional to the rate. Thus, the key question of lossy data compression is the minimum number of bits required to represent the reproduction $\hat{X}$ under the requirement that the distortion between X and $\hat{X}$ be less than D [17].

3.1.1 Rate Distortion Theory

For a source $X = (x_1, \dots, x_n)$ and reconstruction $\hat{X} = (\hat{x}_1, \dots, \hat{x}_n)$, we define

$$d(X, \hat{X}) = \sum_{i=1}^{n} d(x_i, \hat{x}_i) \tag{3.2}$$

and call $d : \mathcal{X} \times \hat{\mathcal{X}} \to \mathbb{R}$ the distortion between X and $\hat{X}$ [17]. We generally require that $d(x, y) \geq 0$ $\forall x \in \mathcal{X}, y \in \hat{\mathcal{X}}$ [17]. For a lossy code, we define the rate R of the code as

$$R = \text{number of bits required to represent one source symbol.}$$

Rate-distortion theory gives the lower bound for the rate at a maximum distortion D as

$$R(D) = \min_{p(\hat{x}|x) \,:\, \mathbb{E}d(X,\hat{X}) \leq D} \{I(X, \hat{X})\} \tag{3.3}$$

with $p(\hat{x}|x)$ the conditional probability mass function of $\hat{X}$ given X and $I(X, \hat{X})$ the mutual information between X and $\hat{X}$ [17]. This makes explicit the tradeoff between rate and distortion, as $R(D)$ is a nonincreasing, convex function, meaning that decreasing the rate necessitates increasing the maximum allowable distortion.

3.1.2 Scalar Quantization

Lossy compression is usually achieved using quantization, which aims to discretize continuous values to a finite range. Specifically, an N-point scaler quantizer is the mapping $Q : \mathbb{R} \to C$, where C is the codebook

$$\mathcal{C} = \{y_1, y_2, ..., y_N\} \subset \mathbb{R} \tag{3.4}$$

such that $y_1 < y_2 < ... < y_N$ [18]. We denote the quantizer cells of the Q as

$$R_i = \{x \subset \mathbb{R} : Q(x) = y_i\} \tag{3.5}$$

and note that the set of all R_i form a partition of $\mathbb{R}$ [18]. An N-level scalar quantizer is evaluated based on two primary performance metrics, a distortion $D(Q)$ and rate $R(Q)$ defined as

$$D(Q) = \mathbb{E}d(X, Q(X)) \tag{3.6}$$

$$R(Q) = \log_2(N) \tag{3.7}$$

As described above, $d(x, y)$ is a distortion measure reflecting the loss induced by reproducing x as y. In particular, for the common squared distortion measure $d(x, y) = (x - y)^2$, you can write

$$\begin{aligned} \mathbb{E}d(X, Q(X)) &= \sum_{i=1}^{N} \mathbb{E}[(x_i - Q(x_i))^2] \\ &= \sum_{i=1}^{N} \int_{R_i} (x_i - y_i)^2 f(x) dx \text{ [18].} \end{aligned} \tag{3.8}$$

For quantizing training samples $X = X_1, \ldots, X_M$ drawn from an unknown distribution, this expected distortion becomes

$$\mathbb{E}d(X, Q(X)) = \frac{1}{M} \sum_{i=1}^{M} (x_i - Q(x_i))^2 \tag{3.9}$$

which is the popular mean-squared error (MSE) distortion [18]. We refer to an N-level scalar quantizer as optimal if it has minimal distortion, that is Q^* is optimal if

$$D(Q^*) = \min_{Q \in Q_N} D(Q) \tag{3.10}$$

for Q_N the set of all N-level scalar quantizers [18]. In practice, an optimal quantizer is often designed using the Lloyd-Max Algorithm, an iterative procedure that aims to satisfy the Nearest Neighbour and Centroid Conditions, which are necessary but not sufficient conditions for optimality. This project is not concerned with the optimality of the quantizers used, instead aiming to design transforms that fit the data to a quantizer that is fixed a priori.

3.1.3 Linear Transform Coding

For a data sample $\mathbf{X}$ of size $1 \times N$, linear transform coding (LTC) converts $\mathbf{X}$ according to Figure 2, where $\mathbf{Y} = \mathbf{AX}$ for an orthogonal $N \times N$ matrix $\mathbf{A}$ [19].

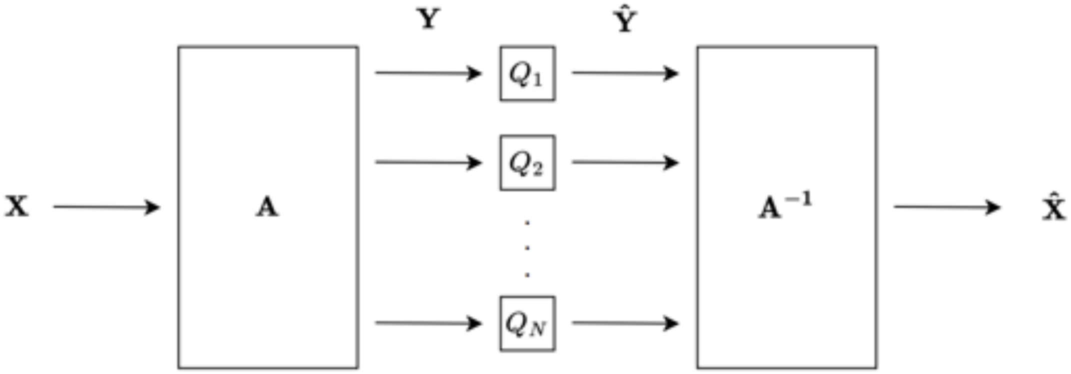


Figure 2: Block diagram of a linear transform coder

In LTC, N independent scalar quantizers $\{Q_1, Q_2, ..., Q_N\}$ are respectively applied to the decorrelated inputs $\mathbf{Y} = \{y_1, y_2, ..., y_N\}$ [19]. The inverse transform $\mathbf{A^{-1}}$ is applied to the quantized output $\hat{\mathbf{Y}}$ to produce the reconstruction $\hat{\mathbf{X}}$ [19]. LTC can also be extended to a two-dimensional input (such as an image) by considering $\mathbf{X}$ as an $N \times N$ block matrix and applying the transform $\mathbf{Y} = \mathbf{AXA^T}$ [19]. As $\mathbf{A}$ is an orthogonal matrix transformation, it is invertible and lossless. For MSE, the overall distortion of the system is therefore

$$\begin{aligned} D_{LTC} &= \sum_{i=1}^{N} \mathbb{E}[(X_i - \hat{X}_i)^2] \\ &= \mathbb{E}[||\mathbf{X} - \hat{\mathbf{X}}||^2] \\ &= \mathbb{E}[||\mathbf{A}(\mathbf{X} - \hat{\mathbf{X}})||^2] \\ &= \mathbb{E}[||\mathbf{Y} - \hat{\mathbf{Y}}||^2] \text{ [19].} \end{aligned} \tag{3.11}$$

Hence, the overall system distortion is equal to the distortion of the quantizers $Q_1, \ldots, Q_N$. These quantizers can be optimized using the Lloyd-Max algorithm to minimize the overall distortion of the system.

The question for LTC becomes determining an appropriate orthogonal transform matrix $\mathbf{A}$, examples of which include the Karhunen-Loeve (KL) Transform and Discrete Cosine Transform (DCT) [19]. The celebrated JPEG compression standard uses the DCT as its orthogonal transform, specifically the matrix

$$\mathbf{T} = \{t_{mn}\}_{m,n=0}^{N-1} = \begin{cases} \sqrt{\frac{2}{N}} \cos(\frac{\pi}{N} m(n + \frac{1}{2})) & m = 1, \ldots, N-1, n = 0, \ldots, N-1 \\ \sqrt{\frac{1}{N}} \cos(\frac{\pi}{N} m(n + \frac{1}{2})) & m = 0, n = 0, \ldots, N-1 \end{cases} \text{[19].} \tag{3.12}$$

In JPEG, the two-dimensional version of $\mathbf{T}$ is applied to 8×8 blocks of pixels, quantized, and then losslessly encoded according to the system shown in Figure 3 [20].

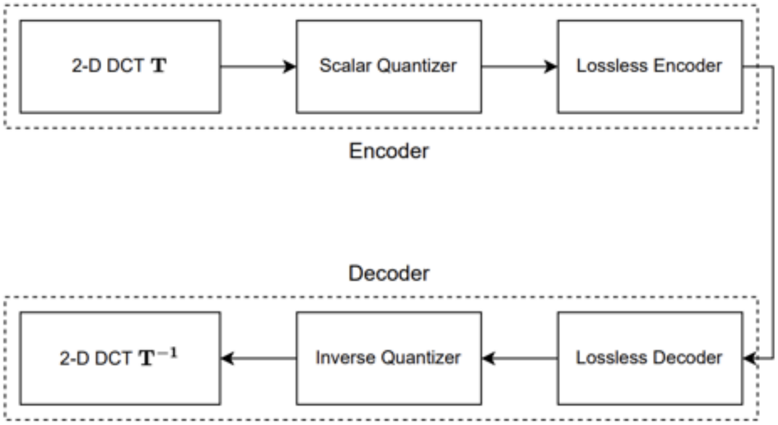


Figure 3: Block diagram of JPEG

3.1.4 Nonlinear Transform Coding

NTC replaces the linear transformation, **A**, with nonlinear analysis and synthesis transformations. Nonlinear functions are more versatile and generally capable of achieving better decorrelation. Furthermore, as seen in Figure 7, LTC is limited to lattice quantization, while NTC can further adapt to the source data distribution [1]. The issue with NTC is that it is difficult in practice to determine an appropriate nonlinear transform, particularly in high dimensions [1]. The universal approximation capabilities of Artificial Neural Networks offer a potential solution to this problem.

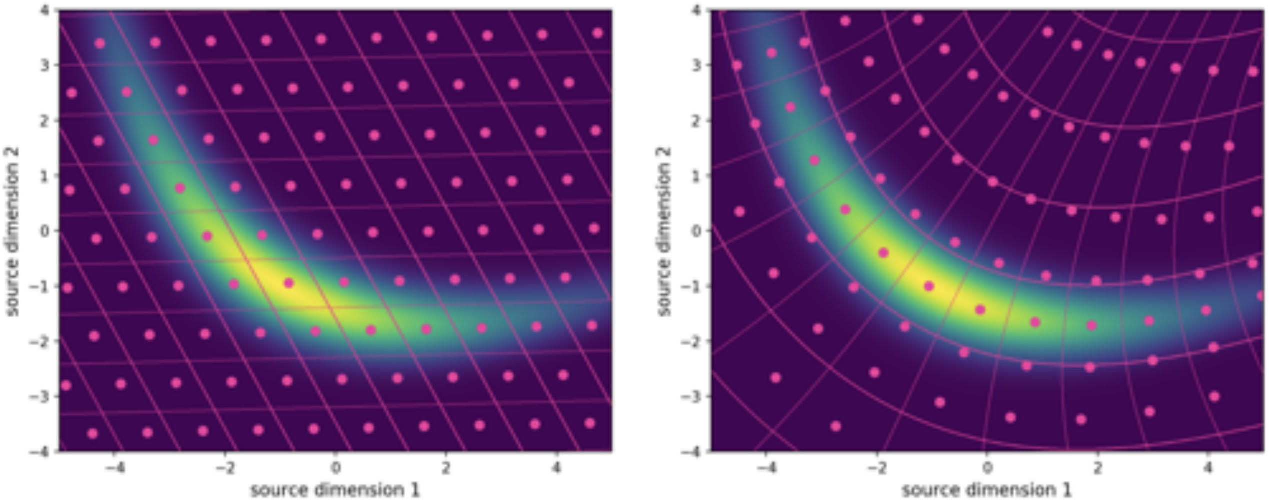


Figure 4: Comparison of LTC (left) and NTC (right) for a banana-shaped distribution [1]. Lines indicate quantization bins, with dots the codebook vectors

3.2 Neural Networks

Here, we give a brief overview of the mathematical formulation of ANNs and convolutional neural networks (CNNs), two architectures used for the analysis and synthesis transformations in this project.

3.2.1 Artificial Neural Networks

Modelled after how the human brain processes information, ANNs comprise multiple layers of parametric functions, where each layer is a mapping $f : \mathbb{R}^A \to \mathbb{R}^B$. Specifically, for input vector $\mathbf{u} \in \mathbb{R}^A$, weight matrix $\mathbf{W} \in \mathbb{R}^{\mathbb{B}\times\mathbb{A}}$, and bias vector $\mathbf{v} \in \mathbb{R}^B$,

$$f(\mathbf{u}) = \sigma(\mathbf{W}\mathbf{u} + \mathbf{v}) \text{ [1].} \tag{3.13}$$

Here, $\sigma : \mathbb{R} \to \mathbb{R}$ is a non-linear activation function applied to the output of the neuron to determine whether it is "on". Common examples include the Sigmoid function $\sigma(x) : \mathbb{R} \to (0, 1)$

$$\sigma(x) = \frac{1}{1 + e^{-x}} \tag{3.14}$$

and Rectified Linear Unit (ReLU) function $\sigma(x) : \mathbb{R} \to [0, \infty)$

$$\sigma(x) = \begin{cases} x & x \geq 0 \\ 0 & x < 0 \end{cases} \tag{3.15}$$

Of these, ReLU is particularly popular due to its low computational cost and simple implementation compared to the Sigmoid and other activation functions [21].

A neural network can have many layers, limited in theory by only computational power. The inner layers of the network are called *hidden layers*, the first layer the *input layer*, and the last layer the *output layer* [21]. Figure 5 shows a simple neural network architecture with two hidden layers.

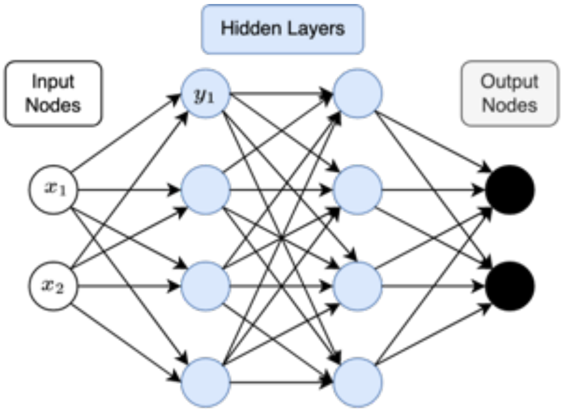


Figure 5: Example neural network architecture

The neural network defined in (3.13) and shown in Figure 5 is often called a *Multilayer Perceptron* (MLP), where information flows only in one direction [21]. In the context of this project, MLPs are useful for their ability to approximate any nonlinear function. In particular, the approximation capability grows with the depth of the network (number of layers) and number of nodes (dimensions of A and B) in a layer [21].

For a neural network $f(\mathbf{x}; \boldsymbol{\theta})$ with parameters $\boldsymbol{\theta} \in \mathbb{R}^s$, training attempts to minimize an objective cost function $C : \mathbb{R}^s \to \mathbb{R}$ with respect to the training parameters. A common choice for C is the function

$$C(\theta) = \mathbb{E}[L(f(\mathbf{x}; \boldsymbol{\theta}), \mathbf{y})] \tag{3.16}$$

which computes the expectation over all training samples of a loss function L that measures the difference between the neural network output and a ground truth value $\mathbf{y}$ [21]. As C is usually highly nonlinear and non-convex, it is infeasible to obtain an analytic global minimum. Instead, gradient descent is performed on C, with parameters updating according to

$$\boldsymbol{\theta}_{t+1} \leftarrow \boldsymbol{\theta}_t - \alpha \nabla C(\boldsymbol{\theta}_t) \tag{3.17}$$

for the learning-rate hyperparameter α and training step t [21]. For a sufficiently small α, this algorithm is guaranteed to find a local minimum of C. In practice, it is computationally impractical to compute the gradient over a large number of data samples. Instead, stochastic gradient descent (SGD) is performed by computing the gradient on a random subset of the data. The SGD algorithm used in this project is Adam, which updates the learning rate α throughout training, improving convergence stability under sparse gradients and noisy data [22].

3.2.2 Convolutional Neural Networks

Of particular interest to the application of image compression are CNNs, a specific type of ANN that includes layers that convolve the data $\mathbf{X}$ with a trainable kernel matrix $\mathbf{h}$ [23]. This creates a lower-dimensional feature map of the original data source that can be fed into dense neural network layers. CNNs have been shown to be effective at image segmentation and recognition while reducing the complexity and size of the network [23]. A summary of the key features of convolution layers is given below, while curious readers are encouraged to read *Saad et al.* for further details [24].

- *Translation Invariance*: The same filters are used across the entire image, meaning convolution layers can detect patterns regardless of their location.
- *Hierarchical representation*: Convolution layers are often stacked on top of each other, letting each successive layer learn more complex and abstract features of the image.
- *Spacial Locality*: Convolution layers only connect neurons to a local neighbourhood, reducing the number of parameters and making the network more computationally efficient.
- *Down-sampling*: CNNs include pooling layers which downsample the feature maps by taking the maximum (or average) across local regions.

A sample CNN architecture is shown in Figure 6. For image compression tasks, the analysis and synthesis transforms will be implemented using CNNs.

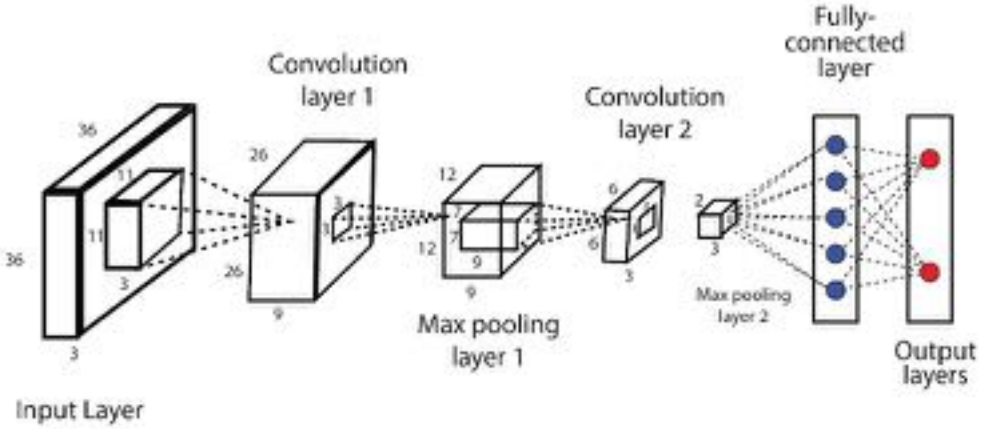


Figure 6: Example CNN architecture [25]

3.3 Previous Work

NTC is a highly active area of research, making it infeasible to conduct an exhaustive literature review. Instead, we discuss here select results that have motivated and guided our work. First, the paper *Nonlinear Transform Coding* by Ballé et al. reviews a large class of NTC methods, in particular deriving loss functions that replace quantization with additive uniform noise [1]. Their results using a randomized offset guide much of our work in creating a differentiable cost function in Section 4. In *Variational image compression with a scale hyperprior*, Ballé et al. introduce learned entropy models, an important extension of the primitive entropy model used in this project [26]. On the topic of activation functions, the paper *End-to-end optimization of nonlinear transform codes for perceptual quality* by Ballé, Laparra, and Simoncelli demonstrated the experimental effectiveness of the Generative Divisive Normalization (GDN) activation function for image compression [27]. In particular, they introduce an approximate inverse to GDN that forms the synthesis transform in NTC. Finally, in *Comparison of Full-Reference Image Quality Models for Optimization of Image Processing Systems* Ding et al. give a thorough comparison of different distortion metrics for image compression [28]. Of particular interest to this project is the discussion of perceptual distortion metrics, which have been shown to better reflect how humans perceive distortion in images. This work also discusses some of the implementation issues of perceptual distortion metrics, primarily stabilization issues including model convergence and sensitivity to initialization conditions. For a more complete review of NTC literature, *Nonlinear Transform Coding* includes a discussion of work in this field [1].

4 Solution

Here, we will give the mathematical formulation of the NTC system created in this project. This description will also include a discussion of design choices specific to this project. Recall that the aim of this project is to replace the analysis and synthesis transforms in LTC with ANNs, as shown in Figure 7.

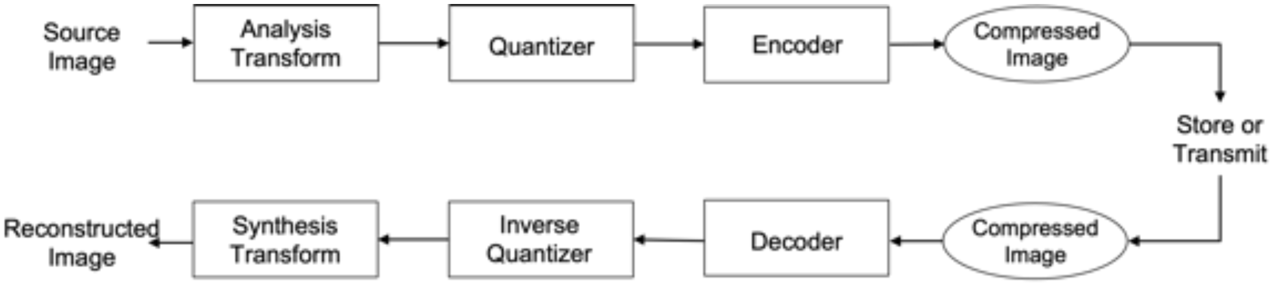


Figure 7: Nonlinear transform coding system outline

It is important to note that although a lossless encoder and decoder are included in this diagram for completeness, they are not a focus of this project and will not be implemented. Instead, entropy rates will be computed using the estimated bit rates of the quantized latent vectors. It is also assumed that the channel or storage system is lossless, which would not be true for practical applications.

4.1 Cost Function

We define the analysis transform and synthesis transforms as g_a and g_s respectively, with respect to the trainable parameter set $\boldsymbol{\theta}$. To optimize the analysis and synthesis neural networks, we minimize the cost function

$$L_{NTC} = \mathbb{E}_{\boldsymbol{x}}[-\log P(\lceil g_a(\boldsymbol{x})\rfloor) + \lambda\, d(\boldsymbol{x}, g_s(\lceil g_a(\boldsymbol{x})\rfloor))] \tag{4.1}$$

where $\lceil\cdot\rfloor$ is quantization to the nearest integer [1]. Here, $-\log P(\lceil g_a(\boldsymbol{x})\rfloor)$ represents the rate of the system in bits (i.e. $\log = \log_2$) for a probability distribution P and d the distortion between $\boldsymbol{x}$ and the reconstruction $g_s(\lceil g_a(\boldsymbol{x})\rfloor)$). The user-defined parameter λ is a Lagrange multiplier used for rate-distortion traversal. Specifically, as training jointly minimizes the rate and distortion, the trade-off between them can be predetermined by the choice of λ. For example, to achieve a lower distortion one can increase the value of λ, which would in turn increase the system rate.

An issue with the cost function given in (4.1) is that the quantization operation is discontinuous and cannot be differentiated. Thus, to perform gradient descent a proxy loss function must be introduced. In particular, by modelling the quantization step as an additive perturbation this problem can be avoided. Consider replacing the quantization step with a known offset value $\boldsymbol{o}$, and formulating (4.1) as

$$L_{NTC} = \mathbb{E}_{\boldsymbol{x},\boldsymbol{o}}[-\log P(\lceil g_a(\boldsymbol{x}) + \boldsymbol{o}\rfloor - \boldsymbol{o}; \boldsymbol{o}) + \lambda\, d(x, g_s(\lceil g_a(\boldsymbol{x}) + \boldsymbol{o}\rfloor - \mathbf{o}))]$$

$$(*) = \mathbb{E}_{\boldsymbol{x},\boldsymbol{o}}[-\log P(g_a(\boldsymbol{x}) + \boldsymbol{o}) + \lambda\, d(\boldsymbol{x}, g_s(g_a(\boldsymbol{x}) + \boldsymbol{o}))] \tag{4.2}$$

with $(*)$ following from [1]. It is clear that when the continuous offset is known, (4.2) can be minimized using SGD. A simple optimization protocol could then be to

1. Minimize (4.2) using offset $\mathbf{o}$;
2. Find the offset $\boldsymbol{o'}$ that minimizes L_{NTC} [1].

Note that if the continuous entropy model is sufficiently accurate, by translation invariance of differential entropy, $P(\boldsymbol{x};\boldsymbol{o}) = P(\boldsymbol{x}+\boldsymbol{o})$. Thus, the minimal offset $\boldsymbol{o'}$ could be determined without re-estimating the discrete entropy models [1]. It is still challenging to find $\boldsymbol{o'}$, as one cannot perform gradient descent on this parameter [1]. While we could find $\boldsymbol{o'}$ using a grid search over all possible offsets, in high dimensions this would be intractable. Instead, some papers have theorized that, motivated by a similar result for Laplacian distributions, choosing an offset to align the mode of the entropy model with a codebook vector of the quantizer is optimal [26] [29]. For a fixed-mode entropy model such as the Gaussian distribution, this suggests that without loss the offset can be fixed a priori. Based on this analysis and for simplicity of implementation, it was therefore assumed that the offset $\boldsymbol{o} \sim \text{Uniform}(\frac{-1}{2}, \frac{1}{2})$ was optimal. Specifically, for a latent vector of dimension N, we add uniform noise according to the random vector $\boldsymbol{u} \sim \text{Uniform}(\frac{-1}{2}, \frac{1}{2})^N$, creating the differentiable cost function

$$L_{NTC} = \mathbb{E}_{\boldsymbol{x},\boldsymbol{u}}[-\log P(g_a(\boldsymbol{x}) + \boldsymbol{u}) + \lambda\, d(\boldsymbol{x}, g_s(g_a(\boldsymbol{x}) + \boldsymbol{u}))]. \tag{4.3}$$

It was observed throughout training, and demonstrated by the results achieved, that this additive noise proxy sufficiently models the quantization process.

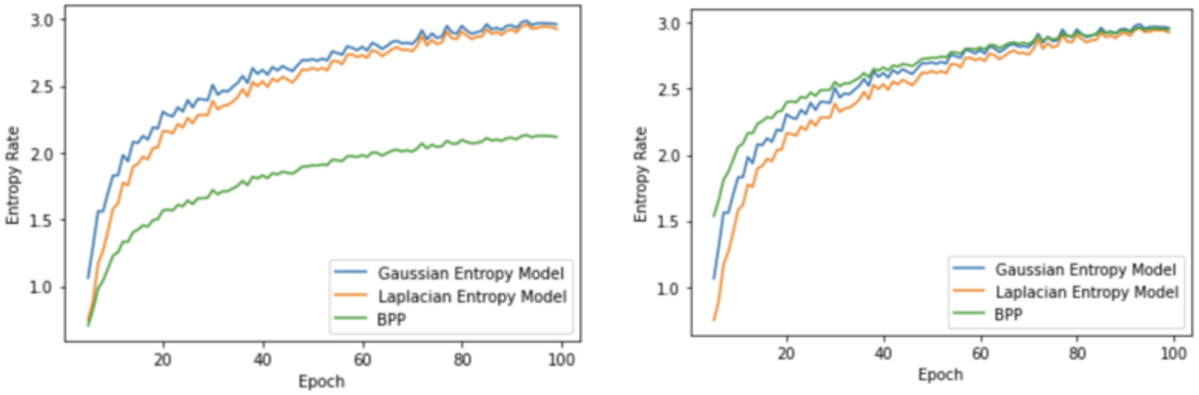
The choice of probability distribution for the entropy model is the next design choice that must be made. We select a continuous entropy model, noting that the addition of uniform noise has caused the latent vectors to be continuous-valued. For this, two common choices include the Normal(μ, σ^2) and Laplace(μ, b) distributions [1]. The Normal(μ, σ^2) has mean μ, variance σ^2, and probability density function (pdf)

$$f(x) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(\frac{-(x-\mu)^2}{2\sigma^2}\right), \tag{4.4}$$

while the Laplace(μ, b) distribution has mean μ, variance $2b^2$, and pdf

$$f(x) = \frac{1}{2b} \exp\left(\frac{-|x-\mu|}{b}\right). \tag{4.5}$$

To determine which entropy model best suited the chosen CT Scan dataset, the correlation between each entropy model and the entropy rate of the quantized latent vector was recorded over several architectures. A training sample is shown in Figure 8, with Figure 8a the original values and Figure 8b the values with a constant offset removed. In both figures, the strong connection between both entropy models and the estimated bits per pixel (bpp) can be seen. It was further observed that training converged similarly well for each entropy model. Ultimately, due to a slightly higher correlation and better training results on the chosen CT scan dataset, the Gaussian entropy model was selected moving forward.



(a) Original training

(b) Offset removed

Figure 8: Estimated entropy rates of images during model training

It is important to note that the quantized latent vector is assumed to be decorrelated and mutually independent, meaning for a latent vector $\boldsymbol{y} = [y_1, \ldots, y_N]$ the total entropy is computed as

$$\begin{aligned} -\log P(\boldsymbol{y}) &= -\log \prod_{i=1}^{N} P(y_i) \\ &= \sum_{i=1}^{N} -\log P(y_i) \end{aligned} \tag{4.6}$$

Using the Gaussian entropy model, the rate component of the loss for a single latent dimension with respect to mean and variance vectors $\boldsymbol{\mu} = [\mu_1, \ldots, \mu_N]$ and $\boldsymbol{\sigma} = [\sigma_1, \ldots, \sigma_N]$ becomes

$$\begin{aligned} -\log P(y_i) &= -\log \left(\frac{1}{\sqrt{2\pi\sigma_i^2}} \exp \left(\frac{-(y_i - \mu_i)^2}{2\sigma_i^2} \right) \right) \\ &= \frac{1}{2} \log(2\pi\sigma_i^2) + \frac{(y_i - \mu_i)^2}{2\sigma_i^2} \log(e). \end{aligned} \tag{4.7}$$

To fit the entropy model to the quantized data, the sample mean and variance are computed over the entire dataset, which are used in the computation of the entropy rate. The computation of $\boldsymbol{\mu}$ and $\boldsymbol{\sigma}$ is described in detail in Section 4.2. The distortion metric originally used was the squared error, meaning for an input image $\boldsymbol{x} = [x_1, \ldots, x_k]$ with the N latent dimensions each having mean $\boldsymbol{\mu} = [\mu_1, \ldots, \mu_N]$ and variance $\boldsymbol{\sigma} = [\sigma_1, \ldots, \sigma_N]$, the cost function becomes

$$L_{NTC} = \mathbb{E}_{\boldsymbol{x},\boldsymbol{u}} \left[\sum_{i=1}^{N} \left(\frac{1}{2} \log(2\pi\sigma_i^2) + \frac{\log(e)}{2\sigma_i^2} (g_a(\boldsymbol{x})_i + u_i - \mu_i)^2 \right) + \frac{\lambda}{k} \sum_{j=1}^{k} (x_k - g_s(g_a(\boldsymbol{x}) + \boldsymbol{u})_k)^2 \right]. \tag{4.8}$$

For a set of M training samples $\{\boldsymbol{x_1}, \ldots, \boldsymbol{x_M}\}$, SGD can then be applied to the training data cost function

$$L_{NTC} = \frac{1}{M} \sum_{m=1}^{M} \left[\sum_{i=1}^{N} \left(\frac{1}{2} \log(2\pi\sigma_i^2) + \frac{\log(e)}{2\sigma_i^2} (g_a(\boldsymbol{x_m})_i + u_i - \mu_i)^2 \right) \right.$$
$$\left. + \frac{\lambda}{k} \sum_{j=1}^{k} (\boldsymbol{x_{m_k}} - g_s(g_a(\boldsymbol{x_m}) + \boldsymbol{u})_k)^2 \right]. \tag{4.9}$$

Implementing (4.9) in Tensorflow caused numerical instability issues in practice, particularly in the computation of the rate for small σ_i. This problem was solved by applying a regularizing coefficient of $\frac{1}{N}$ was applied to the rate term, yielding the final cost function

$$L_{NTC} = \frac{1}{M} \sum_{m=1}^{M} \left[\frac{1}{N} \sum_{i=1}^{N} \left(\frac{1}{2} \log(2\pi\sigma_i^2) + \frac{\log(e)}{2\sigma_i^2} (g_a(\boldsymbol{x_m})_i + u_i - \mu_i)^2 \right) \right.$$
$$\left. + \frac{\lambda}{k} \sum_{j=1}^{k} (\boldsymbol{x_{m_k}} - g_s(g_a(\boldsymbol{x_m}) + \boldsymbol{u})_k)^2 \right]. \tag{4.10}$$

Note that this regularizing coefficient only changes the value of λ required to achieve a certain minimum and does not affect the overall training or rate-distortion performance of the system.

4.2 Training Algorithm

Algorithm 1 was used to optimize the parameters $\boldsymbol{\theta}$ of the neural network analysis and synthesis transforms. In particular, in each epoch the mean and variance of the quantized latent vectors are computed, which are used in the batch loss function given in (4.10). Algorithm 1 was implemented using a custom TensorFlow class, where the Adam optimizer was used to compute and apply gradients using an adaptive learning rate. The code used in the training routine is shown in Appendix 9.1. Note here that the optimization of the end-to-end system does not involve finding optimal quantizers. Instead, quantization is fixed to the nearest integer and the analysis and synthesis transforms are used to determine the effective quantization bins. In this way, the effective quantization bins can be shaped to best fit the learned data distribution. A more detailed discussion of this idea is given in Section 6.1.

Algorithm 1 System Training

```
procedure TRAIN(data, numEpochs, batchSize, $\lambda$, $\alpha$)                ▷ learning rate $\alpha$
    for epoch in numEpochs do
        $\boldsymbol{\mu} \leftarrow$ Mean($\lceil g_a$(data)$\rfloor$)          ▷ Compute mean and variance of quantized data
        $\boldsymbol{\sigma} \leftarrow$ Variance($\lceil g_a$(data$\rfloor$)
        for dataBatch in data do                                   ▷ dataBatch of size batchSize
            Cost $\leftarrow$ rate $+ \lambda *$ distortion                       ▷ Computed using (4.10)
            Gradients $\leftarrow \nabla_{\boldsymbol{\theta}}$(Cost)
            $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \alpha *$ Gradients
        end for
    end for
    return $\boldsymbol{\theta}$                                          ▷ Optimized neural network parameters
end procedure
```

4.3 Implementation Details

4.3.1 Tools and Technology

To implement the solution outlined above, Python was used with TensorFlow, a widely-used library for building deep neural networks [30]. Due to the significant computational demands of training such networks, the project relied on a GPU cluster provided by Queen's School of Computing. In particular, the project utilized a powerful NVIDIA A5000 GPU, which has 24GB of memory and was instrumental in the training process.

4.3.2 Dataset

To train and test the model, the project utilized a large-scale dataset of CT images publicly available through the National Institutes of Health (NIH) called DeepLesion [2]. This dataset contains 32,000 axial computed tomography (CT) slices from 10,594 CT scans of 4,427 unique patients. Each image is a 512x512 grid of grey-scaled pixels, meaning pixel values range from 0 (black) to 255 (white). Figure 9 provides an example of a typical CT scan that this project deals with.

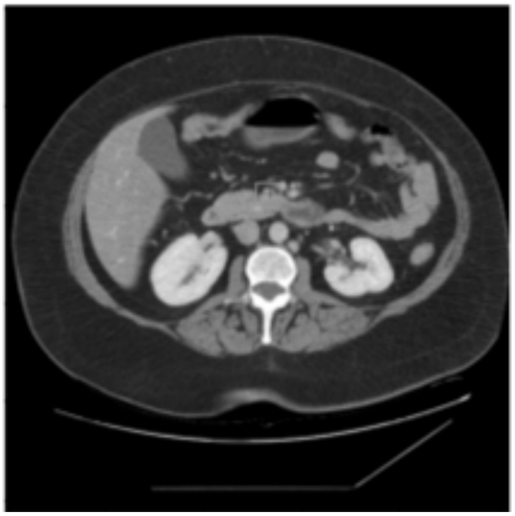


Figure 9: An example of a CT scan image from the DeepLesion dataset

5 Solution Iterations

A series of iterative improvements were carried out on the solution described in Section 4. This section outlines the enhancements made to improve the model's performance, while also describing the trade-offs associated with each decision.

5.1 Image Tiling for Computation Efficiency

A common obstacle in using neural networks for image processing is the computational impracticality of model training caused by the large size of the images. This is because the number of trainable parameters scales quadratically with the input dimensions. A widely-used technique in the literature to tackle this limitation is to partition the original image into tiles and train the model on these smaller tiles instead [31]. When using the model, an image is first sliced into tiles of predetermined dimensions, fed through the model, and the reconstructed tiles are then stitched back together.

This approach was utilized in our model design, as it was found that training on the full 512 × 512 CT scans would exceed the avaiable GPU memory. Each 512 × 512 image was partitioned into 64 64 × 64 tiles, as shown in Figure 10. These tiles were sufficiently small to train the model on the available hardware.

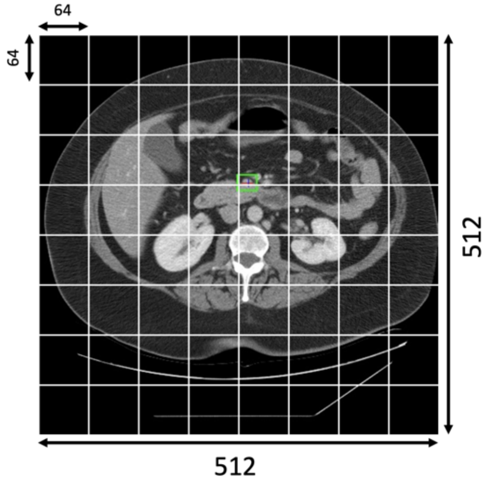
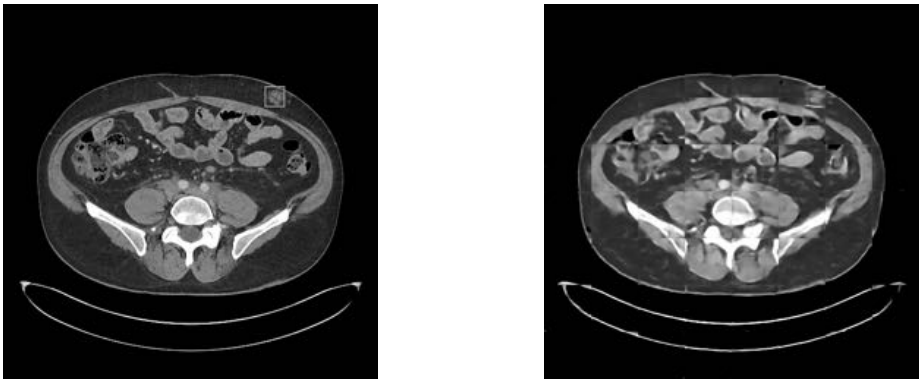


Figure 10: A 512x512 CT scan partioned into 64 × 64 tiles.

While the invariant properties of convolution neural networks enable the model to be trained on tiles rather than full images, there are trade-offs to consider. When deconstructing an image into tiles, the spatial correlations between pixels in different tiles are lost. This results in a visible "blocking effect" as pixels around the tile borders are incorrectly reconstructed because the model is unaware of pixels in adjacent tiles. In Figure 11b the borders of the tiles are visible in the reconstructed image along with the misalignment of features that span multiple tiles. Due to the limited availability of high-performance computing hardware, the team was forced to use image tiling despite the blocking effect. While not explored in this project, Section 7.3 discusses ways to reduce the blocking effect using Fourier transforms and overlapping tiles.



(a) Original

(b) Compressed and reconstructed

Figure 11: A blocking effect is visible in the reconstructed image after stitching together the tiles.

5.2 SSIM Distortion Measure for Improved Human Perception

MSE is a ubiquitous distortion measure in data compression and machine learning due to its ease of computation and desirable mathematical properties. Specifically, MSE is a continuous and differentiable function, making it convenient for optimization algorithms like gradient descent. However, when applied to image comparison, MSE is limited in its ability to represent the human visual system's perception of image quality and is particularly sensitive to noise and image artifacts [32]. Aiming to address these limitations, the structural similarity index measure (SSIM) is a distortion measure that more closely resembles human perception [33]. It addresses the shortcomings of MSE's ability to capture local spacial structures within images and considers an image's luminance and contrast.

Mathematically, SSIM : $(x, \hat{x}) \to [-1, 1]$ is a measure between an image x and its reconstruction $\hat{x}$ comprised of a weighted product of comparison measurements,

$$\text{SSIM}(x, \hat{x}) = l(x, \hat{x})^\alpha \cdot c(x, \hat{x})^\beta \cdot s(x, y)^\gamma. \tag{5.1}$$

Here, $l(\cdot)$, $c(\cdot)$, and $s(\cdot)$ are luminance, contrast, and structure measures defined by

$$l(x, \hat{x}) = \frac{2\mu_x \mu_{\hat{x}} + c_1}{\mu_x^2 + \mu_{\hat{x}}^2 + c_1} \qquad c(x, \hat{x}) = \frac{2\sigma_x \sigma_{\hat{x}} + c_2}{\sigma_x^2 + \sigma_{\hat{x}}^2 + c_2} \qquad s(x, \hat{x}) = \frac{\sigma_{x\hat{x}} + c_3}{\sigma_x \sigma_{\hat{x}} + c_3}$$

where μ_i, σ_i^2, σ_{ij} are sample means, variances, and covariances, respectively, and c_i are variables to stabilize the division with weak denominators [34]. An SSIM value of 1 indicates that the two images are identical while a value of -1 indicates that the two images are completely dissimilar (in practice SSIM values usually fall between 0 and 1). This distortion measure is differentiable, meaning (4.10) can be adapted to use SSIM as the distortion.

The improvements of using SSIM over MSE in human perception can be seen in Figure 12. It is clear that despite both having MSE of 255, the image on the right is more similar to the original image. Opposed to MSE, SSIM is able to correctly capture the difference as can be seen by the higher score of the image on the right in distortions that humans perceive.

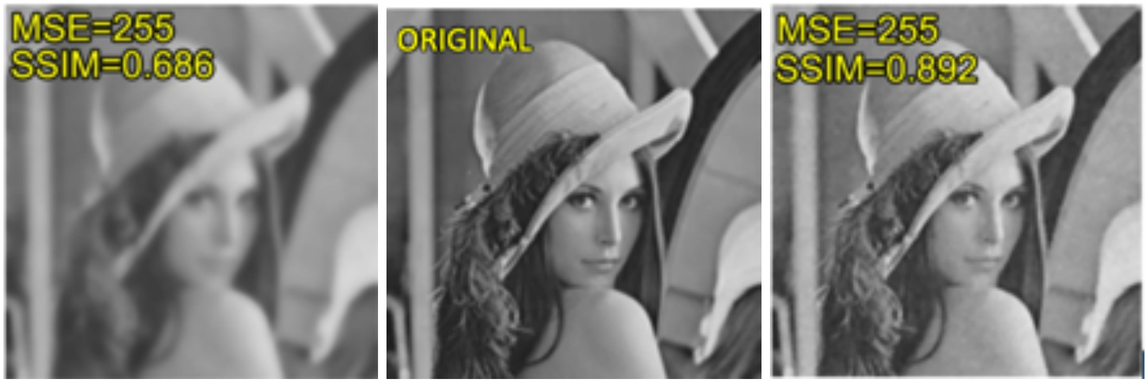


Figure 12: Comparison of SSIM and MSE for image distortion [33].

There are important drawbacks to consider when using SSIM as a distortion measure. The SSIM calculation requires the computation of the sample's mean, variance, and covariance. This is computationally expensive, increasing the training time of the neural networks. Both MSE and SSIM were tested in the loss function and while training took longer using SSIM, the improvements to the reconstruction quality using SSIM outweighed the additional training time. Further, the complexity of SSIM can lead to numeric instability in network training. To circumvent this issue, MSE was used in the initial epochs where the instability is most significant, as the model parameters are randomly initialized and the network has not yet learned to predict accurate outputs. After the initial epochs, SSIM was then used as a more informative distortion measure.

5.3 Regularization Techniques to Address Overfitting

A common problem in machine learning is overfitting. Overfitting is when a model becomes overly complex and fits the training data too closely, leading to poor performance on new, unseen data. In the initial iterations of the model, overfitting was present during training, as can be seen in Figure 13a. The SSIM score on the test data (unseen data) stops improving at the 30th epoch while the score for the training data continues to increase. This indicates that the model is no longer generalizing well and is instead "memorizing" the noise and idiosyncrasies present in the training data.

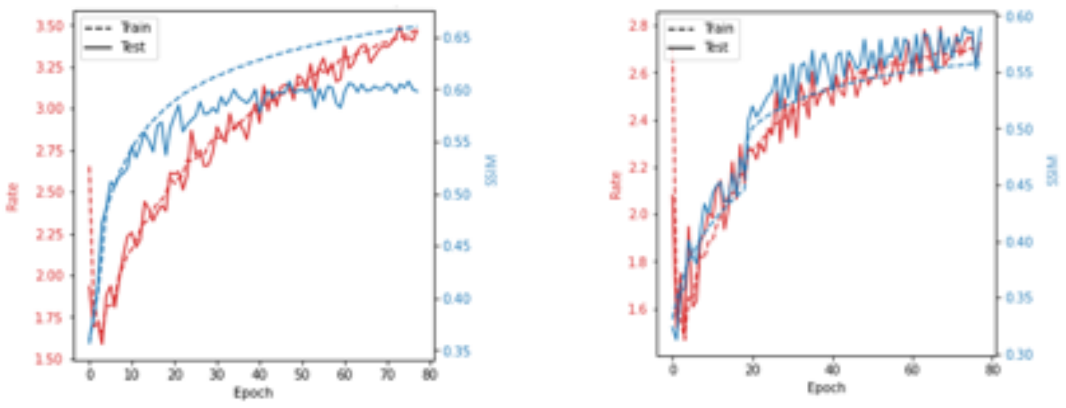
To address overfitting, several regularization techniques commonly used in image processing neural networks were tested. The successful techniques are summarized below:

- **Dropout Layers**: During training, a sample of neurons are temporarily "dropped out" with a preset probability. The remaining neurons are then trained and their parameters updated. In each training iteration, different neurons are dropped from training. This forces the model to learn more generalizable representations of the data rather than memorize specific features in the training data.
- **Weight Decay**: A penalty term was added to the loss function that penalizes large weight values using the L_2 norm,

  $$\lambda \sum_{i=1}^{N} |\beta_i|^2$$

  where β_i are the network weights and λ is a hyper-parameter that dictates how strongly the regularization is enforced. This penalty encourages the model to have small weight values, helping prevent the model from memorizing idiosyncrasies in the training data that aren't present in the population.

- **Diverse Training Set**: A diverse training set ensures that the model performs well on all types of data in the domain. As described in Section 4.3.2, the CT scan dataset contains 32,000 images. After partitioning each image into 64 tiles, there are 2.24 million images for training. Since it is infeasible to train on this much data, a subset was used for the training data. To generate this training dataset, 1,000 images were randomly sampled from each of the 64 tiles to ensure a wide range of CT scans, and an equal proportion of tile locations are contained in the training data.



(a) Before Regularization

(b) After Regularization

Figure 13: Plots of the Rate and SSIM during model training before and after implementing regularization techniques to address overfitting.

After implementing the above regularization techniques, the generalization capabilities of the model vastly improved, as is evident in Figure 13b. Here, the test SSIM score no longer plateaus and instead continues to improve with the training data SSIM. While these techniques helped address overfitting, they resulted in the model taking more training iterations to converge. This is because the dropout layers add stochasticity to the training process and the weight decay adds additional compute cost to the loss function.

5.4 GDN Activation Function for Improved Image Processing

The activation function takes an input signal from a network layer and produces an output signal to be fed into the next layer. As described in Section 3.2.1, ReLU is commonly used due to its computational efficiency and ease of implementation. However, analogous to MSE underperforming in Section 5.2, the simplicity of ReLU results in a loss of statistical properties that are needed to reconstruct the image.

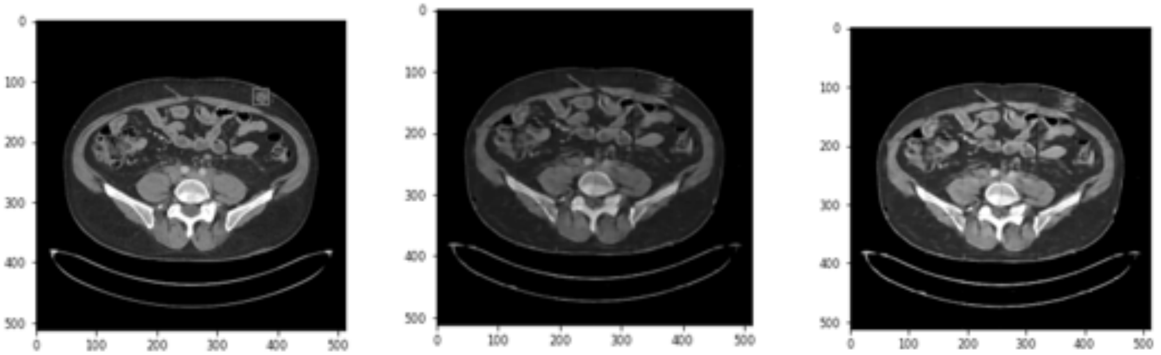
GDN is an alternative activation function commonly used in image compression neural network architectures with a (simplified) equation given by,

$$\mathrm{GDN}(x) = \frac{x}{\sqrt{\sum_{i=1}^{N} \gamma_i x_i^2 + c}} \text{ [35].} \tag{5.2}$$

GDN improves the stability and robustness of the network by normalizing outputs to have zero mean and unit variance. Further, GDN preserves statistical properties of the input data by normalizing each element by the sum of its neighbouring elements [35]. Of particular importance to this project, *Ballé et al.* has shown that GDN has an inverse transform which we use in the synthesis network to reconstruct the image [27]. The downside of GDN is that it is significantly more computationally expensive than ReLU, which slows down the network training.

5.5 Contrast Intensity Rescaling for Enhanced Image Quality

An issue observed while designing and training the compression model was that the reconstructed images had lower contrast than the original, as can be seen in Figure 14. Given the “black box” nature of neural networks, it can be difficult to determine the exact cause of this. After conducting a literature review, several potential culprits have been identified. First, activation functions that have a finite output range can result in vanishing gradients if the inputs are very positive or very negative; as the output becomes “stuck” at the function’s upper or lower limit, resulting in very small gradients [36]. Thus, it is possible that the ReLU activation function in some of the hidden layers were mapping a significant amount of negative values to zero, resulting in a loss of information (this is known as the dying ReLU problem) [36]. Another explanation could be related to the neural network architecture, as improper layer designs and parameter selection can also result in a loss of contrast.



(a) Original

(b) Reconstructed

(c) Rescaled

Figure 14: A reconstructed image with low contrast is rescaled to more closely resemble the original image.

As the team was unable to identify the underlying problem, intensity rescaling was used to match the reconstructed image’s contrast to the original image. Let I_{ij} be the intensity value of a pixel located at position (i, j) in the reconstructed image. Further, let p_1 and p_{99} represent the 1st and 99th percentiles of the intensity distribution of the original image (determined using a sample histogram). Then, the reconstructed image can be rescaled to match the original distribution using

$$I_{ij}^{\text{rescaled}} = \frac{I_{ij} - I_{\min}}{I_{\max} - I_{\min}} \cdot (p_{99} - p_1) + p_1, \tag{5.3}$$

where $I_{\max}$ and $I_{\min}$ are the largest and smallest intensity values, respectively, in the reconstructed image. After applying this transformation to the reconstructed image, it is clear that the contrast of Figure 14c matches that of the original image more closely. In practice, intensity rescaling could be implemented by allocating p_1 and p_{99} as side information stored with the compressed image. Then, reconstruction could use these values to perform the rescaling according to (5.3). The storage size of these values is negligible compared to the images and therefore are omitted in compression rate calculations.

5.6 Final Model Design and Network Architecture

After implementing the changes described above, the final model design is summarized in Figure 15. The analysis and synthesis transforms are implemented as neural networks whose architecture are summarized in Figure 15. For full details on the network implementation, such as specific layer parameters, see Appendix 9.4.

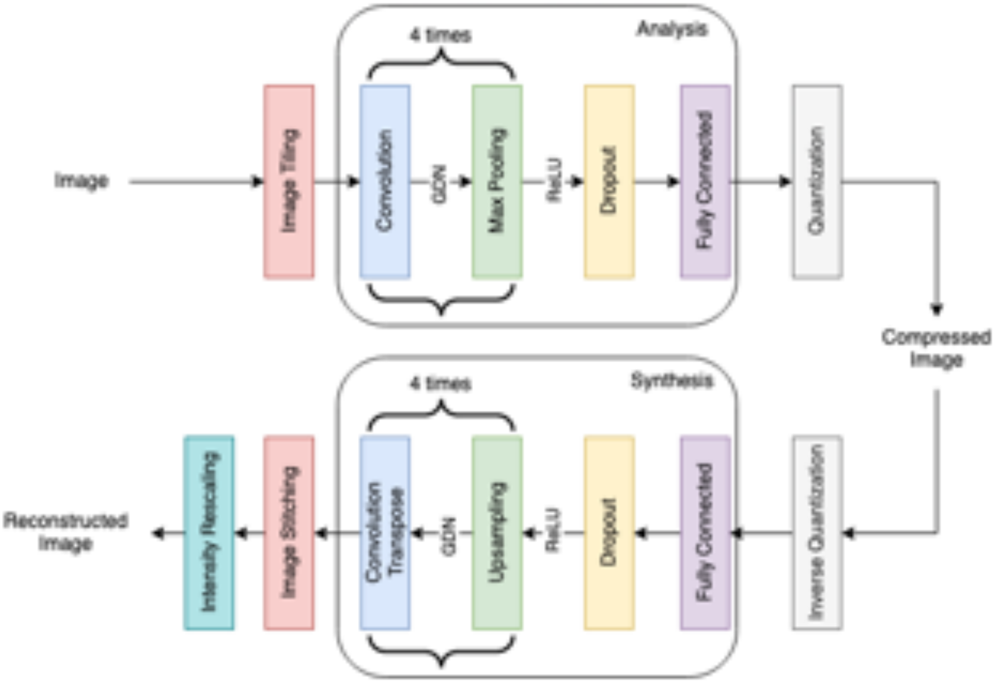


Figure 15: Finalized end-to-end image compression workflow.

As described throughout the above sections, each iteration to the model incurred trade-offs to consider. Table 13 summarizes these trade-offs.

Table 3: Summary of the model design iterations and their tradeoffs.

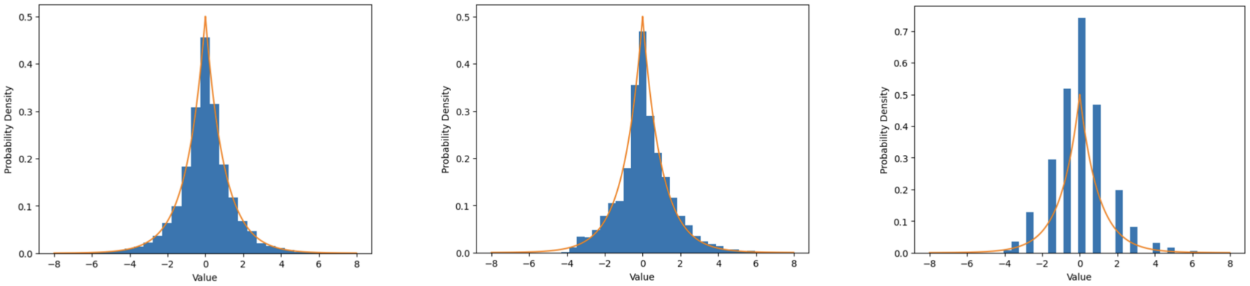
Iteration	Improvement	Drawback
Image Tiling	Made training computationally feasible	Blocking artifacts in reconstructed image
SSIM Distortion Measure	Improved reconstruction quality	Numerical instability and complexity
Overfitting Regularization	Improved model generalization	Increased computational complexity
GDN Activation Function	Improved reconstruction quality	Increased computational complexity
Contrast Intensity Rescaling	Improved reconstructed contrast	Does not address underlying problem

6 Testing and Results

Here, we describe the testing of the system and results achieved. We also give commentary on the limitations of this system, particularly in comparison to industry standards such as JPEG.

6.1 One-Dimensional Laplacian

The system was tested on a toy problem to confirm correct performance, specifically that it is capable of learning and compressing data drawn from a simple source distribution. We generated 10,000 training points according to the Laplace$(0, 1)$ distribution given in (4.5). Then, using the model architecture shown in Appendix 9.2 the model was trained for 20 epochs using a batch size of 128 and $\lambda = 150$. The results of the model on 10,000 test points can be seen in Figure 16. Note that the reconstruction using the uniform noise perturbations in Figure 16b almost exactly matches the original distribution in Figure 16a. This is evidence of the system's ability to learn the input distribution and generalize to unseen data points. The loss from the approximation of quantization through additive uniform noise can be seen in the difference between Figure 16b and 16c. The quantized reproduction, although it maintains the shape of the distribution, is slightly overfit around zero and not as exact of a reconstruction. Still, the similiarity of Figure 16a to 16c supports approximating quantization as uniform noise, as the quantized system still learns the underlying data distribution.



(a) Original Distribution

(b) Uniform Noise Reconstruction

(c) Quantized Reconstruction

Figure 16: Histograms of the Laplacian distribution and model reconstructions

As this is a one-dimensional example, the analysis and synthesis transforms can be visualized, shown in Figure 17. It is interesting to observe that although no restriction was imposed that the transformations be inverses, the analysis and synthesis transforms have naturally converged to be near inverses of each other. This matches our intuitive understanding that the nonlinear analysis and synthesis transforms replace the orthogonal transforms $\mathbf{A}$ and $\mathbf{A^{-1}}$ from LTC. Figure 17 also shows the effective quantization regions and codebook vectors in solid and dotted lines, respectively. It can be seen that the codebook vectors are concentrated close to the high-density area of the data around zero, as would be expected for an optimal quantizer. This shows that the quantization can be fixed a priori, allowing the neural networks to produce an optimal effective quantizer by learning the data distribution.

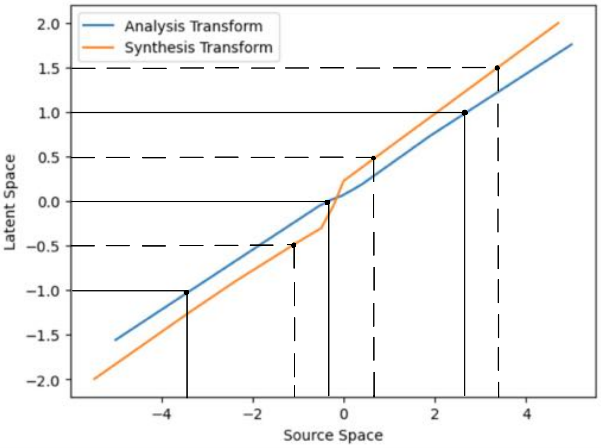


Figure 17: Analysis and synthesis transformations visualized in two dimensions. Dotted lines indicate the codebook vectors, while solid lines indicate the quantization region boundaries.

6.2 Handwritten Images

The NTC system was next tested on two-dimensional images, specifically the database of handwritten images provided by Modified National Institute of Standards and Technology (MNIST). MNIST consists of 70,000 grayscale images, each 28×28 pixels and 8 bpp. This testing aimed to determine if the NTC training could be applied to CNNs and compare different architectures and activation functions using a more computationally efficient application. Specifically, we wanted to compare the efficacy of the ReLU and GDN activation functions for our system. Figure 18 shows the rate-distortion functions for identical analysis and synthesis architectures, differing only in the activation function used. Appendix 9.3 details the exact architecture used, which for each value of λ was trained on 200 epochs using 10,000 training images and a batch size of 32. The Peak Signal-to-Noise Ratio (PSNR) plotted on the y-axis measures the distortion using

$$\text{PSNR} = 10 \log_{10} \left(\frac{255^2}{\text{MSE}} \right). \tag{6.1}$$

The bpp was estimated using an approximation of the entropy rate of the quantized latent vector. For test data $\boldsymbol{x} = \{\boldsymbol{x}_1, \dots, \boldsymbol{x}_M\}$ and test quantized latent vectors $\boldsymbol{y} = \{\boldsymbol{y}_1, \dots, \boldsymbol{y}_M\} = \{\lceil g_a(\boldsymbol{x}_1) \rfloor, \dots, \lceil g_a(\boldsymbol{x}_M) \rfloor\}$ with $\boldsymbol{y}_i = [y_{i,1}, \dots, y_{i,N}]$ (for N scalar quantizers), we note that the assumed independence of each quantizer output allows us to compute the entropy of each dimension individually. Specifically, we use a method similar to Monte Carlo approximations to compute

$$P(y_{i,j}) = \frac{1}{M} \sum_{k=1}^{M} \mathbb{1}_{\{y_{i,j} = y_{k,j}\}}, \tag{6.2}$$

counting the number of appearances of that integer in the test set and normalizing by the number of test points. Then, the entropy of a test point $\boldsymbol{y}_i$ can be estimated in bits as

$$H(\boldsymbol{y}_i) = H\Big([y_{i,1}, \dots, y_{i,N}] \Big) = -\sum_{k=1}^{N} P(y_{k,j}) \log_2 P(y_{k,j}) \tag{6.3}$$

This entropy can then be used to determine the bpp of the image. It is important to note that although this is an estimate of the entropy, the assumption of independence and lack of lossless entropy coding means it is also an upper limit. In practice, using a lossless entropy code would yield lower entropy rates than those reported in this thesis.

Referring to Figure 18, it is clear that the GDN activation function outperforms ReLU using the metric of PSNR. This matches GDN's theoretical superiority discussed previously in Section 5.4. However, this testing also confirmed the increased computational complexity incurred through GDN activation functions, as an epoch of training using GDN took 10 seconds, while an epoch for the ReLU architecture took only 5 seconds (running on a 24 GB NVIDIA RTX a5000 GPU). Despite the longer training time, the requirement of high reconstruction quality in the compression of CT scans means GDN will be used as the activation function in the final testing.

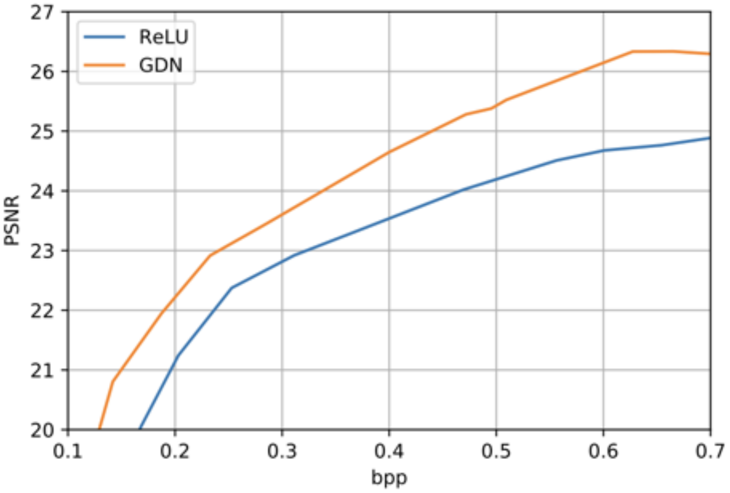
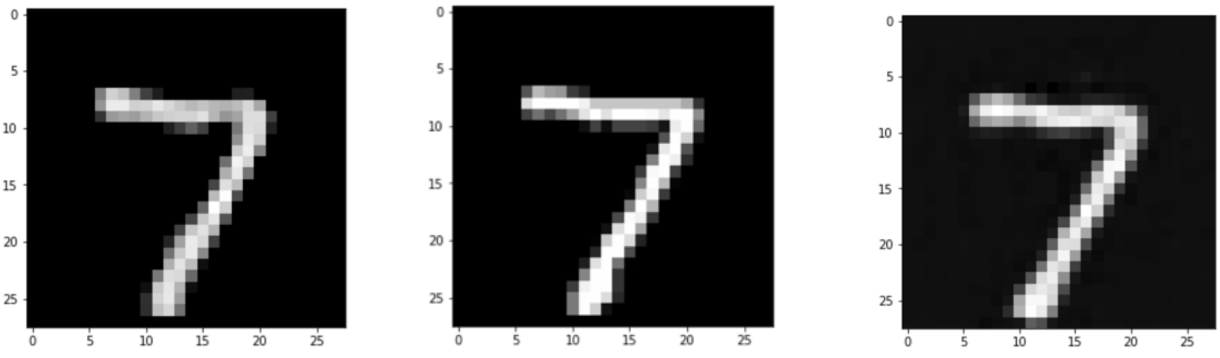


Figure 18: MNIST Rate Distortion Graph of ReLU and GDN activation functions

Figure 19 shows an original image from the MNIST test set and the reconstructed versions using the ReLU or GDN activation functions. Perceptually, there is little difference between the two reconstructions, although the low number of pixels in the original image makes it difficult to assess the overall quality.



(a) ReLU MNIST (0.194 bpp)

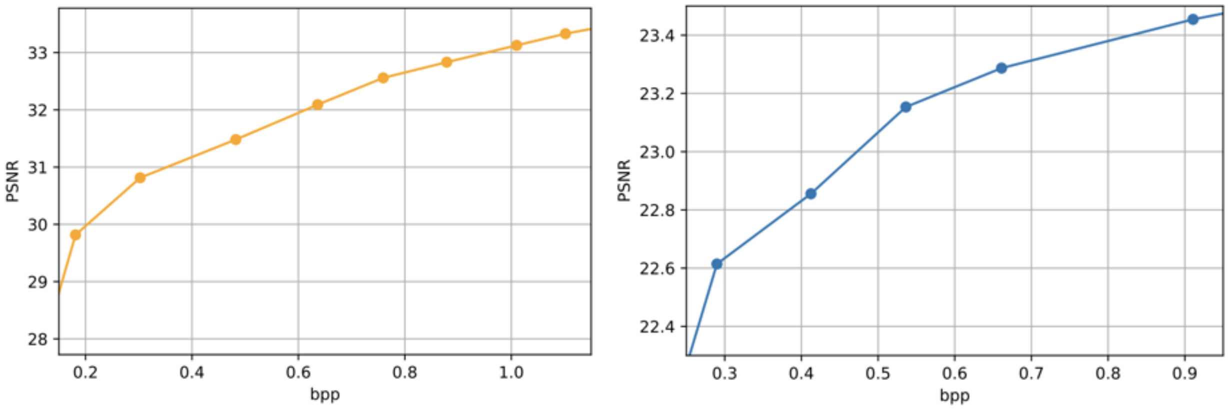
(b) Original MNIST

(c) GDN MNIST (0.179 bpp)

Figure 19: Compressed and original MNIST images from test set

6.3 CT Scans

Using the segmented dataset of 64,000 images described in Section 5.3, for each λ the model was trained for 100 epochs with a Gaussian entropy model and additive uniform noise. The exact architecture of the analysis and synthesis transforms is shown in 9.4. As discussed in Section 5.2, the poor visual quality of reconstructed images motivated the team to investigate the SSIM as an alternative distortion metric. Due to the importance of visual quality in the application of CT scan compression, the SSIM was used in the testing detailed here. Figure 20 shows the rate-distortion function of the CT scan dataset, using the NTC model and JPEG standard as a baseline comparison. It is evident that JPEG outperforms NTC in this setting, achieving a noticeably higher PSNR, corresponding to a lower MSE. However, there are several factors affecting the PSNR of our NTC model. First, the neural networks are optimized for SSIM, meaning that the PSNR is a consequence of this training. It can be reasonably assumed that training the system specifically for MSE would lead to a higher PSNR, although at the cost of perceptual distortion. The lack of lossless entropy encoding, as previously discussed, also leads to a higher estimated bpp than in reality. Finally, there are several significant improvements to the system (discussed in Section 7) that we theorize would cause the results to surpass JPEG, as it has for similar NTC systems [1].



(a) JPEG (LTC)

(b) NTC

Figure 20: Rate distortion graphs for the compression of CT scans

Figure 21 shows a sample CT scan from the test set of full images that has been compressed to 0.77 bpp using JPEG and NTC. The contrast intensity of the NTC reconstruction has been rescaled according to the process described in Section 5.5. In the NTC reconstruction, blocking artifacts can be seen from the image tiling performed in Section 5.1. However, the perceptual similarity to the original image is quite high, with most of the image details preserved despite the high compression rate. The compression and reconstruction time was also computed for the final model to understand if this system could be practically deployable with little to no delay. Running the entire model on 1000 full test images, the entire sequence of tiling, compression, reconstruction, and contrast intensity rescaling took an average of 0.03059 seconds, for a total time of 30.59 seconds. This testing was performed on a Macbook Pro laptop without GPU accelerators, demonstrating that the system can be run on common hardware.

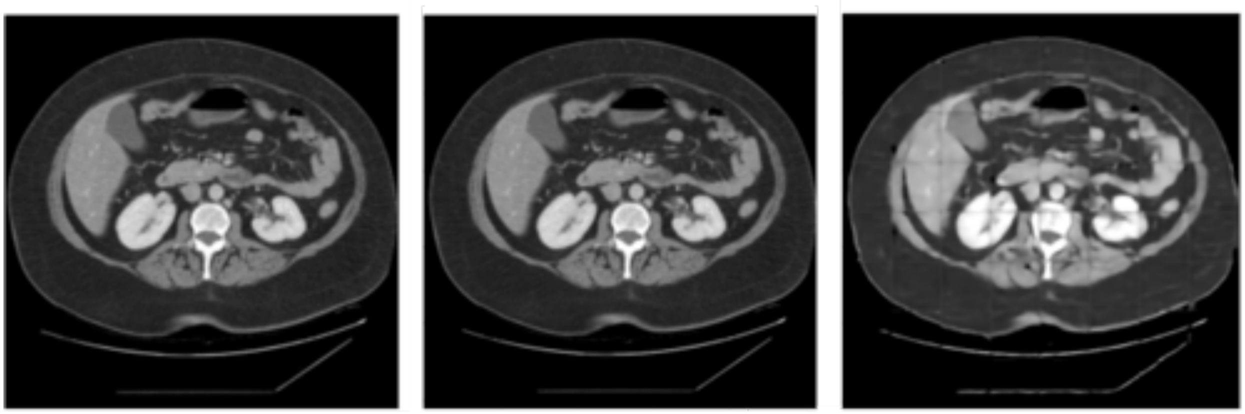


Figure 21: Original (middle) and reconstructed (left: JPEG, right: NTC) images at 0.77 bpp

6.4 Results Discussion

The results described clearly indicate the ability of the NTC system to learn and compress from an arbitrary distribution. Further, with the optimizations specific to CT scan compression, promising results were achieved. Without the blocking effect, the qualitative reproduction quality is quite high, with limited loss of detail and preservation of the core information. It also is sufficiently fast and simple to be practically implemented without the need for advanced computing. Although it falls short of the JPEG standard, based on the time available and the steep learning curve associated with this project, these results can be reasonably declared a success.

7 Future Work

While the group was overall successful in achieving the scope of our project, there are several limitations that could be addressed in future work. Specifically, future work includes refining the entropy model, introducing entropy coding, and overall improvement of model implementation.

7.1 Improve Entropy Model

A key limitation was the primitive Gaussian entropy model used in model training, which could be improved by introducing a Gaussian mixture model. A Gaussian mixture model p is the finite weighted sum of individual Gaussian distributions given by

$$p(x) = \sum_{i=1}^{N} w_i g_i(x) \tag{7.1}$$

for a Gaussian distribution g_i with mean μ_i and variance σ_i and mixture weighting w_i such that $\sum_{i=1}^{N} w_i = 1$ [37]. Mixture models can fit more general data and would likely give a better approximation of the entropy. However, this entropy model would also require careful consideration of how to best fit it to a set of latent output vectors in a computationally efficient way. The current implementation is simple, which is beneficial for implementation, but as was discussed in Section 4 is not a perfect model of the actual entropy rate. Introducing mixture distributions would improve the system by accurately converging to a rate minimum during optimization.

The model could also be improved by introducing models that do not assume the decorrelation of latent vectors. Decorrelation is not a practical assumption in most real-world situations, nor is it optimal in general, meaning designing a system that can incorporate and leverage the correlation between latent variables would be an important next step [38]. Previous work on NTC has had success using learned entropy models to approach this problem [1]. In particular, learned entropy models use forward and backward adaption to incorporate ANN-based entropy models that can be jointly optimized alongside the transforms [1]. This technique has seen considerable success, notably being shown to outperform JPEG and other NTC models on image compression in [1].

7.2 Introduce Entropy Coding

Another limitation of the current model is that it estimates the final entropy rate instead of implementing entropy coding. Entropy coding is a lossless data compression technique that encodes data into a binary string. In the compression pipeline seen in Figure 7, entropy coding would be performed on the quantizer output to create a binary string used for storage or transmission. The number of bits in the string would be directly used to compute the bpp of the image, which in the current implementation is an estimate.

At a high level, an optimal entropy code aims to minimize the entropy of the data by assigning shorter codes to more frequently occurring symbols and longer codes to less frequently occurring symbols. Two of the most commonly used entropy encoding methods are Huffman and Arithmetic coding. Huffman coding constructs a binary tree where each leaf node represents a symbol in the data, and the path from the root to the leaf node represents the code assigned to that symbol [39]. The algorithm starts by creating a set of leaf nodes, one for each symbol in the data. It then repeatedly combines the two nodes with the lowest frequency of occurrence, creating a new internal node whose frequency is the sum of the frequencies of its children. This process continues until there is only one node left, which represents the root of the binary tree. The codes are assigned by traversing the binary tree from the root to each leaf, with a 0 assigned to each left branch and a 1 assigned to each right branch [39]. The data is then encoded by assigning each symbol its respective code.

Arithmetic coding is another method that assigns a fractional value to each symbol in the data based on its probability of occurrence. The algorithm divides the range [0,1) into sub-intervals, with the size of each sub-interval being proportional to the probability of the symbol it represents [39]. The encoded value for the entire input source is the midpoint of the sub-interval corresponding to the entire data stream [39]. To decode the data, the decoder uses the same probability distribution to reconstruct the original intervals and determine the symbol corresponding to each [39].

Both Huffman coding and arithmetic coding are optimal and uniquely decodable algorithms. Of these, arithmetic coding would be implemented, as it can be decoded sequentially and does not require storage of a binary tree that grows exponentially in size with the number of code words [39].

7.3 Improvements to Model Implementation

While the team was satisfied with the model's performance, there were some limitations as described in the results in Section 6. The following list contains areas of improvement for the model implementation.

- **Reducing the Blocking Effect**: In our current implementation, the tiles partition the image without any overlap. This results in a blocking effect caused by a loss of information of nearby pixels around tile borders. *Tang et al.* have proposed a patch-based overlapping tile strategy to mitigate this effect by blending the over-lapping regions after processing [40]. Moreover, using Fourier transforms to "smooth" the blocking effect in the frequency domain is another strategy worth investigating. Lastly, sufficient computing resources could circumvent the need for image tiling altogether.

- **Improving Reconstruction Contrast**: As described in Section 5.5, the reconstructed images had a lack of contrast. Due to time constraints, the team was unable to identify the source of the problem and instead used intensity re-scaling as a temporary measure. The next step would thus be to analyze the network architecture to identify the source of the problem. For instance, adjustments to the convolution layer parameters may be necessary, or the dying ReLU problem may be present.

- **Exploring Other Network Architectures**: A CNN architecture was used in this project; however, other network architectures are worth exploring. For instance, *Toderici et al.* implemented an image compression network using a Recurrent Neural Network (RNN) architecture [41]. There is also literature on the use of Transformer architectures in computer vision tasks, which may be applicable to image compression [42].

8 Conclusion

With the current world increasingly reliant on multi-media communication, the demand for the effective compression of data grows each day. The development of ANNs and computational resources has motivated a potential shift from LTC to NTC for image compression. The application of this thesis was to develop a custom NTC model for the compression of high-quality CT scan images for the detection of cancer lesions. Such an application is vital to the patients, practitioners, and healthcare system identified as stakeholders and analyzed in Section 2.2.

The results of the project indicate that a custom NTC model is capable of complex image compression while meeting the needs of all stakeholders. Through the incorporation of entropy modelling and additive uniform noise, a custom optimizable system was created. By implementing the GDN activation function and SSIM distortion, this system was trained to compress CT scans. The results were compared to JPEG compression, and although it did not surpass this standard there are clear avenues for future work that would improve the system. Importantly, it was also demonstrated that this system can be run on conventional hardware for fast compression and reconstruction. The results achieved demonstrate the viability of this method and motivate future work regarding the compression of CT scans using NTC. Specifically, exploring topics such as improved entropy models, lossless entropy coding, and architecture refinements have been shown in similar systems to yield significant improvements. It is expected that with further work, NTC will surpass JPEG and can be implemented as the method of compression in this and many other applications.

9 Appendix

9.1 Nonlinear Transform Coding Class

Shown here is the CustomCompression class used to implement Nonlinear Transform Coding with neural network analysis and synthesis transforms. Note that this version has been significantly reduced from the actual code to preserve only the core functionality of training. In particular, code to record the training loss history and compute the loss on a test dataset has not been included, though it was used in practice. Code implementing the Laplacian entropy model and SSIM distortion are also not given in this version.

```
class CustomCompression(Model):
"""Custom Compression class inheriting from tf.keras.Model. Implements Nonlinear
    Transform Coding using neural network analysis and synthesis transforms.
    alpha : lambda parameter used for lagrangian RD traversal
    analysis : analysis transform; tensorflow.keras.Sequential model
    synthesis : synthesis transform; tensorflow.keras.Sequential model"""
def __init__(self , alpha, analysis, synthesis):
    super(CustomCompression, self).__init__()
    self.analysis = analysis
    self.synthesis = synthesis
    self.alpha = alpha
    self.var = [] # Class attributes to hold latent variance and mean
    self.mu = []

def call(self, inputs, training=True): # Overwritten call function, usage: model(data)
    x = self.analysis(inputs)
    if training:    # Use additive uniform noise when training
      x = self.add_noise(x)
    else:   # Quantize during testing
      x = self.quantize(x)
    return self.synthesis(x)

def gaussian(self, latent): # Gaussian entropy model
    result = 1/2*tf.experimental.numpy.log2(2 * 3.1415926535 * self.var) +
            tf.math.pow((latent-tf.cast(self.mu, dtype=tf.float32)), 2)/(2 * self.var)*
            np.log2(2.718)
    return result

def add_noise(self, latent): # Add uniform noise to each dimension
    return latent + tf.random.uniform(tf.shape(latent), -1/2, 1/2)

def quantize(self, latent): # Quantize to the nearest integer
    return tf.math.floor(latent + 1/2)

def compression_loss(self, input, output):  # Compute loss on a training batch
    distortion = tf.reduce_mean(tf.square(input - output))
    latent = self.analysis(input)
    rate = tf.reduce_mean(self.gaussian(self.add_noise(latent)))
    return rate, distortion

def train_step(self, x): # Compute loss of the training data x
    with tf.GradientTape() as tape:
        y = self(x, training=True)
```

```
            rate, distortion = self.compression_loss(x, y)
            loss = rate + self.alpha * distortion
        # Compute and apply gradients to the trainable parameters
        gradients = tape.gradient(loss, self.trainable_variables)
        self.optimizer.apply_gradients(zip(gradients, self.trainable_variables))
        return

    def train(self, train_data, num_epochs, batch_size=32, lr=0.001):
        optimizer = tf.keras.optimizers.Adam(learning_rate=lr)
        for epoch in range(num_epochs):
            latent = self.analysis(train_data)
            self.mu = tf.math.reduce_mean(latent, 0)
            self.var = tf.math.reduce_variance(latent, 0)
            for i in range(0, len(train_data), batch_size):
                train_batch = train_data[i:i+batch_size]
                self.train_step(train_batch)
        return
```

9.2 Laplacian Neural Network Architecture

The following code details the neural network architecture used in the one-dimensional Laplacian source testing in Section 6.1.

```
analysis = tf.keras.Sequential([
  layers.Dense(1),
  layers.LeakyReLU(),
  layers.Dense(10),
  layers.LeakyReLU(),
  layers.Dense(1),
  layers.LeakyReLU()
])

synthesis = tf.keras.Sequential([
  layers.Dense(1),
  layers.LeakyReLU(),
  layers.Dense(10),
  layers.LeakyReLU(),
  layers.Dense(1),
  layers.LeakyReLU()
])
```

9.3 MNIST Neural Network Architecture

The following code details the analysis and synthesis transform architecture used to compare the performance of the ReLU and GDN activation functions on the MNIST dataset. Note that tfc stands for TensorFlow Compression, a Python library that contains data compression tools for TensorFlow.

```
GDN = True # If testing GDN activation function
# GDN = False # If testing ReLU activation function
analysis = tf.keras.Sequential(layers.Reshape((28, 28, 1)))
if GDN:
  analysis.add(layers.Conv2D(16, 3, padding=’same’, activation=tfc.GDN()))
  analysis.add(layers.MaxPooling2D())
  analysis.add(layers.Conv2D(32, 3, padding=’same’, activation=tfc.GDN()))
else: # Testing ReLU
  analysis.add(layers.Conv2D(16, 3, padding=’same’, activation=’ReLU’))
  analysis.add(layers.MaxPooling2D())
  analysis.add(layers.Conv2D(32, 3, padding=’same’, activation=’ReLU’))

analysis.add(layers.MaxPooling2D())
analysis.add(layers.Flatten())
analysis.add(layers.Dense(784))
analysis.add(layers.LeakyReLU())
analysis.add(layers.Dense(64))
analysis.add(layers.LeakyReLU())

synthesis = tf.keras.Sequential([
  layers.Dense(128),
  layers.LeakyReLU(),
  layers.Dense(784),
  layers.LeakyReLU(),
  layers.Reshape((7, 7, 16))
])
if GDN:
  synthesis.add(layers.Conv2DTranspose(32, 3, strides=2, padding=’same’, activation=tfc.GDN(
                                              inverse=True)))
  synthesis.add(layers.Conv2DTranspose(1, 3, strides=2, padding=’same’, activation=tfc.GDN(
                                              inverse=True)))
else: # Testing ReLU
  synthesis.add(layers.Conv2DTranspose(32, 3, strides=2, padding=’same’, activation=’ReLU’))
  synthesis.add(layers.Conv2DTranspose(1, 3, strides=2, padding=’same’, activation= ’ReLU’))
synthesis.add(layers.Reshape((28, 28)))
```

9.4 CT Scan Neural Network Architecture

The following code details the neural network architecture used in the final model. Once again, note that tfc stands for TensorFlow Compression, a Python library that contains data compression tools for TensorFlow.

```
analysis = tf.keras.Sequential([
  layers.Lambda(lambda x : x / 255.),
  layers.Conv2D(filters=16, kernel_size=3, strides=1, activation=tfc.GDN),
  layers.MaxPooling2D(),
  layers.Conv2D(filters=32, kernel_size=3, strides=1, activation=tfc.GDN),
  layers.MaxPooling2D(),
  layers.Conv2D(filters=64, kernel_size=3, strides=1, activation=tfc.GDN),
  layers.MaxPooling2D(),
  layers.Conv2D(filters=128, kernel_size=3, strides=1, activation=tfc.GDN),
  layers.MaxPooling2D(),
  layers.Flatten(),
  layers.Dropout(0.5),
  layers.Dense(2048),
  layers.LeakyReLU()
  ])

synthesis = tf.keras.Sequential([
  layers.Reshape((4,4,128), input_shape=(2048,)),
  layers.Dropout(0.5),
  layers.UpSampling2D(),
  layers.Conv2DTranspose(filters=128, kernel_size=3, strides=1, activation=tfc.GDN(inverse),
  layers.UpSampling2D(),
  layers.Conv2DTranspose(filters=64, kernel_size=3, strides=1, activation=tfc.GDN(inverse)),
  layers.UpSampling2D(),
  layers.Conv2DTranspose(filters=32, kernel_size=3, strides=1, activation=tfc.GDN(inverse)),
  layers.UpSampling2D(),
  layers.Conv2DTranspose(filters=1, kernel_size=3, strides=1, activation=tfc.GDN(inverse)),
  layers.Reshape((64, 64, 1)),
  layers.Lambda(lambda x : x * 255.)
  ])
```

References

[1] Johannes Ballé et al. “Nonlinear Transform Coding”. In: *IEEE Journal of Selected Topics in Signal Processing* PP (Oct. 2020), pp. 1–17. DOI: 10.1109/JSTSP.2020.3034501.

[2] Ke Yan et al. “DeepLesion: automated mining of large-scale lesion annotations and universal lesion detection with deep learning”. In: *Journal of Medical Imaging* 5.3 (2018), p. 036501. DOI: 10.1117/1.JMI.5.3.036501.

[3] D. Koff, P. Bak, and et al. “Pan-Canadian Evaluation of Irreversible Compression Ratios (“Lossy” Compression) for Development of National Guidelines”. In: *J Digit Imaging* 22 (2009), pp. 569–578. DOI: 10.1007/s10278-008-9139-7.

[4] SP Power et al. “Computed tomography and patient risk: Facts, perceptions and uncertainties.” In: *World Journal of Radiology* (2016). DOI: 10.4329/wjr.v8.i12.902.

[5] Geoffrey D. Rubin. “Lung nodule and cancer detection in computed tomography screening.” In: *Journal of thoracic imaging* (2015). DOI: 10.1097/RTI.0000000000000140.

[6] Eliezer Robinson et al. “Delay in diagnosis of cancer. Possible effects on the stage of disease and survival”. In: *Cancer* 54.7 (1984), pp. 1454–1460.

[7] Lucian Leape, Donald Berwick, and David Bates. “Counting Deaths Due to Medical Errors—Reply”. In: *JAMA : the journal of the American Medical Association* 288 (Nov. 2002), p. 2405. DOI: 10.1001/jama.288.19.2405-JLT1120-2-3.

[8] C. Bergeron et al. “Lack of CT scanner in a rural emergency department increases inter-facility transfers: a pilot study”. In: *BMC Research Notes* 10.1 (2017), p. 749. DOI: 10.1186/s13104-017-3071-1.

[9] James Holmes et al. “Rate and Reasons for Repeat CT Scanning in Transferred Trauma Patients”. In: *The American surgeon* 83 (May 2017), pp. 465–469. DOI: 10.1177/000313481708300519.

[10] Harvard Health Publishing. *Radiation risk from medical imaging.* Website. 2021. URL: https://www.health.harvard.edu/cancer/radiation-risk-from-medical-imaging.

[11] *How to Manage Your Medical Records.* 2022. URL: https://www.cmpa-acpm.ca/en/advice-publications/browse-articles/2003/a-matter-of-records-retention-and-transfer-of-clinical-records.

[12] Jason McKenzie and Stacy Goergen. *Computed Tomography (CT).* 2017. URL: https://www.insideradiology.com.au/computed-tomography/.

[13] Ravi Varma. “Storage media for computers in radiology”. In: *The Indian journal of radiology imaging* 18 (Nov. 2008), pp. 287–9. DOI: 10.4103/0971-3026.43838.

[14] *Ontario Ministry of Health - Health Services Funding.* https://www.health.gov.on.ca/en/pro/programs/ecfa/funding/hs_funding.aspx.

[15] Andrew Reichman. “File storage costs less in the cloud than in-house”. In: *Forrester Research, Cambridge, MA* (2011).

[16] Canadian Radio-television and Telecommunications Commission. *Internet.* Online. n.d. URL: https://crtc.gc.ca/eng/internet/internet.htm.

[17] Tamas Linder. *Data Compression and Source Coding I: Fundamentals of Rate Distortion Theory.* Jan. 2023.

[18] Tamas Linder. *Data Compression and Source Coding III: Scalar Quantization.* Mar. 2023.

[19] Tamas Linder. *Data Compression and Source Coding V: Transform Coding.* Mar. 2023.

[20] G.K. Wallace. "The JPEG still picture compression standard". In: *IEEE Transactions on Consumer Electronics* 38.1 (1992), pp. xviii–xxxiv. DOI: 10.1109/30.125072.

[21] Adam Gronowski. "Information Bottleneck Methods for Fairness and Privacy in Machine Learning". English. PhD thesis. 2022, p. 152. ISBN: 9798358407466.

[22] Diederik P. Kingma and Jimmy Ba. *Adam: A Method for Stochastic Optimization.* 2017. arXiv: 1412.6980 `[cs.LG]`.

[23] Y. Lecun et al. "Gradient-based learning applied to document recognition". In: *Proceedings of the IEEE* 86.11 (1998), pp. 2278–2324. DOI: 10.1109/5.726791.

[24] Saad Albawi, Tareq Abed Mohammed, and Saad Al-Zawi. "Understanding of a convolutional neural network". In: *2017 International Conference on Engineering and Technology (ICET).* 2017, pp. 1–6. DOI: 10.1109/ICEngTechnol.2017.8308186.

[25] Iqbal Sarker. "Deep Cybersecurity: A Comprehensive Overview from Neural Network and Deep Learning Perspective". In: *SN Computer Science* 2 (May 2021). DOI: 10.1007/s42979-021-00535-6.

[26] Johannes Ballé et al. *Variational image compression with a scale hyperprior.* 2018. arXiv: 1802.01436 `[eess.IV]`.

[27] Johannes Ballé, Valero Laparra, and Eero P. Simoncelli. *End-to-end optimization of nonlinear transform codes for perceptual quality.* 2016. arXiv: 1607.05006 `[cs.IT]`.

[28] Keyan Ding et al. "Comparison of Full-Reference Image Quality Models for Optimization of Image Processing Systems". In: *International Journal of Computer Vision* 129.4 (Jan. 2021), pp. 1258–1281. DOI: 10.1007/s11263-020-01419-7.

[29] David Minnen, Johannes Ballé, and George Toderici. *Joint Autoregressive and Hierarchical Priors for Learned Image Compression.* 2018. arXiv: 1809.02736 `[cs.CV]`.

[30] Martín Abadi et al. *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems.* Software available from tensorflow.org. 2015. URL: https://www.tensorflow.org/.

[31] G. Anthony Reina et al. "Systematic Evaluation of Image Tiling Adverse Effects on Deep Learning Semantic Segmentation". In: *Frontiers in Neuroscience* 14 (2020). ISSN: 1662-453X. DOI: 10.3389/fnins.2020.00065.

[32] Zhou Wang and Alan C. Bovik. "Mean squared error: Love it or leave it? A new look at Signal Fidelity Measures". In: *IEEE Signal Processing Magazine* 26.1 (2009), pp. 98–117. DOI: 10.1109/MSP.2008.930649.

[33] Peter Ndajah et al. "SSIM image quality metric for denoised images". In: Nov. 2010, pp. 53–57.

[34] Jim Nilsson and Tomas Akenine-Möller. *Understanding SSIM.* 2020. arXiv: 2006.13846 `[eess.IV]`.

[35] Johannes Ballé, Valero Laparra, and Eero P. Simoncelli. *Density Modeling of Images using a Generalized Normalization Transformation.* 2016. arXiv: 1511.06281 `[cs.LG]`.

[36] Lu Lu. "Dying ReLU and Initialization: Theory and Numerical Examples". In: *Communications in Computational Physics* 28.5 (June 2020), pp. 1671–1706. DOI: 10.4208/cicp.oa-2020-0165.

[37] Douglas A Reynolds et al. “Gaussian mixture models.” In: *Encyclopedia of biometrics* 741.659-663 (2009).

[38] V.K. Goyal. “Theoretical foundations of transform coding”. In: *IEEE Signal Processing Magazine* 18.5 (2001), pp. 9–21. DOI: 10.1109/79.952802.

[39] Tamas Linder. *Data Compression and Source Coding II: Lossless Data Compression.* Feb. 2023.

[40] “High-resolution 3D abdominal segmentation with random patch network fusion”. In: *Medical Image Analysis* 69 (2021), p. 101894. ISSN: 1361-8415. DOI: https://doi.org/10.1016/j.media.2020.101894.

[41] George Toderici et al. *Full Resolution Image Compression with Recurrent Neural Networks.* 2017. arXiv: 1608.05148 [cs.CV].

[42] Salman Khan et al. “Transformers in Vision: A Survey”. In: *ACM Comput. Surv.* 54.10s (Sept. 2022). ISSN: 0360-0300. DOI: 10.1145/3505244.