



SELINUS UNIVERSITY

OF SCIENCES AND LITERATURE

AN EFFECTIVE JPEG IMAGE COMPRESSION TECHNIQUE USING CONVOLUTIONAL NEURAL NETWORK(CNN) TO REDUCE CONSTRUCTION LOSS

SUMAN KUNWAR

2022

DECLARATION

The dissertation titled “An Effective JPEG Image Compression Technique Using Convolutional Neural Network (CNN) to Reduce Construction Loss” is submitted for the degree of Doctor of Philosophy in Computer Science at Selinus University. This study is my original research work. All peer-reviewed journal articles and academic books cited in this dissertation are referenced in keeping with the Code of Honor and Academy Integrity.

“I do hereby attest that I am the sole author of this Ph.D. Dissertation and that its contents are only the result of my readings and research.”

Suman Kunwar

August 2022

ACKNOWLEDGEMENTS

Firstly, I would like to express my sincere gratitude to Prof. Dr. R. Logeswaran N. Rajasvaran for the initial support related research, for his patience, motivation, and immense knowledge. His guidance helped me in all the time of research and writing of this thesis.

In addition, I would like to dedicate this work to my parents, whose love, patience, and support have been a constant source of inspiration for me. Last but not least, I would like to acknowledge my dear wife Yashu Shrestha, who has been with me all these years and made the best of my life. This accomplishment would not have been possible without her love and support.

Suman Kunwar, Lufkin, August 2022

ABSTRACT

Interest in digital imaging has grown tremendously in recent years. As a result, various data compression techniques have been proposed, which are mainly concerned with minimizing the information used to represent images. Among them, JPEG compression is one of the most common techniques that has been widely used in multimedia and digital applications. There are several transformations that can determine the frequency of images, including the Discrete Fourier Transform (DFT), Discrete Wavelet Transform (DWT), and Discrete Cosine Transform (DCT). From its early discovery, DFT has been used in image processing and compression. It converts a spatial image to a frequency image. Due to the periodic nature of DFT, it is impossible to meet the periodic condition of opposite borders of an image causing severe artifacts, resulting in a decreased perceptual visual quality image. On the other hand, deep learning has produced remarkable results for tasks such as natural language processing, visual recognition, image compression, and speech recognition in recent years. Convolutional Neural Networks (CNN) have been studied more extensively than most other types of deep neural networks. The use of convolution in feature extraction reduces the size of the dataset and produces a less redundant feature map, which is an important part of image compression. This dissertation begins with the discussion of DFT in relation to image compression, discusses the problem causing a discontinuity in an image during compression, and then proposed a model based on a literature review and identified factors. The proposed model is then implemented, and the results are measured. The study results indicated several significant findings that suggested CNN can achieve good compression with better reconstruction. The results also shows that the proposed model outperformed the JPEG compression using DFT and has provided ample of room for improvements.

TABLE OF CONTENTS

DECLARATION	ii
ACKNOWLEDGEMENTS	iii
ABSTRACT	iv
LIST OF TABLES	vii
LIST OF FIGURES	viii
LIST OF ABBREVIATIONS	x
CHAPTER 1: INTRODUCTION	1
1.1 Background	1
1.2 Problem Statement	2
1.3 Purpose of the Study	4
1.4 Objectives of the Study	4
1.5 Significance of the Study	4
1.6 Scope of the Study	5
1.7 Structure of the Thesis	5
CHAPTER 2: LITERATURE REVIEW	7
2.1 Introduction	7
2.2 JPEG Compression	7
2.2.1 JPEG Variations	8
2.3 Discrete Fourier Transform (DFT)	9
2.4 Border Effect Problem	9
2.4.1 Window Functions	12
2.5 Deep Learning	14
2.6 Convolutional Neural Network (CNN)	15
2.6.1 Types of CNN Architecture	22
2.6.2 Optimization Algorithms	27
2.6.3 Generalization, Overfitting, and Underfitting	29
2.7 Related Works	30

2.8	Comparison of Previous Methods	32
2.9	Summary	36
CHAPTER 3: RESEARCH METHODOLOGY		37
3.1	Introduction	37
3.2	Research Framework	37
3.2.1	Data Collection and Preprocessing	38
3.2.2	Model Development and Testing	39
3.2.3	Quality Evaluation	40
3.3	Research Plan	42
3.4	Summary	43
CHAPTER 4: IMPLEMENTATION		44
4.1	Introduction	44
4.2	Coding	46
4.3	Testing	61
4.4	Summary	62
CHAPTER 5: RESULTS AND ANALYSIS		63
5.1	Introduction	63
5.2	Results	63
5.3	Evaluation of the Proposed Method	67
5.4	Quality Measurements	70
5.5	Summary	78
CHAPTER 6: DISCUSSION AND CONCLUSIONS		79
6.1	Introduction	79
6.2	Discussion and Conclusion	79
6.3	Contribution of the study	79
6.4	Future Recommendations	80
6.5	Summary	80
REFERENCES		81

LIST OF TABLES

Table 2.1:	Comparison of various CNN techniques to minimize JPEG artifacts . .	38
Table 4.1:	Composition of encoder layer	45
Table 4.2:	Composition of decoder layer	45
Table 4.3:	Composition of model	46
Table 5.1:	Comparison of proposed model with different settings	69
Table 5.2:	Comparison of proposed model output with JPEG	70
Table 5.3:	Quality measurement using quality assessment techniques for original vs reconstructed image	71
Table 5.4:	Quality measurement using quality assessment techniques for noisy vs reconstructed image	75

LIST OF FIGURES

Figure 1.1:	Original and Compressed image after DFT with respective magnified view	3
Figure 2.1:	JPEG compression technique (Kunwar, 2018)	8
Figure 2.2:	Input image and its magnitude calculation	10
Figure 2.3:	Fourier magnitude after applying logarithmic operator	10
Figure 2.4:	Phase of transformed image	11
Figure 2.5:	Image after Inverse Fourier Transform	12
Figure 2.6:	The three-dimensional diagrams of different window functions (Dong et al., 2019)	13
Figure 2.7:	The resulting images of image filtered by different window functions and the corresponding amplitude spectrum (Dong et al., 2019)	14
Figure 2.8:	CNN Architecture, comprised of just five layers (O’Shea & Nash, 2015)	16
Figure 2.9:	Visual representation of a convolutional layer.	16
Figure 2.10:	Simple three-layer feedforward neural network (FNN) consisting of an input layer, a hidden layer, and an output layer. (O’Shea & Nash, 2015)	18
Figure 2.11:	Sigmoid, Tanh, ReLU and Leaky ReLU activation functions	18
Figure 2.12:	AlexNet Architecture (Krizhevsky et al., 2012a)	23
Figure 2.13:	Residual learning: a building block (He et al., 2015)	23
Figure 2.14:	Architecture of VGG (Alzubaidi et al., 2021)	24
Figure 2.15:	Block diagram of Xception block architecture (Alzubaidi et al., 2021)	25
Figure 2.16:	ZFNet architecture (Zeiler & Fergus, 2013)	25
Figure 2.17:	A 5-layer dense block with growth rate of k=4. Each layers takes all preceding feature-map as input (Huang et al.,2016)	26
Figure 2.18:	Basic architecture of GoogleNet block (Alzubaidi et al., 2021)	27
Figure 2.19:	Underfitting, overfitting and good fitting examples (Kiourt et al., 2020)	30
Figure 3.1:	Research Framework	38
Figure 3.2:	Proposed Model Architecture	39

Figure 4.1:	Process Flow Chart	44
Figure 5.1:	Loss vs Model & Accuracy vs Model Accuracy after 15 epochs	63
Figure 5.2:	Original and reconstructed images over 15 epochs	64
Figure 5.3:	Loss vs Model & Accuracy vs Model Accuracy after training 400 epochs	64
Figure 5.4:	Loss vs Model & Accuracy vs Model Accuracy after training 200 epochs	65
Figure 5.5:	Original and reconstructed images over 200 epochs	65
Figure 5.6:	Loss vs Model & Accuracy vs Model Accuracy after training 400 epochs	66
Figure 5.7:	Original and reconstructed images over 400 epochs	66
Figure 5.8:	Loss vs Model & Accuracy vs Model Accuracy after 15 epochs	67
Figure 5.9:	Loss vs Model & Accuracy vs Model Accuracy after 100 epochs	67
Figure 5.10:	Loss vs Model & Accuracy vs Model Accuracy after 100 epochs using batch size of 32	68
Figure 5.11:	Loss vs Model & Accuracy vs Model Accuracy after 100 epochs with regularization.	68
Figure 5.12:	Loss vs Model & Accuracy vs Model Accuracy after training 200 epochs with regularization	69
Figure 5.13:	Original, JPEG and Model Output image	70

LIST OF ABBREVIATIONS

Adam	Adaptive Moment Estimation
BP	Back Propagation
CNN	Convolutional Neural Network
CR	Compression Ratio
DCT	Discrete Cosine Transform
DFT	Discrete Fourier Transform
DPCM	Differential Pulse Code Modulation
DWT	Discrete Wavelet Transform
ELU	Exponential Linear Unit
FSIM	Feature Similarity Indexing Method
GD	Gradient Descent
JPEG	Joint Photographic Experts Group
MSE	Mean Square Error
PSNR	Peak Signal to Noise Ratio
ReLU	Rectified Linear unit
ResNet	Residual Network
SGD	Stochastic Gradient Descent
SSD	Single Shot Multibox Detector
SSIM	Structural Similarity Index Matrix
VGG	Visual Geometry Group

CHAPTER 1

INTRODUCTION

1.1 BACKGROUND

In today's digital era, human beings are surrounded by digital gadgets. Photographs are now an integral part of a person's daily life, and digital images are widely used in a variety of applications. With the increase in megapixels of digital cameras, more memory and bandwidth are required for storing and transmitting digital images (Goel & Gabarda, 2016). As digital imaging and multimedia services advance, more and more people can share their data on the Internet. The number of internet users is growing day by day rapidly (Rahman et al., 2018), resulting in increased data transfer, which necessitates efficient image compression.

Images can be compressed in two ways: lossy and lossless (Wang et al., 2021). Lossy image compression techniques are non-reversible and can achieve a higher compression ratio whereas lossless methods provide the best visual experience. In multimedia, JPEG is one of the most used lossy compression techniques (Hussain et al., 2020). There are several variations of JPEG: JPEG 2000, JPEG XS, JPEG Systems, JPEG Pleno, and JPEG XL (SMPTE, 2020). Based on the data provided by Web Technology Survey, 74.3% of websites use the JPEG image format (W3Techs, 2022). In digital images, pixels have high correlations, and the removal of this correlation will not affect the visual quality of the image (Gonzalez et al., 2009; Yuan & Hu, 2019). To achieve the best quality with the smallest possible size, the low frequency values are preserved as they define the content of the image, and the high frequency values are truncated by a certain amount (Li & Lo, 2019; Rasheed et al., 2020). Discrete Fourier Transform (DFT) has been used for image processing and compression since its discovery due to its good performance (Rasheed et al., 2020).

DFT has been used in many of the applications such as Orthogonal Frequency Division Multiplexing (OFDM) systems (Kudeshia & Potnis, 2017), texture synthesis (Abraham et al., 2005), big data analysis (Giannakis et al., 2014), spectral analysis (De Carvalho et al., 2014), image registration (Tzimiropoulos et al., 2010), super resolution

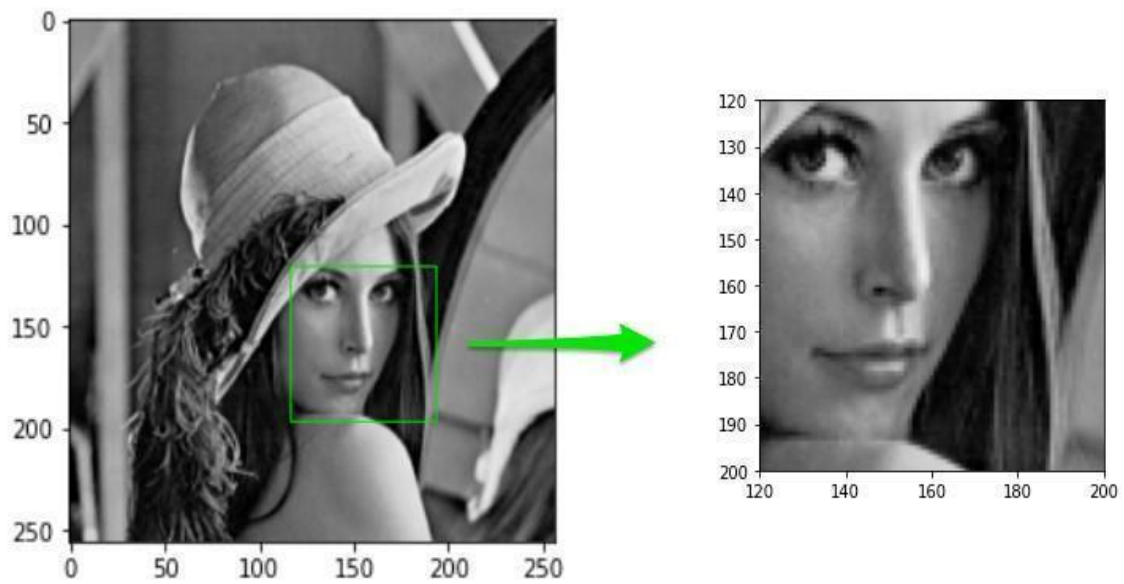
(Tian & Ma, 2011), image denoising (Shui, 2009), audio processing (Malvar, 1999) and object tracking (Annoni & Forster, 2012; Naftel & Khalid, 2006).

With the help of DFT, images in the spatial domain can be converted into the frequency domain, and certain frequencies can be ignored or modified to produce a low-information image with adequate quality (Siddeq & Rodrigues, 2014a; Siddeq & Rodrigues, 2014b; Siddeq & Al-Khafaji, 2013).

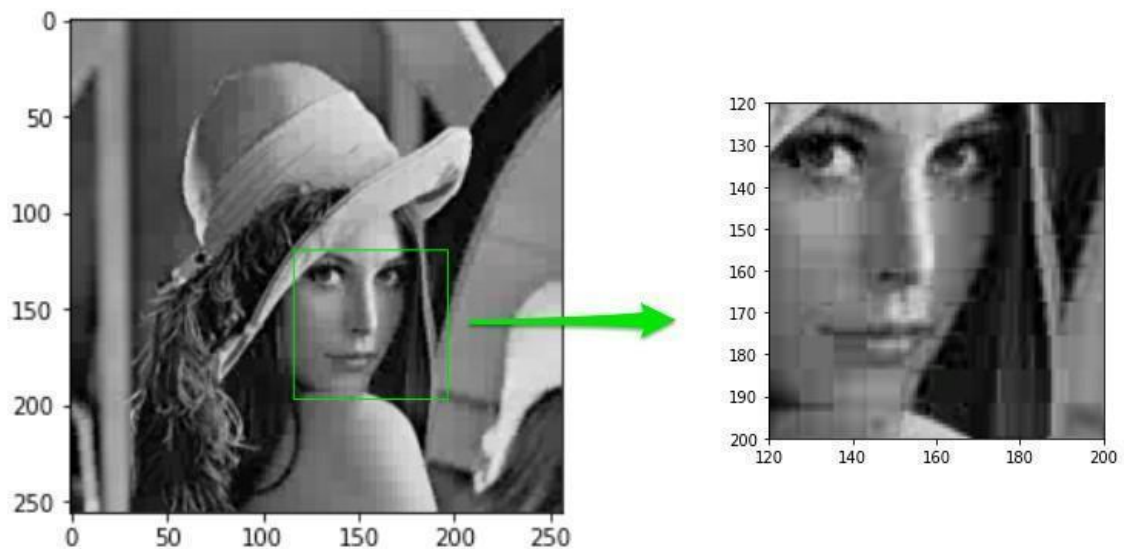
In practical application, when we compute the DFT of an image, it is impossible to meet the periodic condition that opposite borders of an image are alike, and the image always shows strong discontinuities across the frame border. As a result, this affects the registration accuracy and success rate and requires (Dong et al., 2019). To solve this problem, various approaches have been taken. Among them, raised-cosine window (Leprince et al., 2007), blackman window (Podder et al., 2014), and flap-top window (Ge et al., 2014) are the most popular ones. While these solutions are promising, still some information is lost in the process. Furthermore, other techniques have used neural networks.

1.2 PROBLEM STATEMENT

When computing the DFT of an image, the image is assumed to be periodic. It is unreasonable to assume that the opposite border will be similar in all circumstances, so the assumption of a periodic image will produce strong discontinuities across the border (Dong et al., 2019). Moreover, the discontinuities give rise to blocking and ringing artifacts in the Fourier Transform, specifically the cross structure of high energy coefficients along the axes and is referred to as the image border effect. The original image and the compressed image, together with magnified views, are shown in Figure 1.1. The strong discontinuity at the border can be seen in Figure 1.1(b).



(a) Original image with a magnified view



(b) Compressed image with a magnified view

Figure 1.1: Original and Compressed image after DFT with respective magnified view

The most commonly used approach for measuring the effect of the border of the image is to weight the reference and sensed images separately using a window function in the spatial domain. Furthermore, the operation of the weighting window function can cause additional problems, such as reducing the amount of common and useful information for the registration of image pairs based on phase correlation, especially in

cases of small patch-based matching or small overlaps between image pairs (Dong et al., 2019).

1.3 PURPOSE OF THE STUDY

This research aims to develop an effective JPEG image compression technique to minimize construction loss. Different techniques will be analyzed based on the state of the art, and their appropriateness will be determined to produce an effective image compression algorithm.

Due to DFT's periodic nature, it is unlikely that the opposite borders of an image are always identical. As a result, artifacts are created, and the quality of the image is reduced. Using convolution, it is possible to reduce the dimensions of data and produce a less redundant data set. Features extraction, highly accurate recognition results and the possibility of retaining existing networks makes Convolutional Neural Network (CNN) more popular. Based on the current state of art, this research aims to reduce the construction loss that happens during image compression by purposing an effective compression technique.

1.4 OBJECTIVES OF THE STUDY

The objectives of this research are as follows:

- To investigate the features that causes the construction losses.
- To design and develop a more efficient JPEG image compression method in order to reduce construction loss.
- To evaluate the performance of the developed method using state of art quality metrics.

1.5 SIGNIFICANCE OF THE STUDY

In the modern digital age, image compression techniques are essential to improve the performance of images and videos on the internet and in multimedia, including storage space, bandwidth usage, and secure transmission. A wide range of image compression is available, each with different compression ratios and levels of coding complexity. Among

them, JPEG compression is one of the most popular lossy image compression methods, which trades off file size with image quality.

The DFT algorithm produces a complex number that requires a lot of storage for JPEG compression, and it produces discontinuities in the pixels around the edges, making it less efficient than other compression methods. Minimizing this problem with a promising result will result in a better-quality image. Standards can also be formulated using existing research and use the resulting model for further research.

1.6 SCOPE OF THE STUDY

While there are several compression standards for JPEG, including JPEG, JPEG-LS, and JPEG-2000, this study only addresses JPEG since it is the most used image compression in multimedia applications (Hussain et al., 2020; W3Techs, 2022), and focuses on the discontinuity across the border problem, due to reconstruction loss of DFT, which has demonstrated promising results with recent advances in image processing and computational power (Cavigelli et al., 2015; Hussain et al., 2020; Maleki et al., 2018; Mao et al., 2016).

1.7 STRUCTURE OF THE THESIS

This chapter provides information about JPEG, DFT, and highlights the need for compression in digital media. It later describes the discontinuity caused by the DFT in the problem statement and advocates the need for this research. The purpose of this research, its objectives, and its significance along with its scope are described in respective sections of Chapter 1.

Chapter 2 gives more in-depth knowledge about the DFT's border effect problem in relation to JPEG compression, JPEG variations, DFT, Deep learning, CNN, and highlights various types of CNN Architectures. Optimization Algorithms and the concept of Generalization, Overfitting and Underfitting is also introduced here. Later, it describes the relevant work in the field of JPEG compression using neural networks and points out the previous works.

Chapter 3 explains the research methodology that is applied in this research. It is further decomposed into the research framework, research plan, and summary section to conclude this chapter. Data collection and pre-processing, model development and testing, and performance evaluation are described in the research framework section of this chapter. The research plan is set up here to accomplish the research.

Chapter 4 discusses the implementation of the proposed method. Here, the actual development of the model happens. It is then followed by testing where the performance of the proposed method is tested along with the reconstructed image quality using different state of art quality metrics.

Chapter 5 discusses the results obtained from model testing. Here, the performance of the model along with the quality of the reconstructed result is also measured.

Chapter 6 summarizes this dissertation with a discussion and conclusion. Conclusions are driven from the evaluation of proposed method. Contribution of this study and future recommendations are also discussed here.

CHAPTER 2

LITERATURE REVIEW

2.1 INTRODUCTION

In order to gain exposure with regards to the different aspects of the existing problem of border effects in JPEG compression, a detailed study is conducted. The existing solutions and proposed methods are explored, compared by various aspects.

In this chapter, the background to JPEG compression as well as JPEG variations are examined, followed by a discussion of the border effect problems in JPEG image compression. It then compares the different solutions based on the window functions and also lists the neural network solutions. Additionally, the usage of CNN, and performance indicators are discussed. This chapter summarizes previous works and compares them with state-of-the-art performance.

2.2 JPEG COMPRESSION

JPEG is one of the most used compression techniques that has been widely accepted as a standard for lossy image compression (Kunwar, 2017). JPEG, JPEG-LS, and JPEG2000 are some of the compression standards available for JPEG. The Joint Photographic Experts Group has developed two basic image compression algorithms (Wallace, 1991), one of which defines a combination of prediction method and entropy coding, and the other defines a hybrid compression method based on DCT (Rao & Ochoa-Dominguez, 2019) and entropy coding. The first method is a lossless compression technique based on Differential Pulse Code Modulation (DPCM), while the second is a lossy compression technique and is mainly used because of its high compression ratio.

When compression occurs, the result is a tradeoff between storage size and image quality. Image quality can be compromised when distortions occur during the acquisition and processing of images. Noise, blurring, ringing, and compression artifacts are examples of distortion. The basic JPEG image compression technique is shown in Figure 2.1.

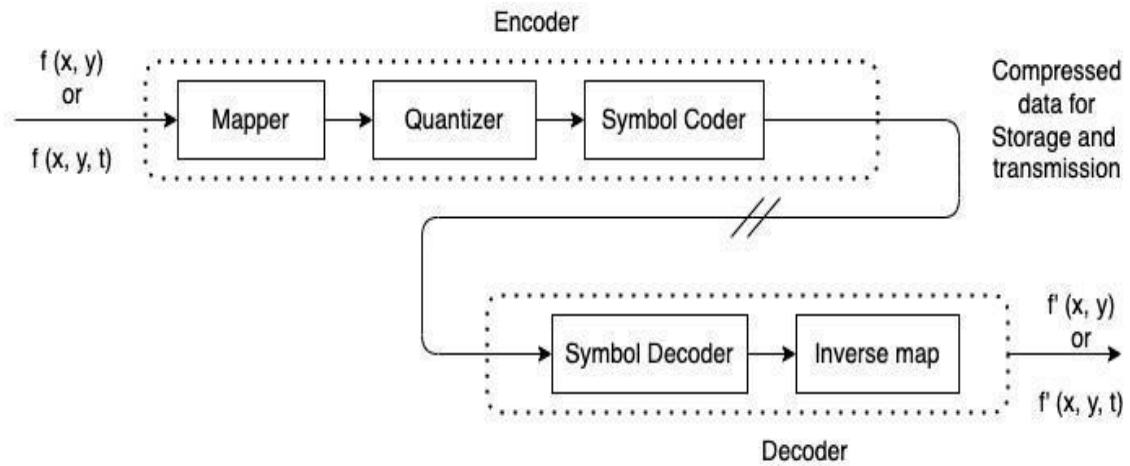


Figure 2.1: JPEG compression technique (Kunwar, 2018)

2.2.1 JPEG Variations

Multiple variations of the JPEG format are available in order to meet the ever-changing needs of the market, including 3D, lossless compression, and video formats. The variations of JPEG are described below.

- A. **JPEG 2000:** The JPEG 2000 format is a flexible format that includes a number of advantages over the standard JPEG format, including a choice of lossy or lossless compression, an improved compression process, and support for high dynamic range. This format can be also used for videos.
- B. **JPEG XS:** It is designed for low-latency, multiple encoding-decoding cycles and is best suited for professional video and IP transport, as well as virtual and augmented reality usage. With this process, uncompressed data is replaced with visually lossless compressed data in formats such as RGB/444, RGBA/4444, YCbCr 444/422, YCbCrA 4444/42224, and YCbCr 420.

- C. **JPEG Systems:** To ensure better interoperability between legacy and future JPEG standards, JPEG Systems defines the overall framework for future JPEG standards. Several recent additions have included extensions for privacy, security and intellectual property rights (IPR), a standard for 360-degree images, and a universal metadata box format (JUMBF).
- D. **JPEG Pleno:** The JPEG Pleno standard defines “imaging modalities”. These are light representations that provide 3D spatial representations for the likes of texture-plus-depth, light field, and point cloud. The framework supports image manipulation, metadata, access and interaction.
- E. **JPEG XL:** It’s designed to replace the existing JPEG format with better quality images and higher compression. The advantage of JPEG XL is that it is backward compatible, which means that traditional JPEG decoders will be able to open the image, while newer JPEG XL decoders can access the embedded metadata to create a higher quality image.

2.3 DISCRETE FOURIER TRANSFORM (DFT)

DFT is a simplified Fourier transform and contains only a set of samples that is large enough to describe the spatial domain image. In DFT, discrete signals in the time domain are transformed into their discrete frequency domain representations (Sevgi,2014). It is because of this property that DFT is so important in the field of spectrum analysis. Number of frequencies corresponds to the number of pixels in the spatial domain image and are of the same size (Mathur & Mathur, 2012). The Fourier transform produces an output image weighted by complex numbers, which can be represented by two images, either the real and imaginary parts or magnitude and phase. In image processing, magnitude is often display as it contains most information about the geometric structure of the image in the spatial domain.

Magnitude and phase of the image are preserved after preprocessing in order to transform it back to the correct spatial domain. Comparing to spatial domain, Fourier domain image has a much larger range, and its values are usually calculated and stored in float values. The Fourier transform is used to access the geometric features of an image

in the spatial domain. By decomposing the image into its sinusoidal components in the Fourier domain, certain frequencies can be examined or processed, which affects the geometric structure in space. While implementing Fourier transformation, the image is shifted towards the center, farther the pixel value from a center, the higher its corresponding frequency. Fourier transform of an image (I) and its magnitude calculation result is shown in Figure 2.2(a) and Figure 2.2(b) respectively.



(a) Input image



(b) Magnitude Calculation

Figure 2.2: Input image and its magnitude calculation

Because of the large dynamic range, only the largest value in the center of the image is seen. All other values appear as black on the screen. A logarithmic operator is applied to the Fourier image instead, as shown in Figure 2.3.



Figure 2.3: Fourier magnitude after applying logarithmic operator

Results show image contains all frequencies, but their size becomes smaller at higher frequencies. The logarithmic operator amplifies low intensity pixel values while compressing high intensity values in a relatively small pixel area. Thus, if an image

contains important high-intensity information, using the logarithmic operator may result in information loss. Additionally, the transformed image shows us that the Fourier image has two dominant directions, one vertical and one horizontal. These were derived from patterns in the original image background. The phase of the Fourier transform of the same image is shown in Figure 2.4.

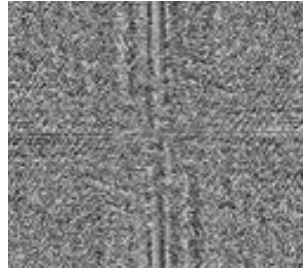


Figure 2.4: Phase of transformed image

The value of each point determines the phase of the corresponding frequency. The vertical and horizontal lines in the magnitude image correspond to the patterns in the original image. The phase image does not provide much new information about the structure of the image in the spatial domain. Therefore, in the following examples, only the magnitude of the Fourier transform is demonstrated. Although this image has the same frequency (and number of frequencies) as the original input image, it has been distorted beyond recognition. A correct spatial image reconstruction depends on phase information. For N real numbers DFT is defined as:

$$c_k = \frac{1}{N} \sum_{j=0}^{N-1} f_j e^{-ijk2\pi/N}, \quad k = 0, \dots, N - 1. \quad (2.1)$$

Inverse Discrete Fourier Transform maps c into f and is given by

$$f_k = \sum_{i=0}^{N-1} c_i e^{ijk2\pi/N}, \quad k = 0, \dots, N - 1. \quad (2.2)$$

In a straightforward implementation, the computations of matrix-vector products require N^2 multiplications. Before abandoning the phase image altogether, note the magnitude image is transformed using the inverse Fourier transform (and then histogram-equalized), the following result is obtained as shown in Figure 2.5.

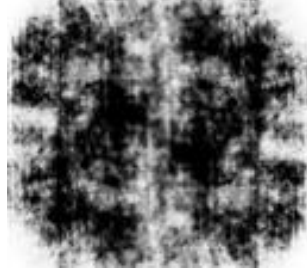


Figure 2.5: Image after Inverse Fourier Transform

2.4 BORDER EFFECT PROBLEM

Due to the implicit periodicity property of DFT and the fact that opposite borders on an image are not identical, images are always subject to a border effect when registering based on phase correlation. Various approaches have been proposed to solve this problem among them window functions and the deep learning are promising ones.

2.4.1 Window Functions

Window function reduces the impact of the edges of the image by using a center weighting method. Window functions should be smooth and continuous, allowing for better center selectivity, thus reducing interferences at the edges and preventing truncation errors. Various window functions are available to solve this problem. Among them, raised-cosine window, blackman window, and flap-top window are the most popular approaches (Dong et al., 2019).

- A. Blackman window: In signal processing, blackman window is one of the most commonly used function when using spectral analysis. The discontinuities across the image border are smoothed out for an image patch weighted by the blackman window, but a significant amount of signal is also degraded. The blackman window of N length has the following formula in one dimension (Lai, 2004):

$$w(n) = 0.42 - 0.5 \cos \frac{2\pi n}{N-1} + 0.08 \cos \frac{4\pi n}{N-1}, 0 \leq n \leq N-1 \quad (2.3)$$

- B. Raised-cosine window: Raised-cosine was also proposed to achieve a good balance between reducing the effects of image borders and retaining the information about the image. Here is the one-dimensional expression for the raised-cosine window of length N (Dong et al., 2019):

$$w(n_1, n_2) = \begin{cases} 1, & \Gamma(n_1, n_2) \geq 1, \\ \Gamma(n_1, n_2), & \text{otherwise} \end{cases} \quad (2.4)$$

where:

$$\Gamma(n_1, n_2) = k \cdot 0.5(1 - \cos(2\pi(n_1/M))) \cdot 0.5(1 - \cos(2\pi(n_2/N)))$$

M, N = sizes of the images

k = stretch factor

C. Flat-top window: It is an improved version of classical Hanning window, with involving parameters optimization in the process. A window function is added to the input images in the space domain and a weighting function to the spectrum in the frequency domain.

The result of an image filtered by different window function is illustrated in Figure 2.6. These functions are progressively blurring the border of the image and preserving the content away from the image's border. To avoid adding new discontinuous content to the images, some content near the border will be fuzzed to varying degrees. Each window function differs in how much information is processed near the image border. The three-dimensional diagrams of blackman window function, raised-cosine window function and flap-top window function are shown in Figure 2.6(a-c) respectively. The length of all window functions is 100. For raised-cosine window function, its roll-off factor β is set to 0.25. For the flap-top window function, its stretch factor k is set to 2.7.

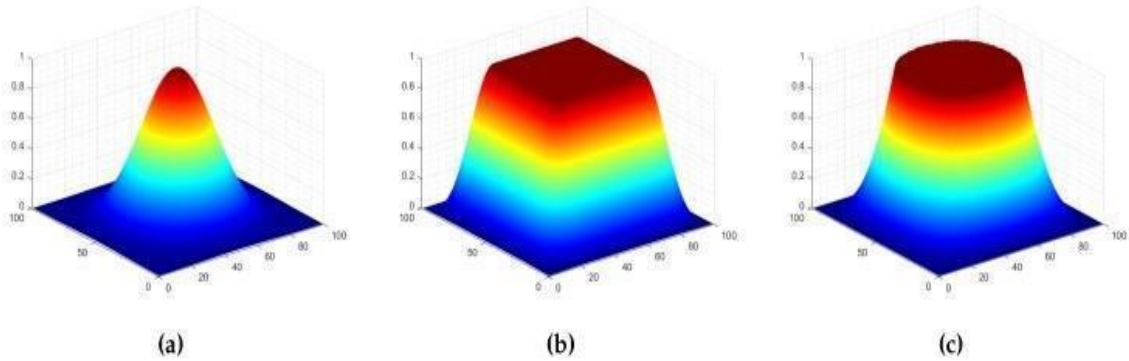


Figure 2.6: Three-dimensional diagrams of window functions (Dong et al., 2019).

The resulting images of image filtered by different window functions and the corresponding amplitude spectrum are shown in Figure 2.7. Here, the cross structure is clearly visible in the amplitude spectrum of the original image but disappears from the other three amplitude spectrums. The original image of its size 1000×1000 pixels is shown in Figure 2.7(a). The result images after filtering by Blackman, flap-top and raised-cosine window function are shown in Figure 2.7(b–d) respectively. Figure 2.7(e–h) are the corresponding amplitude spectrum of Figure 2.7(a–d), respectively.

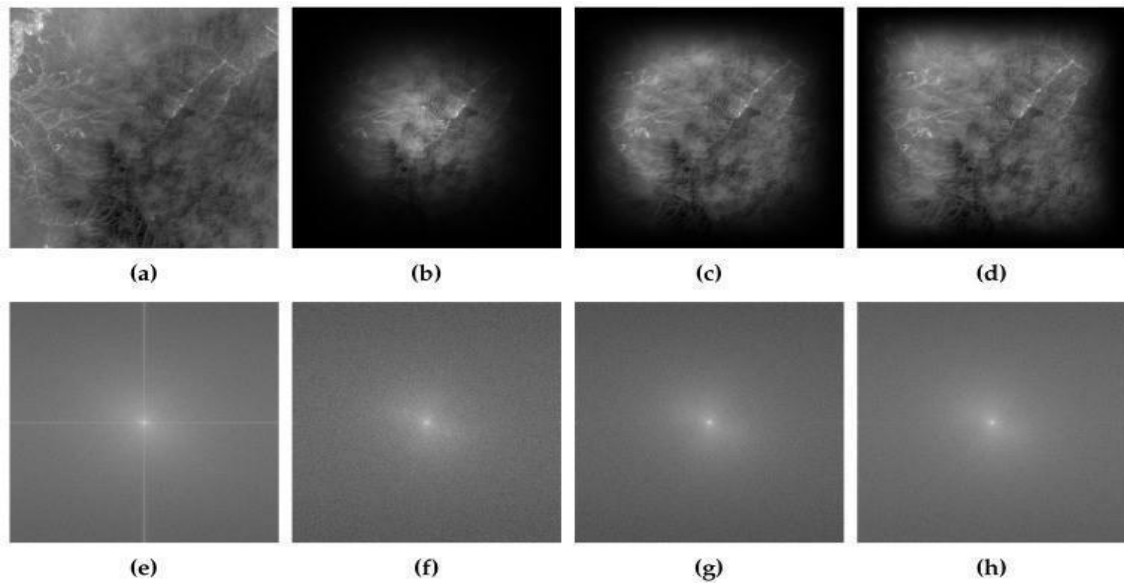


Figure 2.7: The resulting images of image filtered by different window functions and the corresponding amplitude spectrum (Dong et al., 2019).

The discontinuities across the image border are smoothed out for an image patch weighted by the blackman window, but a significant amount of signal is also degraded. In raised-cosine and flap-top, they blur the borders while keeping the middle of the image as unchanged as possible. Unfortunately, some contents near the border are also degraded.

2.5 DEEP LEARNING

Deep learning is a subset of machine learning and is designed in a way that mimics the human cerebral cortex. It has been applied to a variety of fields and proved their usefulness in many applications such as image classification (Ren et al., 2016), object recognition (Long et al., 2015), and semantic segmentation (Cavigelli et al., 2015). It has

also gained relevance for regression tasks in low-level image and video processing by computing saliency maps (Dosovitskiy et al., 2015), optical flow fields (Dong et al., 2014), and single-frame super-resolution images (Poor, 1988) with state-of-the-art performance.

Recursive Neural Network (RvNN), Recurrent Neural Network (RNN), CNN, Deep Generative Networks are some of the examples of deep learning networks (Pouyanfar et al., 2019). Among them, CNN is one of the most popular deep neural networks has gained more attention in the domain of computer vision, particularly for applications like detection and interpretation (Sujitha et al., 2021). As CNN have advanced rapidly, the problem of removing image artifacts from the decoded images has been re-examined. Several state-of-the-art deep learning-based algorithms have been developed with great success using this approach (Baig et al., 2017; Dong et al., 2019; Li, Wang, et al., 2020; Santurkar et al., 2017; Svoboda et al., 2016).

Approaches like using the Hann windows for reducing edge-effects in patch-based image segmentation with CNNs has shown promising results and pointed out the further investigation with different window functions and with reducing the amount of context needed (Pielawski & Wählby, 2020).

2.6 CONVOLUTIONAL NEURAL NETWORK (CNN)

CNN is a deep learning algorithm that takes in an input image, assigns weights and biases to various aspects and objects in the image, and then differentiates them from each other. It consists of neurons that have learnable weights and biases. Each neuron receives some input, performs a dot product, and optionally follows it with a non-linearity (Kunwar, 2018). CNNs are being increasingly used in image recognition and classification tasks, and have achieved great success (Krizhevsky et al., 2017).

The use of CNNs for deep learning is popular due to three important factors:

1. In CNN, features are learned directly from CNN, eliminating the need for manual feature extraction.
2. It produces highly accurate recognition results.

3. It is possible to retrain CNNs for new recognition tasks, so existing networks can be expanded.

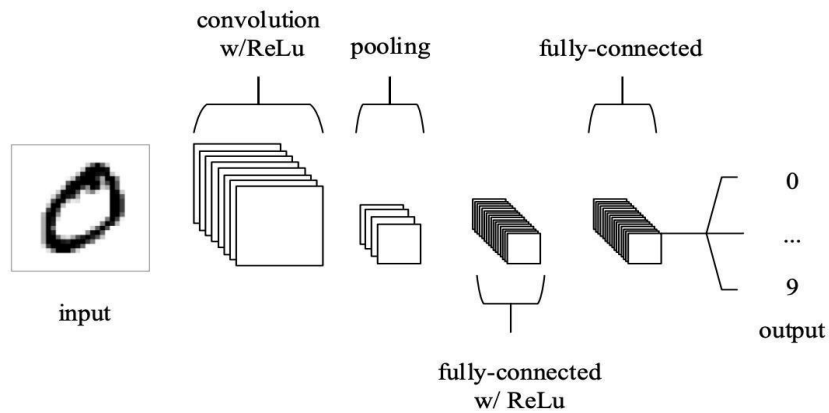


Figure 2.8: CNN Architecture, comprised of just five layers (O’Shea & Nash, 2015)

Figure 2.8 depicts the CNN architecture, comprised of five layers. Pixel values are stored in the input layers. The convolution layers compute the output of the neurons associated with local regions by computing the scalar product between the weights of the neurons and the region of the input volume. The Rectified Linear Unit (ReLU) adds an activation function to the previous layer’s output. Different layers of a CNN are described below:

- A. Convolutional layer: The convolutional layer consists of a set of learnable kernels/filters that perform convolution operations as they scan the input image according to its dimensions to produce an output called an activation map or a feature map.

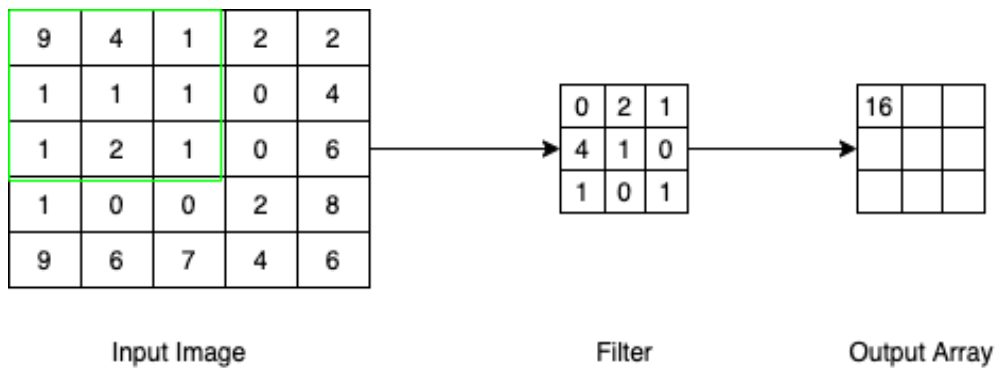


Figure 2.9: Visual representation of a convolutional layer.

As shown in Figure 2.9, the filter map value is connected to the receptive field, where the filter is being applied. The hyperparameters that affect the output size are Stride, Number of filters, and Zero-padding.

- **Stride:** The distance that the kernel travels across the input matrix. A stride value of two or more is rare, but a larger stride results in a smaller output.
- **Number of filters:** The number of filters affects the depth of the output, because each filter yields a different depth feature map.
- **Zero-padding:** In most cases, it is used when the filters do not fit the input image. The elements that are outside of the input matrix in this case are set to zero, resulting in an output that is larger or the same size. Valid padding, Full padding and Same padding are three different types of padding that can be used.

B. **Pooling layer:** A pooling layer reduces the number of parameters and computations in the network by progressively reducing the size of the representation. Each feature map is processed independently by the pooling layer. The function “MAX” maps the input and scales its dimensionality. Max-pooling is the most popular method of pooling.

C. **Fully connected layer:** The neurons in the fully connected layer are directly connected to neurons in the two adjacent layers, without being connected to any of the layers within them. In traditional ANNs, neurons are arranged according to the way in which they are shown on Figure 2.10.

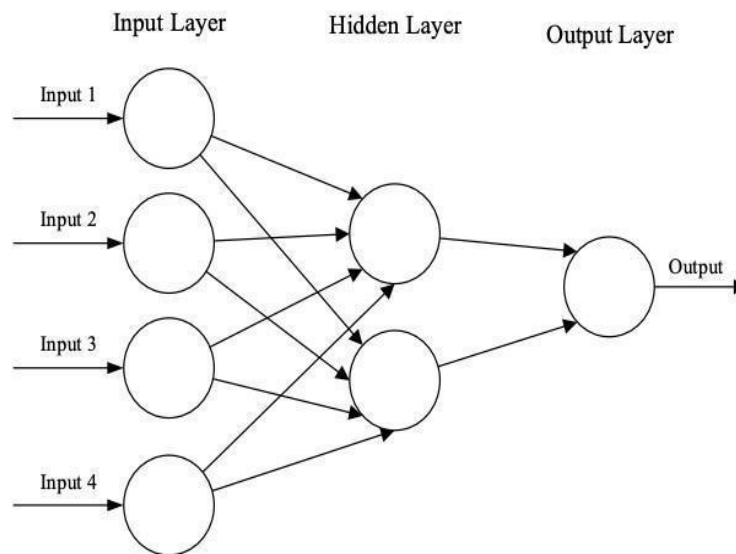


Figure 2.10: Simple three-layer feed forward neural network (FNN) consisting of an input layer, a hidden layer, and an output layer. (O’Shea & Nash, 2015)

D. Activation Function: The activation functions map the input to the output in all types of neural network. A neural network’s performance depends heavily on the activation function it uses, and different activation functions may be used in different parts. Some of the most commonly used activation functions are shown in Figure 2.11.

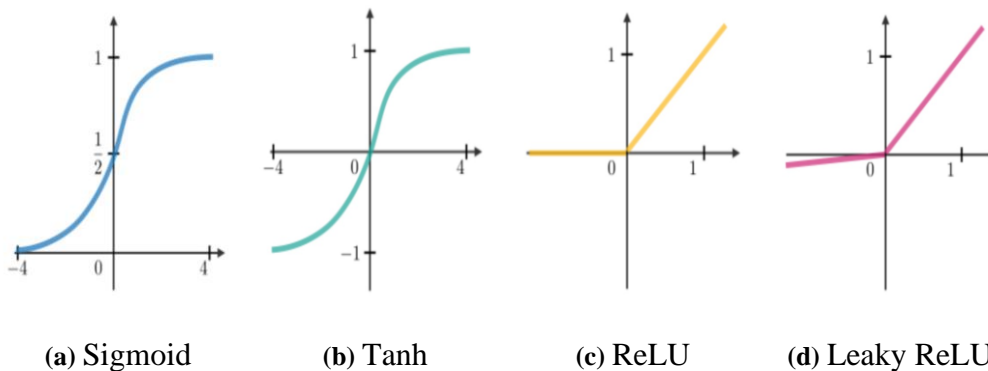


Figure 2.11: Sigmoid, Tanh, ReLU and Leaky ReLU activation functions

A linear activation function cannot perform backpropagation, while a nonlinear activation function can stack multiple layers of neurons into a deep neural network, that can learn complex data sets accurately and quickly. Some of the examples of non-linear activation functions are as follows:

- Sigmoid: This activation function takes real numbers as input and restricts the output to zero or one. Sigmoid function curves have an S-shaped shape and are mathematically represented by the equation below.

$$f(x)_{\text{sigm}} = \frac{1}{1 + e^{-x}} \quad (2.5)$$

- Tanh: Like the sigmoid function, it takes real numbers as input, but its outputs between -1 and 1 . In mathematics, it is represented by following equation.

$$f(x)_{\text{tanh}} = \frac{e^x - e^{-x}}{e^x + e^{-x}} \quad (2.6)$$

- ReLU: ReLU is one of the most used functions in CNN. It generates positive numbers from the whole value inputs. The main benefit of ReLU to others is its low lower computational load. It is mathematically represented by the equation below.

$$f(x)_{\text{ReLU}} = \max(0, x) \quad (2.7)$$

There are cases where ReLU neurons become inactive and only output 0 for any input, creating dying ReLU problem (Lu, 2020). Different solutions have been put forward to solve this problem.

- Leaky ReLU: ReLU downscales negative inputs, but this activation function ensures that these inputs are never ignored. It is used to solve the problem of Dying ReLUs. It is mathematically represented by the equation below.

$$f(x)_{\text{LeakyReLU}} = \begin{cases} x, & \text{if } x > 0 \\ mx, & x \leq 0 \end{cases} \quad (2.8)$$

- Noisy ReLU: To make ReLU noisy, this function uses a Gaussian distribution. It is mathematically represented by the equation below.

$$f(x)_{\text{NoisyReLU}} = \max(x + Y), \text{ with } Y \sim N(0, \sigma(x)) \quad (2.9)$$

- Parametric Linear Units: The main difference between this function and Leaky ReLU is that the leak factor in this function is updated during model training. It is mathematically represented by the equation below.

$$f(x)_{ParametricLinear} = \begin{cases} x, & \text{if } x > 0 \\ ax, & x \leq 0 \end{cases} \quad (2.10)$$

where:

a = learnable weight

E. Loss Functions: CNN architectures achieve final classification through the output layer, which represents the last layer. In the output layer, loss functions are used to calculate predicted errors created across the training samples. The error shows the difference between the actual and predicted outputs. Then, it will be optimized using CNN learning. Different problem types require different types of loss functions.

Here are some concise explanations of loss function based on regression problems and classification problems.

- Mean Absolute Error (MAE): It calculates the mean of the absolute error between the predicted value and the actual value. MAE results are not derivable, so the update rate cannot be estimated during optimization.

$$\sum_{i=1}^D |x_i - y_i| \quad (2.11)$$

- Mean Squared Error (MSE): In CNN, regression problems are mostly dealt with MSE or MAE (Li, Yang, et al., 2020). MSE calculates the mean of square error of the predicted value and the actual value. It results are derivable and is possible to control the update rate.

$$\sum_{i=1}^D (x_i - y_i)^2 \quad (2.12)$$

- **Cross-Entropy Function:** In addition to being known as the log loss function, this function is commonly used to measure CNN model performance. It is mostly used for multi-class classification problems and the output is measured in probability $p \in \{0, 1\}$ (Alzubaidi et al., 2021). For all classes in the problem, cross-entropy calculates the average difference between actual and predicted probability distributions (Brownlee, 2021). The mathematical representation of Cross-Entropy Function is given below.

$$p_i = \frac{e^{a_i}}{\sum_{k=1}^N e_k^a} \quad (2.13)$$

where:

e^{a_i} = non-normalized output from the preceding layer

N = number of neurons in the output layer

Cross entropy loss only considers the correctness of the classification, not compactness within a class or the margin between classes. To overcome these various approaches have been proposed such as Contrastive loss (Hadsell et al., 2006), triplet (Schroff et al., 2015), center loss (Wen et al., 2016), and large margin softmax loss (Liu et al., 2016).

- **Euclidean Loss Function:** In regression problems, this function is widely used. Additionally, it is also known as the mean square error. The mathematical representation of Euclidean Loss Function is given below.

$$H(p, y) = \frac{1}{2N} \sum_{i=1}^N (p_i - y_i)^2 \quad (2.14)$$

- **Hinge Loss Function:** The hinge loss function is an alternative to cross-entropy mostly used in binary classification problems, primarily designed for use with Support Vector Machines (SVMs) models. The maximum-margin-

based classification problem occurs mostly with SVMs, which use hinge loss functions, where the optimizer tries to maximize margins around dual objective classes. It is mathematically represented by the equation below.

$$H(p, y) = \sum_{i=1}^N \max(0, m - (2y_i - 1)p_i) \quad (2.15)$$

where:

p_i = predicted output

m = margin

y_i = desired output

2.6.1 Types of CNN Architecture

CNN architectures have evolved over time in different ways (Khan et al., 2020). It has undergone various modifications, such as structural reformulation, regularization, and parameter optimization. However, its performance improvement has been primarily attributed to the restructuring of processing units and the development of new blocks (Alzubaidi et al., 2021). Different types of CNN architecture are available based on the architectural modification. Some of them are discussed below:

- A. AlexNet: In 1998, LeNet became the first CNN architecture to be released (LeCun et al., 1995). It was developed to recognise hand-written digits from the MNIST Dataset. Later, AlexNet was proposed that achieve impressive results in terms of classification and image recognition (Krizhevsky et al., 2012b). AlexNet, uses more filters than LeNet, allowing it to categorise a much wider range of objects.

In AlexNet, there are 5 convolutional layers, including a max-pooling layer, 3 fully connected layers, and 2 dropout layers. To address overfitting, “dropout” was used instead of regularization. All layers use Rectified Linear Unit (ReLU) as their activation function except the output layer, which uses the Softmax activation function. Figure 2.12 shows the AlexNet Architecture.

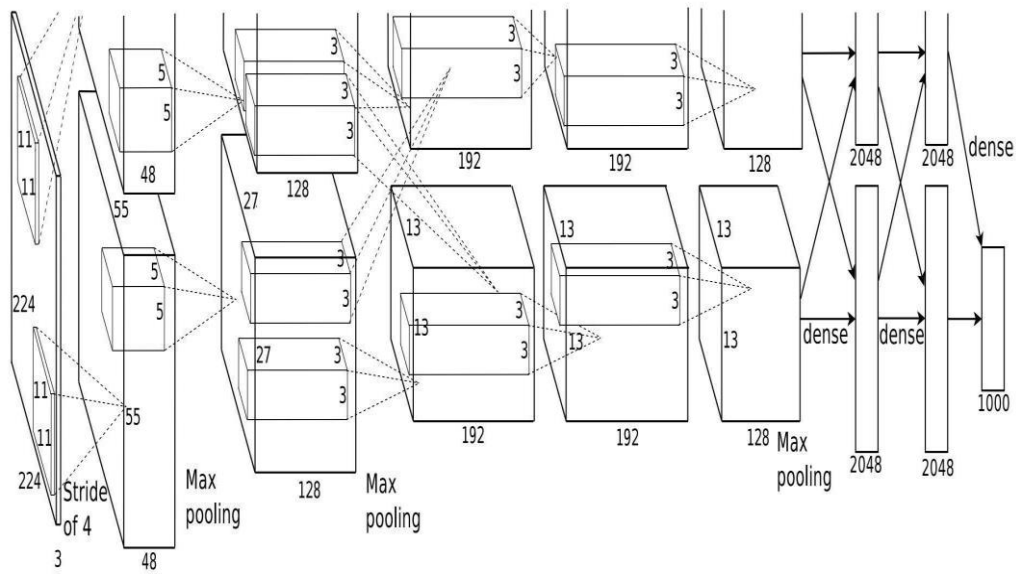


Figure 2.12: AlexNet Architecture (Krizhevsky et al., 2012a)

B. ResNet: ResNet (Residual Network) was developed by (He et al., 2015) with the objective to design an ultra-deep network free of the vanishing gradient issue, as compared to the previous networks. ResNets are made up of residual block. This is built on the concept of “skip-connections” and uses heavy batch-normalization to effectively train hundreds of layers without sacrificing speed. Figure 2.13 shows the residual block.

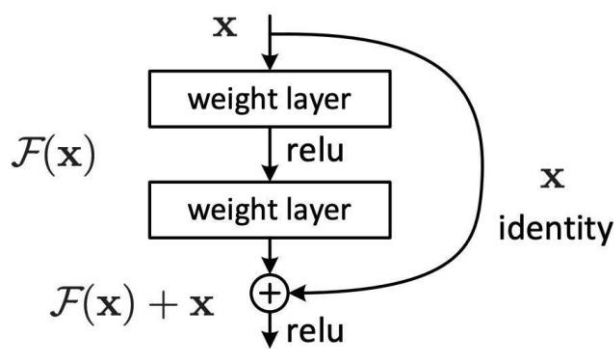


Figure 2.13: Residual learning: a building block (He et al., 2015)

With the idea of not having more mistakes than the shallower equivalents, skip connections were introduced (Shrivastav, 2021). The creator used a pre-activation

variation of the residual block so gradients can flow through the shortcut link to the earlier layers, minimizing the vanishing gradient problem.

- C. Visual Geometry Group (VGG): It is a simple and efficient design principle for CNN was proposed by Simonyan and Zisserman after CNN proved effective in image recognition. It's a multilayer model that has nineteen more layers than ZFNet (Zeiler & Fergus, 2014) and AlexNet (Rozenwald et al., 2020) to replicate the relationship between depth and network representational capability. Figure 2.14 shows the architecture of VGG.

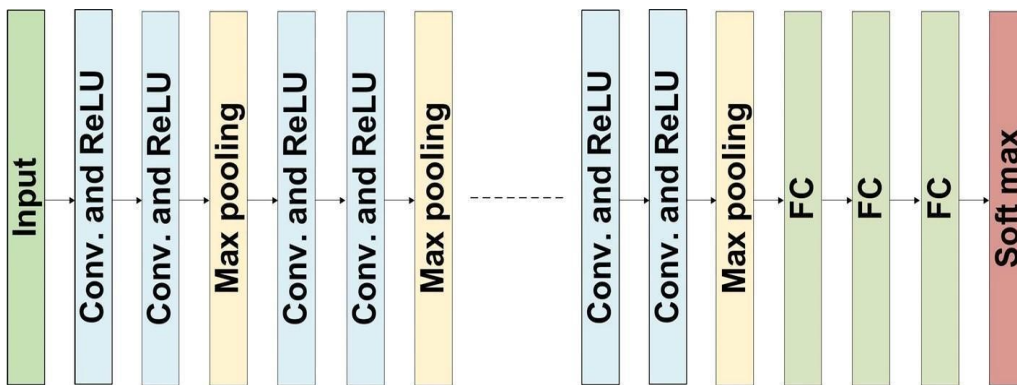


Figure 2.14: Architecture of VGG (Alzubaidi et al., 2021)

VGG showed that the filters with small sizes when assigned parallel enhances the CNN performance (Simonyan& Zisserman, 2014). With the decrease in the number of parameters, small-size filters reduced the computational complexity. As a result of these results, a novel research trend for CNN has emerged related to small-size filters.

- D. Xception: The main characteristic of Xception is the extreme inception architecture (Chollet, 2017). With the idea of depth wise separable convolution Xception was created. By increasing the size of the original inception block, Xception replaced the multiple spatial dimensions (1x1, 5x5, 3x3) with one dimension (3x3) followed by a convolution of 1x1. Through the separation of spatial and feature-map correlation, Xception improves the network's computational efficiency. Figure 2.15 depicts the Xception block's architecture.

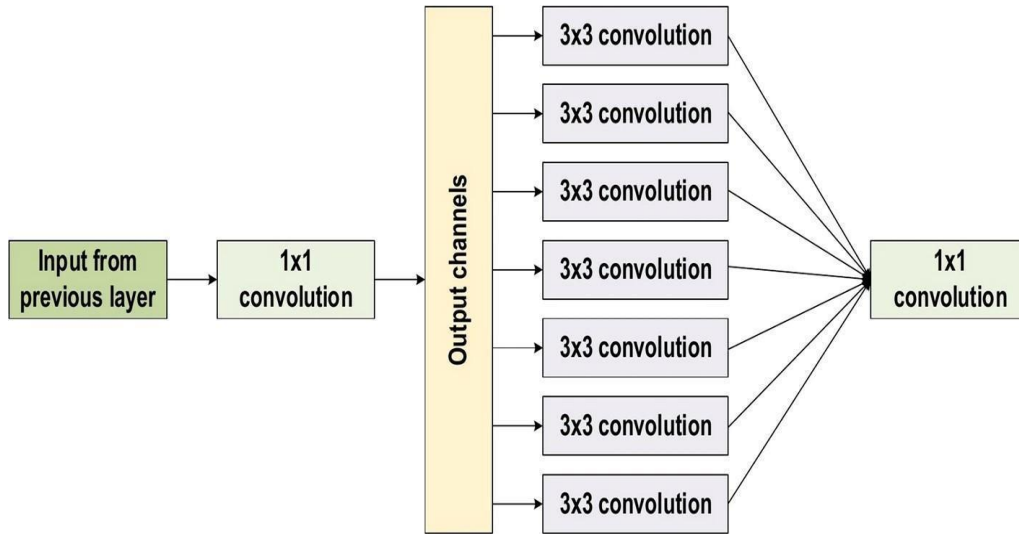


Figure 2.15: Block diagram of Xception block architecture (Alzubaidi et al., 2021)

The transformation technique used by Xception does not reduce the number of parameters, but it makes learning more efficient and leads to better performance (Lo et al., 2019).

- E. ZFNet: ZFNet is a multilayer Deconvolutional NN introduced by Zeiler and Fergus in 2013 (Zeiler & Fergus, 2013). ZFNet visualizes network performance statistically. Through the analysis of neuron activation, the network activity visualization tracked CNN performance.

Five shared convolutional layers, max-pooling layers, dropout layers, and three fully connected layers comprise its architecture. As part of the first layer, it used a 77 size filter and a low stride value. The last layer of the ZFNet is the softmax layer.

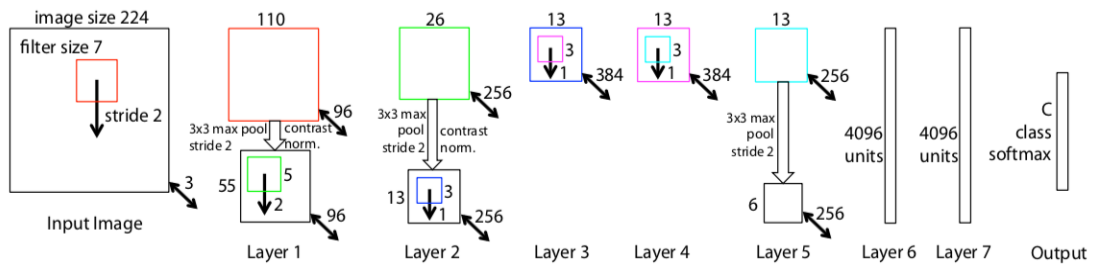


Figure 2.16: ZFNet architecture (Zeiler & Fergus, 2013)

- F. DenseNet: DenseNet was proposed to overcome the vanishing gradient problem in high-level network (Huang et al., 2016). It works in similar manner as Highway Networks and ResNet. ResNet retains information explicitly through additive identity transformation, which may result in many layers providing very little or no information. To solve this problem, DenseNet uses a modified form of cross-layer connection. With a feed-forward approach, DenseNet connects each layer to the next.

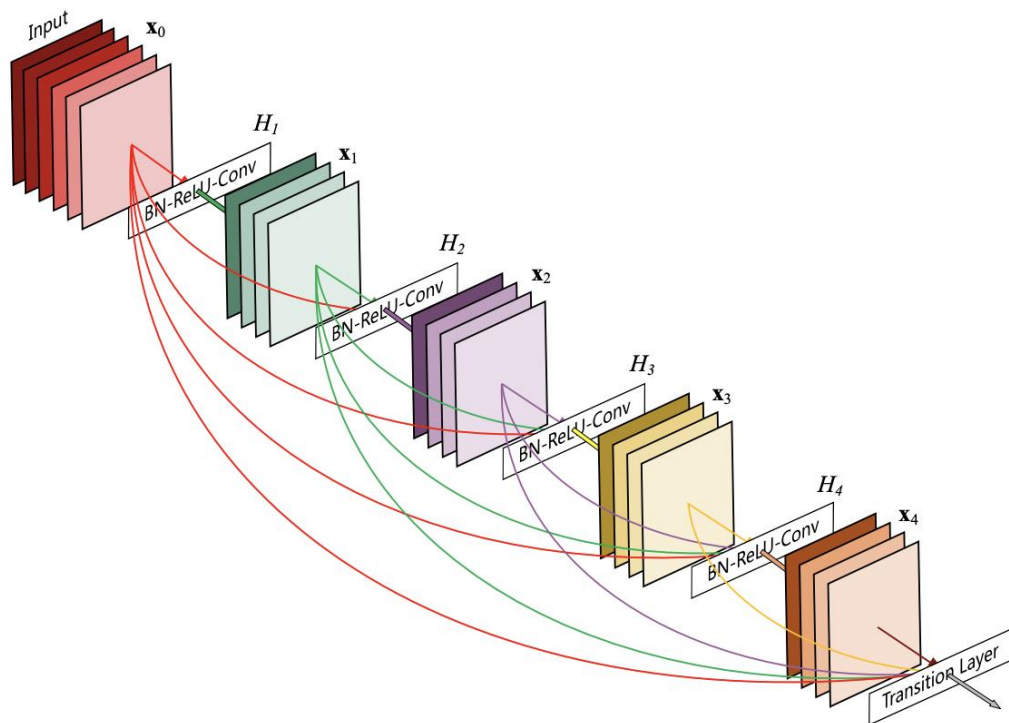


Figure 2.17: A 5-layer dense block with a growth rate of $k = 4$. Each layer takes all preceding feature-maps as input (Huang et al., 2016)

- G. GoogleNet: With the aim of achieving high-level accuracy and decreased computational cost, GoogleNet (also known as Inception-V1) architecture was created and won the 2014-ILSVRC competition (Szegedy et al., 2014). It uses a new inception block (module) that combines all the multiple-scale convolutional transformations for feature extraction. Figure 2.18 shows the basic architecture of GoogleNet.

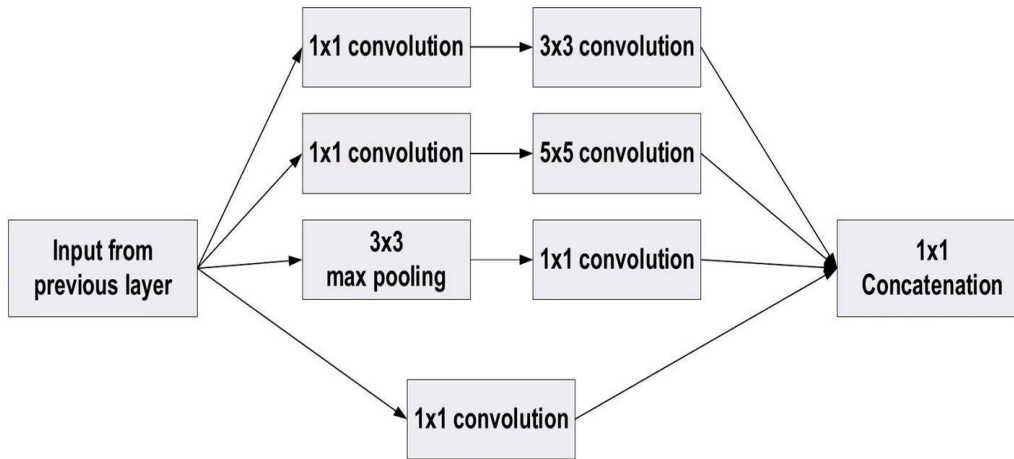


Figure 2.18: Basic architecture of GoogleNet block (Alzubaidi et al., 2021)

Filters of different sizes are incorporated together in this architecture to capture channel information along with spatial information at diverse ranges of spatial resolution. GoogleNet used sparse connections to overcome the redundant information problem and reduces cost by ignoring irrelevant channels.

2.6.2 Optimization Algorithms

Optimization is an integral part of machine learning. Most machine learning algorithms build an optimization model and learn parameters from the data (Sun et al., 2019). Machine learning models are widely used and popularized because of the effectiveness and efficiency of numerical optimization algorithms. A series of effective optimization methods were proposed to promote machine learning, which has improved machine learning's performance and efficiency (Hosseini et al., 2021; Nanni et al., 2021; Seyyarer et al., 2019). Some of the most commonly used optimization algorithms are described below:

- A. Adam: Adam (Adaptive Moment Estimation) is an optimizer that uses momentum and adaptive gradient to compute adaptive learning rates for each parameter (Kingma & Ba, 2014). Using exponential moving averages of the gradient and its square, it updates based on the gradient value at the current step. An exponentially moving average of past gradients is stored by Adam (m_t) and squared gradients (v_t).

$$\begin{aligned}
m_t &= \beta_1 m_{t-1} + (1 - \beta_1) g_t \\
v_t &= \beta_2 v_{t-1} + (1 - \beta_2) g_t^2
\end{aligned}
\tag{2.16}$$

β_1 and β_2 are hyper-parameters governing the decay rates. At the beginning of training, the moving averages are initialed as 0's, skewing the estimates of first and second moments towards zero. To counteract this, Adam utilizes correction terms

$$\begin{aligned}
\hat{m}_t &= \frac{m_t}{1 - \beta_1^t} \\
\hat{v}_t &= \frac{v_t}{1 - \beta_2^t}
\end{aligned}
\tag{2.17}$$

Then, the updated adam rule

$$\hat{x}_t = x_{t-1} - \frac{\eta}{\sqrt{\hat{v}_t} + \epsilon} \hat{m}_t
\tag{2.18}$$

Several variants of Adam are available including AdaMax, NAdam and AMSGrad (He et al., 2021). Adam algorithms failed to converge in convex example (Reddi et al., 2018).

- B. Stochastic Gradient Descent: Gradient Descent (GD) calculates the gradients using all training samples whereas Stochastic Gradient Descent (SGD) performs one weights update for each training sample. In gradient descent converging to a local minimum takes extensive time, and determining a global minimum is not always possible. Which has been addressed in SGD.

$$x = x - \eta \nabla l(x, s)
\tag{2.19}$$

It is faster than gradient descent, but gradients calculated from just one sample are not representative enough of the whole training dataset (Johnson & Zhang, 2013; Nanni et al., 2021). As a result, gradient variance causes the loss function to fluctuate intensely.

C. Momentum: In the loss function surface, SGD has difficulty navigating long, narrow valleys, in which the gradient is almost perpendicular to the long axis of the valley. Such a situation causes the system to oscillate back and forth in the short axis, while moving very slowly along the long axis.

Momentum strategy helps to counteract oscillation along the short axis by accumulating contributions along the long axis (Qian, 1999). It strengthens when gradients point in the same direction and dampens when gradients change, reducing the training loss in fewer steps than full batch gradient descent. By adding the previous update to the current update, momentum SGD determines the next update (v_t) as a linear combination of the gradient and the previous update (m_{t-1}):

$$v_t = \beta v_{t-1} - \eta \nabla l(x) \tag{2.20}$$

$$x = x - v_t$$

2.6.3 Generalization, Overfitting, and Underfitting

Generalization refers to the model's ability to estimate unseen test data (out-of-sample). In machine learning, the goal is generalization so that predictions can be made about future data and unseen data. Both overfitting and underfitting are caused by the lack of generalization.

Overfitting occurs when a model is trained so simply that its estimation has low variance and high bias. Models that are overfitted learn concepts from noise and fit closely to the training data, leading to poor performance on new data. Low error rates and a high variance are good indicators of overfitting. A "test set" of the training dataset is typically set aside to prevent overfitting. The concept of underfitting refers to models that cannot be generalized to new data or model the training data. A machine learning model that is underfit is not suitable for training data and will result in poor performance.

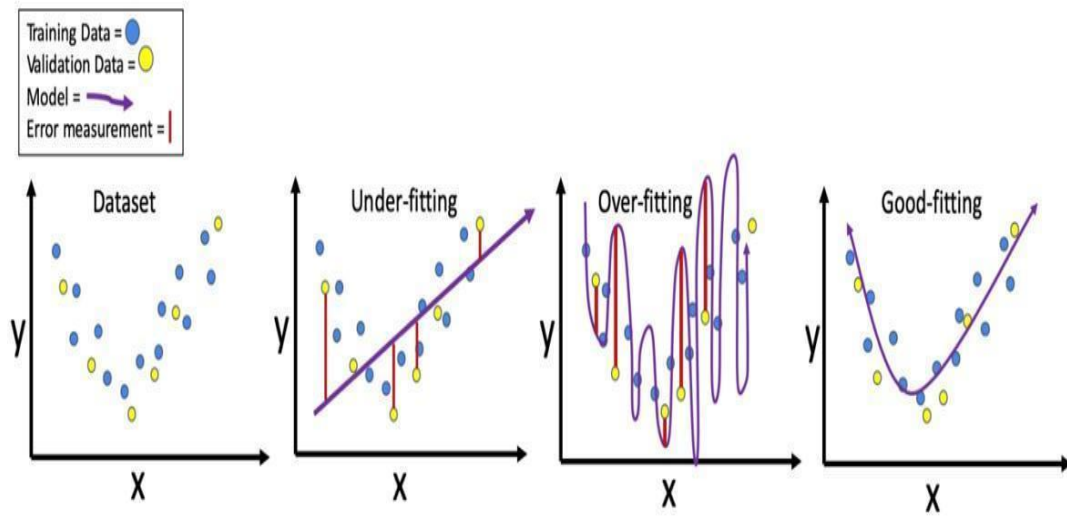


Figure 2.19: Underfitting, overfitting and good fitting examples (Kiourt et al., 2020)

2.7 RELATED WORKS

In an effort to eliminate the discontinuities across borders, various research methods have been developed. These methods often incorporate deep learning techniques and are summarized here.

Dong et al. presented an approach to eliminate the effect of image border, namely decomposing the image into two images: one being the periodic image and the other the smooth image. According to the study, when Fourier Transform is applied to an original image, the periodic image replaces it without affecting the image border, and in some circumstances, removing the image border can improve the success rate and accuracy of phase correlation-based image registration (Dong et al., 2019).

Puchala & Stokfiszewski proposed a structure of CNN for lossy compression of images intended as an extension of JPEG image compression standard where they were using high-quality human face images to train the CNN model and compared it with other methods such as discrete cosine transform, lobed orthogonal transform, modulated lobed transform, and Karhunen-Loeve transform, which also appear in the JPEG standard. Their

proposed model showed slightly better image quality and also enabled significant reduction of the blocking defects (Puchala & Stokfiszewski, 2021).

Using the recently developed Periodic Plus Smooth Decomposition Technique, Hoven et al. study shows that the edge discontinuities can be reliably removed through a simple efficient procedure. Edge artifacts are reduced by subtracting a smooth background based on boundary conditions set by the image's edges. Periodic Plus Smooth Decomposition preserves sharp reciprocal lattice peaks across the entire image area unlike traditional windowed Fourier transforms (Hovden et al., 2015).

Pielawski & Wählby introduced windowing methods from signal processing to effectively reduce edge-effects in patch-based image segmentation. Based on the assumption that the center of an image patch contains more contextual information than its sides and corners, they reconstruct the prediction by overlapping patches weighted according to 2-dimensional windows. Furthermore, they pointed out that by combining their proposed windowing method with any CNN model for segmentation, network predictions can be substantially improved without requiring additional modifications (Pielawski & Wählby, 2020).

Rasheed et al. (2020) presented a method for compressing high-resolution images using DFTs and a Matrix Minimization algorithm (MM) where each component (real and imaginary component) is quantized independently to increase the number of high frequency coefficients. In the end, LFC and HFC matrices are recoded using the MM algorithm and arithmetic coding. Data is decoded in reverse order using a sequential search algorithm. A matrix is then built by combining all decoded LFC and HFC values, followed by an inverse DFT. Research shows that their method yielded high compression ratios over 98% for structured light images with good image reconstruction (Rasheed et al., 2020).

Wan et al. (2020) developed training model named Feature consistency Training to minimize the distortions caused by the JPEG artifacts. With each iteration of the training model, raw images and their compressed versions of randomly sampled quality were added to the training process. As a result of adding feature consistency constraints to the

objective function, feature distortion in the representation space is minimized to learn robust filters (Wan et al., 2020).

Yuan & Hu (Yuan & Hu, 2019) defined a bit rate that requires high performance on predictive tasks that are invariant under a set of transformations, such as data augmentation. Based on this, unsupervised objectives for training neural compressors are designed. A generic image compressor was developed that achieved large rate savings without reducing the quality of downstream classification (Dubois et al., 2022).

A novel 12-layer deep convolutional network is presented with hierarchical skip algorithms for suppressing compression artifacts connections and a multi-scale loss function. The PSNR was improved by up to 1.79 dB over ordinary JPEG, and by up to 0.36 dB over the previous best ConvNet result. It also found that the network trained for a specific quality factor (QF) is resilient to the QF used to compress the input image (Cavigelli et al., 2017).

Model-based estimation techniques available to estimate the primary quantization matrix in double-compressed JPEG images work under specific conditions. A single CNN based estimation method was proposed that can be applied to a wide range of situations. By adapting a dense CNN network, the experimental results, the new method has several advantages, including: i) working under very general conditions, ii) improved performance in terms of MSE and accuracy when results are not aligned, iii) better spatial resolution due to being able to perform well even on small image patches (Niu et al., 2020).

2.8 COMPARISON OF PREVIOUS METHODS

Several techniques have been developed using CNN to minimize the artifacts caused by JPEG compression along with the improvement in compression. The list of techniques that have shown promising results are listed below.

Table 2.1: Comparison of various CNN techniques to minimize JPEG artifacts

Research	Result and Discussion
<p>Zhang et al. (Zhang et al., 2017) trained a network to reduce Gaussian noise and this network does not require the level of noise.</p>	<p>Their method not only produces desirable image denoising performance quantitatively and qualitatively, but also has a promising run time with GPU implementation.</p> <p>An appropriate investigation of CNN models for denoising of images with real complex noise and other general images restoration tasks should be conducted.</p>
<p>Ballé et al. (Ballé et al., 2017) used a nonlinear analysis transformation, a uniform quantizer, and a nonlinear synthesis transformation to optimize image compression.</p>	<p>The optimized method generally exhibits better rate-distortion performance than the standard JPEG and JPEG 2000 compression methods. And dramatic improvement in visual quality for all images at all bit rates, which is supported by objective quality estimates using MS-SSIM.</p> <p>Their method exhibits better rate-distortion performance than both JPEG and JPEG 2000 for most (but not all) test images, especially at the lower bit rates.</p>

<p>Mao et al. (Mao et al., 2016) proposed a deep encoding and decoding framework for image restoration. Skip connections were introduced to recover clean images and to tackle optimization difficulty for better performance.</p>	<p>Their proposed network achieves better performance than state of art methods on image noising and denoising.</p> <p>Further research can be conducted to explore the degradation of performance with different levels of corruption.</p>
<p>Dong et al. (Dong et al., 2015) trained a network to reduce JPEG compression artifacts.</p>	<p>Their method showed superior performance than the SA-DCT approach, both on the benchmark datasets and the real-world use case (i.e., Twitter).</p> <p>The research pointed out that the large filter size also helps to improve the performance and further research is needed.</p>
<p>Maleki et al. (Maleki et al., 2018) proposed a BlockCNN that performs both artifact removal and image compression.</p>	<p>The proposed algorithm performs better at low compression factors that don't predict high-frequency details.</p> <p>Research can be conducted to make it workable at higher bit rates.</p>

<p>Svoboda et al. (Svoboda et al., 2016) applied residual representation learning to define an easier task for the network with a combination of skip architecture, and symmetric weight initialization for artifacts reduction and better compression.</p>	<p>The network was compared with three different objectives— direct mapping, residual learning, and edge-preserving, and found out that residual learning provides the best reconstruction results.</p> <p>Further implementation can be conducted to other compression methods such as JPEG 2000, JPEG XR, or WebP and its performance.</p>
<p>Li et al. (Li, Wang, et al., 2020) proposed a single model convolutional neural network for conducting image artifacts removal of then JPEG-decoded images.</p>	<p>The proposed model was found to be beneficial to high-resolution image cases.</p> <p>The research was focused on JPEG compression with quality factors ranging from 1 to 60.</p>
<p>Santurkar et al. (Santurkar et al., 2017) used Deep Generative models to reproduce and remove image and video artifacts.</p>	<p>The research demonstrates that generative compression is orders-of-magnitude more resilient to bit error rates (e.g., from noisy wireless channels) than traditional variable-length coding schemes.</p> <p>The large image compression through still needs further research as the advancement of GAN continues.</p>

<p>Cavigelli et al. (Cavigelli et al., 2017) presented a novel 12-layer deep convolutional network with hierarchical skip algorithms for suppressing compression artifacts connections and a multi-scale loss function.</p>	<p>This result shows an improvement of up to 0.36 dB over the best previous ConvNet results with a PSNR of up to 1.79 dB in comparison to ordinary JPEG.</p> <p>The proposed approach does not exceed the PSNR-B value achieved by the L8 network for lower compression and requires further research.</p>
<p>By using the residual blocks with skip connections, Alexandre et al. (Alexandre et al.) proposed a lossy image compression system using the deep learning autoencoder structure.</p>	<p>The importance maps are generated by a separate neural net in the encoder, which is trained jointly by minimizing a weighted sum of mean squared errors, MS-SSIM, and a rate estimate. Despite outperforming JPEG significantly, the proposed model faces a performance gap relative to BPG suggesting ample room for improvement. Further research is needed on the impact of importance maps on subjective quality.</p>

2.9 SUMMARY

Recent decades have seen a tremendous increase in interest in digital imaging. As a result, many data compression techniques have been proposed, which are aimed at minimizing the amount of information used to represent images. Image compression is becoming more effective thanks to the advances in deep neural networks. Encoders and decoders have become a major trend in the development of CNN architectures to improve performance. In Image Processing and Computer Vision, a variety of approaches have been used to solve the compression artifacts. Results are promising with regards to performance, quality, and compression ratio for each method. However, it remains to be seen if these three factors can be met simultaneously.

CHAPTER 3

RESEARCH METHODOLOGY

3.1 INTRODUCTION

This chapter provides the design of the proposed framework to solve the border effect problem along with the research framework. A research plan is also setup which acts as a guide to accomplish this task.

3.2 RESEARCH FRAMEWORK

The research framework that is used in this research consists of multiple phases where each phase's output is an input to the next phase. Phase 1 is based on dataset collection and pre-processing. Phase 2 is about designing an efficient compression modal to reduce construction loss and is followed by Phase 3 where the developed model is evaluated using quality metrics: Mean Square Error (MSE), Peak Signal to Noise Ratio (PSNR), and Structural Similarity Index (SSIM). These phases are depicted in Figure 3.1.

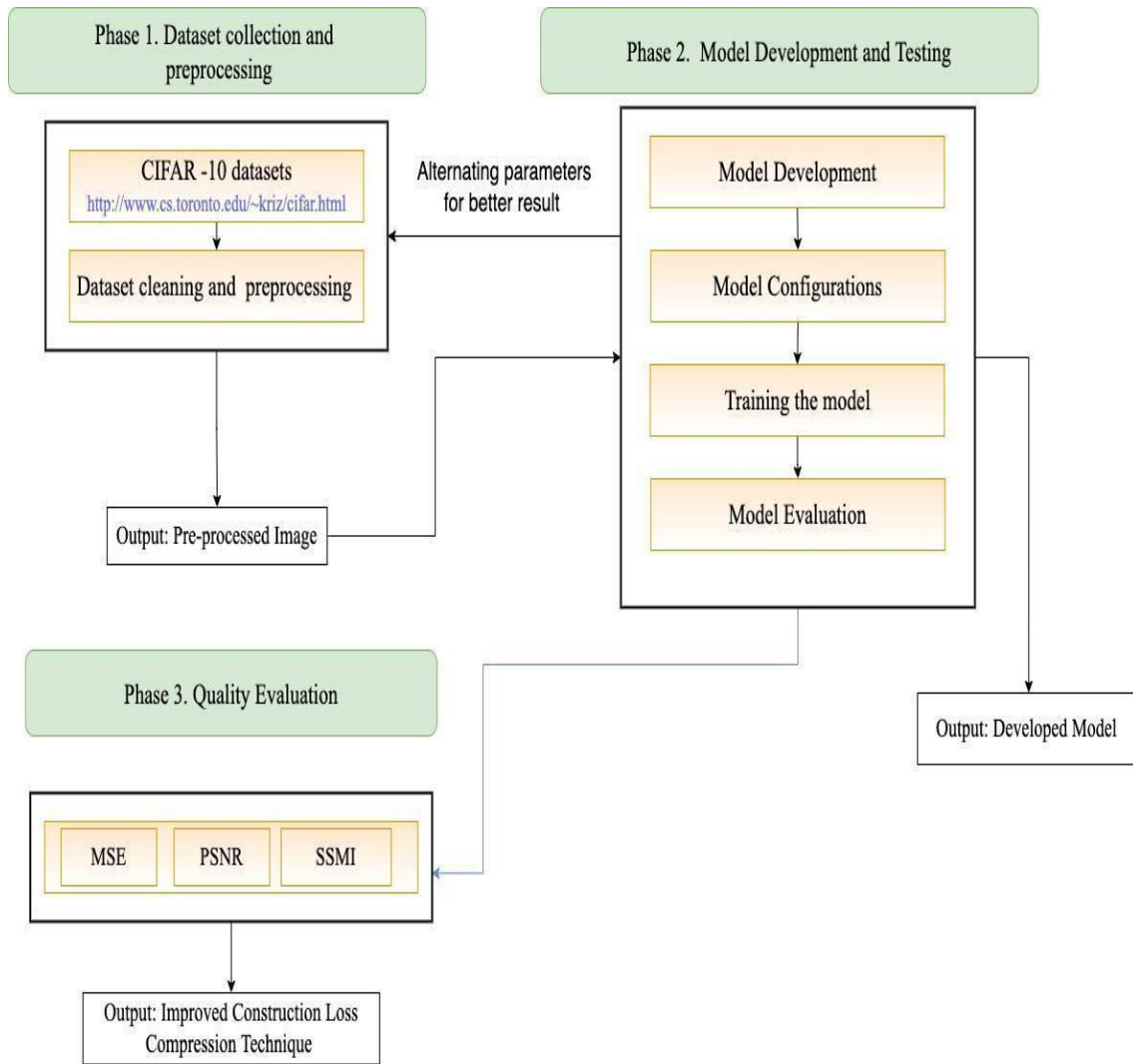


Figure 3.1: Research Framework

3.2.1 Data Collection and Preprocessing

There are many research organizations making data available on the web, and one of them is CIFAR-10 dataset which is most popular in computer vision tasks (DeVries & Taylor, 2017; Shorten & Khoshgoftaar, 2019; Xie et al., 2016). It is a publicly available dataset that consists of 60000 32x32 colour images arranged in 10 classes of 6000 images each. This includes 50000 training images and 10000 test images. The datasets are divided into two sets: training and testing datasets. In CIFAR10, there are several challenges, as the images vary in size, position, pose, and illumination (Ho-Phuoc, 2018), requiring preprocessing.

Simple random sampling will be used to represent the different sets of images in this research as it gives an equal probability of selecting a particular item (Gupta, 2021). Here, samples are only taken from the training images, the dataset used to train the algorithm. As the method aims to evaluate itself using both real and representative data, no training is done on holdout tests (Yadav & Shukla, 2016).

3.2.2 Model Development and Testing

To reduce the construction loss associated with JPEG compression, a method is proposed. This method comprises different steps such as encoding the image through an auto encoder network, decoding it, and calculating reconstruction loss. The proposed method is depicted in Figure 3.2.

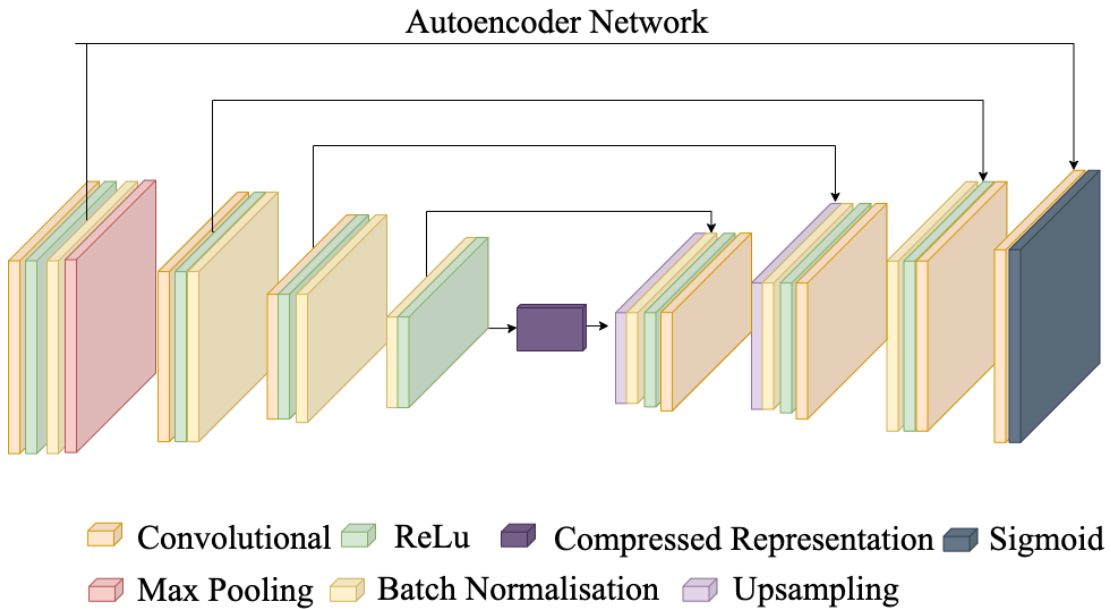


Figure 3.2: Proposed Model Architecture

Pre-processed data are fed into the CNN autoencoder neural network for the dimension reduction (Legrand et al., 2018). The projection to a lower dimensional space facilitates the identification of several hidden features (Bank et al., 2020). The encoders are trained along with the decoders without labels. Eight layers of convolution is used in the auto encoder network as it helps to reduce the computational costs and weight sharing. Batch Normalization is used to enable faster and more stable training of deep neural networks (Santurkar et al., 2018). The benefit of using a layer is, it allows similar

operations to be performed simultaneously. With more convolution kernels, the number of parameters increases linearly. As a result, the number of output channels also increases linearly and helps to reduce the artifacts caused by the gaussian noise (Audhkhasi et al., 2016). The computation time is also proportional to the size of the input channel and to the number of kernels.

Sigmoid activation function is used in the output layer, whose output is bound between 0 and 1 range, and can be prone to suffering from the vanishing gradient problem (Gustineli, 2022). To overcome this, ReLU is used as an activation function to increase to speed up the application and for better results (Dubey et al., 2021). Upsampling is also done to increase the spatial dimensions of the feature maps (Kundu et al., 2020).

For error correction, regularization options (Steck & Garcia, 2021) are explored with the model. The reconstructed results are analyzed and compared to the proposed method in terms of loss value, which measures the difference between the reconstructed image and desired image.

3.2.3 Quality Evaluation

Image quality can be compromised as a result of distortions during the acquisition and processing of images. Different metrics have been used to measure the quality of compressed results, including Mean Square Error (MSE), Peak Signal to Noise Ratio (PSNR), and Structural Similarity Index (SSMI) (Deshmukh, 2019). Measures such as MSE and PSNR are easy to calculate and applicable in most cases, but they do not always correlate to perceived quality and are often not normalized in display. To solve this problem, SSIM and feature similarity indexing method (FSIM) have been developed (Sara et al., 2019).

Different quality metrics such as MSE, PSNR and SSMI are used to test the quality of the reconstructed result. These metrics are discussed below.

A. Mean Square Error (MSE)

MSE is one of the most common image error metrics used to compare image compression quality. This metric represents the cumulative squared error between

the compressed and the original image. A lower MSE indicates a lower error. The Mean Square Error can be calculated with the following expression:

$$MSE = \frac{1}{mn} \sum_{i=1}^m \sum_{j=1}^n (x_{ij} - y_{ij})^2 \quad (3.19)$$

where:

m = number of rows in cover image

n = number of columns in cover image

x_{ij} = pixel value from cover image

y_{ij} = pixel value from stego image

B. Peak Signal to Noise Ratio (PSNR)

PSNR is an acronym for peak signal to noise ratio. Often, this ratio is used to compare the quality of an original image and a compressed image. In the quality degradation of image and video compression, PSNR values differ from 30 to 50 dB for 8-bit data and from 60 to 80 dB for 16-bit data. The accepted range of quality loss is about 20 - 25 dB for wireless transmission (Sara et al., 2019). An image that is compressed or reconstructed with a higher PSNR value will have better quality. The PSNR can be calculated with the following expression:

$$PSNR(x,y) = \frac{10\log_{10}[\max(\max(x), \max(y))]^2}{(x - y)^2} \quad (3.20)$$

where:

x = original reference image

y = restored or noisy image

C. Structural Similarity Index (SSMI)

A structural similarity index measures the similarity between images based on perception. Deterioration of an image is viewed here as a change in perception of structural information. The SSMI can be calculated with the following expression:

$$SSIM(x, y) = \frac{(2\mu_x\mu_y + c_1)(2\sigma_{xy} + c_2)}{(\mu_x^2 + \mu_y^2 + c_1)(\sigma_x^2 + \sigma_y^2 + c_2)} \quad (3.21)$$

where:

μ_x = average of x

μ_y = average of y

σ_x^2 = variance of x

σ_y^2 = variance of y

σ_{xy} = the covariance of x and y

$c_1 = (k_1L)^2$ variables to stabilize the division with weak denominator

$c_2 = (k_2L)^2$ variables to stabilize the division with weak denominator

L = dynamic range of the pixel-values

$k_1 = 0.01$ default

$k_2 = 0.03$ default

3.3 RESEARCH PLAN

To accomplish this research, a research plan is proposed, which is divided into three parts: preparation and planning, development and test, and report and presentation. In preparation and planning section activities like designing research question, literature review, data collection, designing a method, presenting a proposal, along with paper publication. In development and testing part activities like specifying detail requirements, developing a prototype, followed by the actual development of technique, which then is tested and validated. The incremental release will be performed here, corrections will be also performed wherever needed.

Different tasks will be performed in the report and presentation part. Results and measurements will be carried out followed by thesis writing, and paper publication. Training's, prerequisites courses will be taken to develop the skill sets requires to construct and support the research.

3.4 SUMMARY

A research framework is set up here which acts as guidance to complete the research that comprises different steps. The output of each step serves as an input for the next steps. Ways of collecting data, sampling, and pre-processing are described in the data collection and preprocessing section. Later a method is proposed that feeds the pre-processed data and uses a CNN autoencoder neural network to output a result. Various industry-standard quality metrics are proposed to measure its performance. A research plan is also developed to accomplish this research.

CHAPTER 4

IMPLEMENTATION

4.1 INTRODUCTION

This chapter describes the steps taken to implement the model. First, the process flow is mapped out and then developed. Later, the developed model is tested with different quality metrics. Figure 4.1 depicts the process flow diagram. Table 4.3, Table 4.2 and Table 4.3 depicts composition of encoder layer, decoder layer and model respectively.

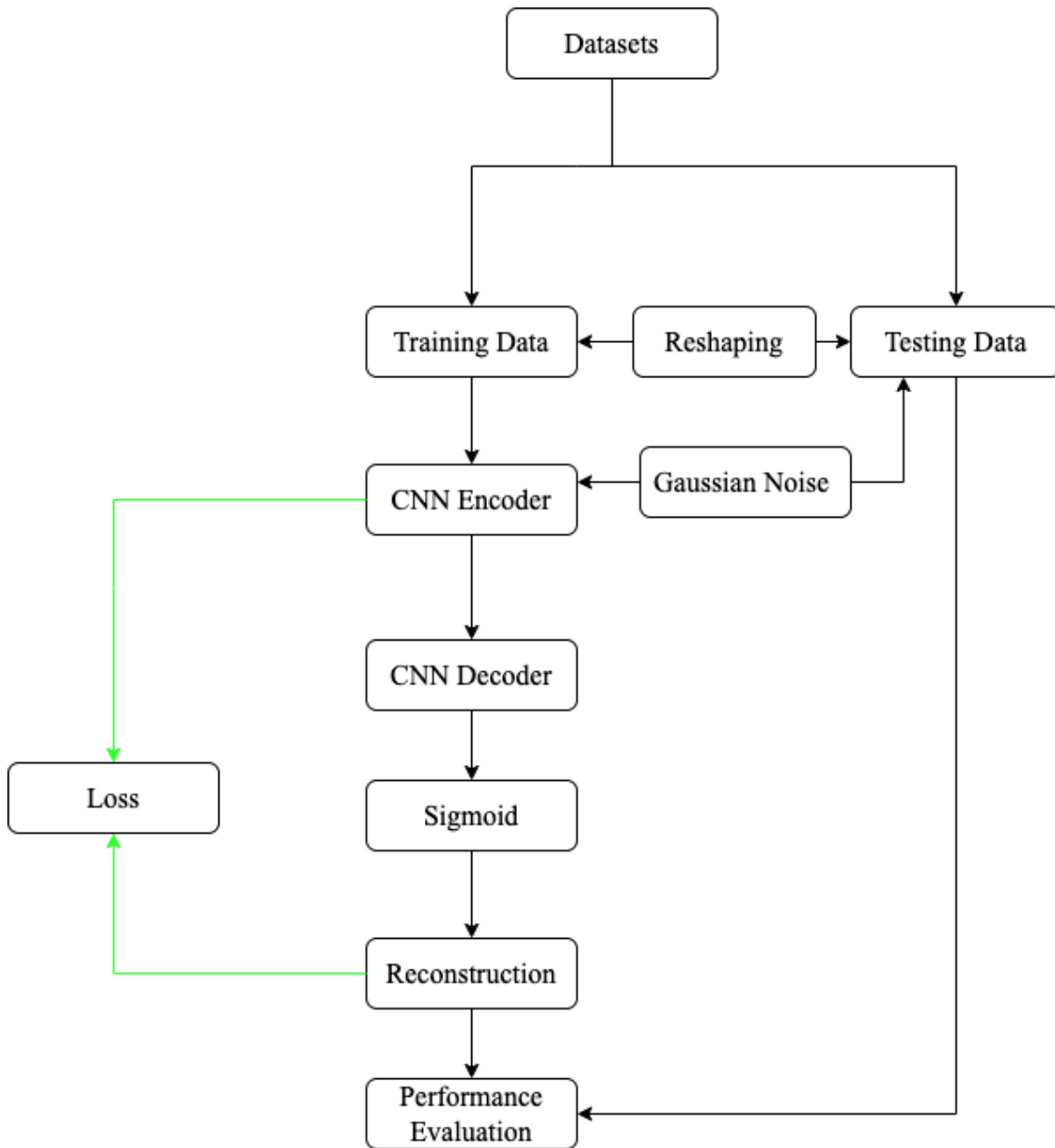


Figure 4.1: Process Flow Chart

Table 4.1: Composition of encoder layer

Layer (type)	Filters	Kernel Size	Strides	Padding	Activation
Conv2D	32	3	1	Same	ReLU
Batch Normalization					
Max Pooling					
Conv2D	16	3	1	Same	ReLU
Batch Normalization					
Conv2D	8	3	1	Same	ReLU
Batch Normalization					
Conv2D	8	3	1	Same	ReLU

Table 4.2: Composition of decoder layer

Layer (type)	Filters	Kernel Size	Strides	Padding	Activation
Conv2D	32	3	1	Same	ReLU
Batch Normalization					
UpSampling					
Conv2D	16	3	2	Same	ReLU
Batch Normalization					
Up Sampling					
Conv2D	16	3	1	Same	ReLU
Batch Normalization					
Conv2D	3	3	1	Same	Sigmoid

There are four convolutional layers in the encoder layer, each of which uses ReLU as an activation function. Each layer has a filter size of 32, 16, 8, and 8. In the first three layers, batch normalization is performed. After batch normalization in the first layer, maximum pooling is performed.

The decoder layer consists of four convolutional layers. Besides the last layer, each layer has ReLU as an activation function. Filter sizes are 32, 16, 16, and 3 for each layer. The first three layers are normalized in batch, and the first two layers are upsampled. The last layer uses sigmoid as an activation function. Each layer of the encoder and decoder layer are stacked together, and a sequential model is built.

Table 4.3: Composition of model

Layer (type)	Output Shape	Param
InputLayer	[(None, 32, 32, 3)]	0
Sequential	(None, 16, 16, 8)	7488
Sequential	(None, 32, 32, 3)	9587

4.2 CODING

Due to its simplicity, consistency, access to AI and machine learning (ML) libraries, flexibility, and platform independence, this model is built using Keras, TensorFlow's high-level API for constructing deep learning models. For measuring quality, scikit-image package is used. Scikit-image is a collection of image processing algorithms (van der Walt et al., 2014). At first, all the required libraries are imported, which is then followed by loading training and testing datasets. Each pixel in these images will have a value ranging from 0 to 255. To make the model efficient and faster, small centered values near 0 are taken.


```

from keras.datasets import cifar10
from keras.layers import Input, Conv2D, MaxPooling2D,
    BatchNormalization, UpSampling2D
from keras.models import Model, Sequential
import numpy as np # linear algebra
import matplotlib.pyplot as plt #visualization library

#load training and test dataset from cifar10 dataset
(X_train, _), (X_test, _) = cifar10.load_data()
# Unit normalizing
X_train = X_train.astype('float32')/255
X_test = X_test.astype('float32')/255
# Reshaping training and test datasets
X_train = X_train.reshape(len(X_train), X_train.shape[1], X_train.
    shape[2], 3)
X_test = X_test.reshape(len(X_test), X_test.shape[1], X_test.
    shape[2], 3)
print(X_train.shape) #(50000, 32, 32, 3)
print(X_test.shape) #(10000, 32, 32, 3)

```

An Autoencoder Network is then constructed, where the encoder is used to represent the data in the simplest possible way. This is done by extracting the most prominent features and presenting them in a way that the decoder can understand. Whereas the decoder learns to read compressed code representations and generate them based on those representations. Each convolution layer uses ReLU as an activation function to increase the computational speed and also removes the vanishing gradient problem.

The Batch Normalization ensures that the gradient signal is heard by preventing gradients from becoming too small. Upsampling is done in the decoder network using UpSampling2D, which scales up an image by using nearest neighbour or bilinear upsampling. At the end of the layer sigmoid is used as an activation function, making sure that the output is between 0 and 1.

```

def build_autoencoder(img_shape):
    # The encoder network
    encoder = Sequential()
    encoder.add(Conv2D(32, kernel_size=3, strides=1, padding='
        same', activation='relu', input_shape=img_shape)) # 32
        x32x32
    encoder.add(BatchNormalization()) # 32x32x32
    encoder.add(MaxPooling2D(2, padding='same')) # 16x16x32
    encoder.add(Conv2D(16, kernel_size=3, strides=1, padding='
        same', activation='relu')) # 16x16x16
    encoder.add(BatchNormalization()) # 16x16x16
    encoder.add(Conv2D(8, kernel_size=3, strides=1, padding='
        same', activation='relu')) # 16x16x8
    encoder.add(BatchNormalization()) # 16x16x8
    encoder.add(Conv2D(8, kernel_size=3, strides=1, padding='
        same', activation='relu')) # 16x16x8

    # The decoder network
    decoder = Sequential()
    decoder.add(Conv2D(32, kernel_size=3, strides=1, padding='
        same', activation='relu')) # 16x16x32
    decoder.add(BatchNormalization()) # 16x16x32
    decoder.add(UpSampling2D()) # 32x32x32
    decoder.add(Conv2D(16, kernel_size=3, strides=2, padding='
        same', activation='relu')) # 16x16x16
    decoder.add(BatchNormalization()) # 16x16x16
    decoder.add(UpSampling2D()) # 32x32x16

```

```

decoder.add(Conv2D(16, kernel_size=3, strides=1, padding='
    same', activation='relu')) # 32x32x16
decoder.add(BatchNormalization()) # 32x32x16
decoder.add(Conv2D(3, kernel_size=1, strides=1, padding='
    same', activation='sigmoid')) # 32x32x3

return encoder, decoder

IMG_SHAPE = X_train.shape[1:] #(32, 32, 3)
input_img = Input(shape=IMG_SHAPE) #create image input
encoder, decoder = build_autoencoder(IMG_SHAPE)
code = encoder(input_img) #encode image
reconstruction = decoder(code) #decoder image
autoencoder = Model(input_img, reconstruction)

```

The autoencoder model is then compiled using adam optimizer and the loss is measured using MSE.

```

#create autoencoder model and compile it using adam optimizer,
    and measure loss using mean_squared_error
autoencoder.compile(optimizer='adam', metrics=['accuracy'], loss
    ='mean_squared_error')
print(autoencoder.summary())

```

Model: "model"

Layer (type)	Output Shape	Param #
input_1 (InputLayer)	[(None, 32, 32, 3)]	0
sequential (Sequential)	(None, 16, 16, 8)	7488
sequential_1 (Sequential)	(None, 32, 32, 3)	9587

=====
Total params: 17,075
Trainable params: 16,835
Non-trainable params: 240
=====
None

As, the comparison is happening between constructed and original images, both x and y are equal to X_{train} . Here $epochs$ define the number of times the training data to be passed through the model and the $validation_data$ is the validation set use to evaluate the model after training. Later, the loss and accuracy between training and test data are calculated.

trains model with fixed number of epochs

#trains model with fixed number of epochs

```
history = autoencoder.fit(x=X_train , y=X_train , epochs=15,  
                          validation_data=[X_test , X_test ])
```

To reduce overfitting regularization options such as early stopping is carried out. ModelCheckpoint callback is used here to save weights and models (in a checkpoint file) at intervals, so that they can be loaded later to continue the training.

```

# simple early stopping
es = EarlyStopping(monitor='val_loss', mode='min', patience=200)

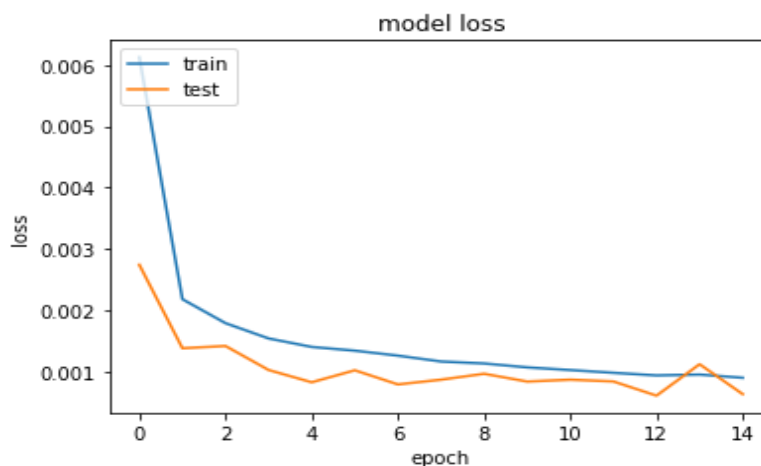
# modelcheckpoint stores the best weights in a model with an
  interval for future use
mc = ModelCheckpoint('auto_encoder_model.h5', monitor='
  val_accuracy', mode='max', verbose=1, save_best_only=True)

history = autoencoder.fit(x=X_train, y=X_train,
  validation_data=[X_test, X_test], epochs=3,
  callbacks=[es, mc])

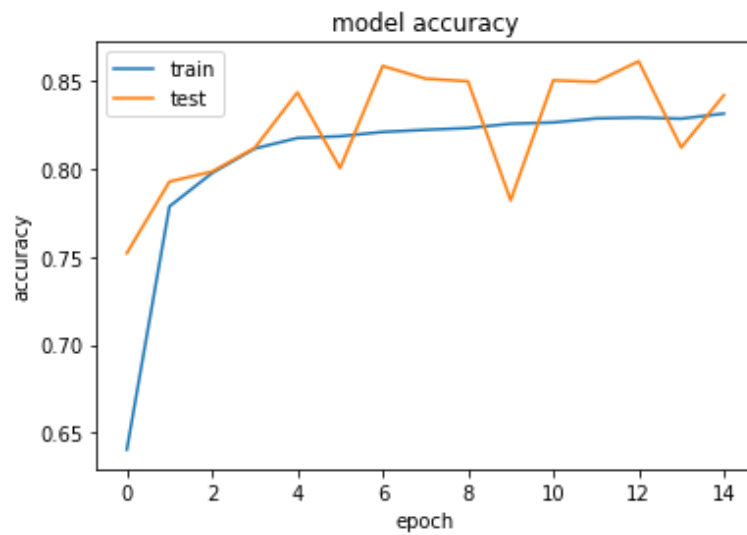
#loss between train and test data based on training results
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()

#evaluate the model
autoencoder.evaluate(X_test, Y_test)
#loss: 8.6279e-04 - accuracy: 0.8722

```



```
#accuracy between train and test data based on training results
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()
```



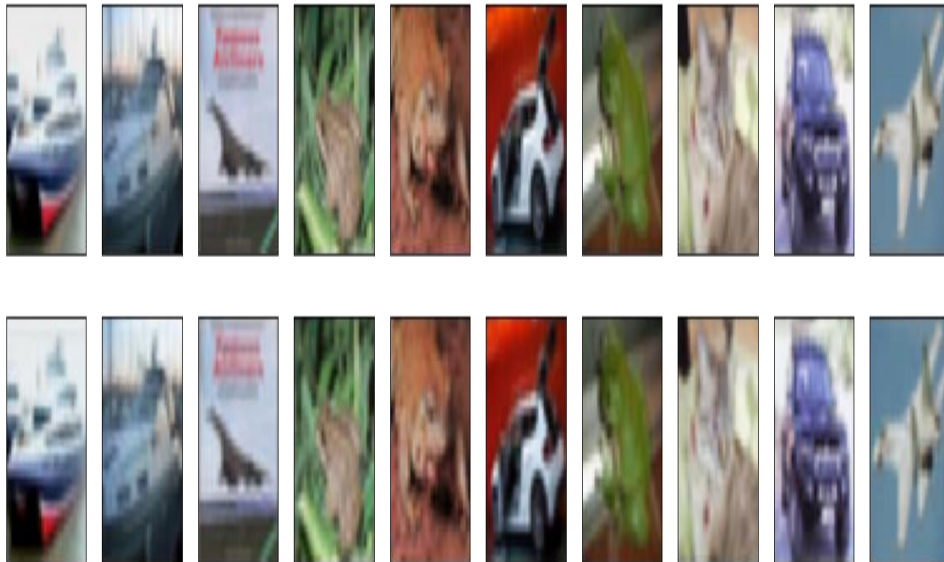
A random set of 10 images is picked from the test set and is compared with the original and reconstructed images.

```

decoded_imgs = autoencoder.predict(X_test)
n = 10
plt.figure(figsize=(20, 4))
for i in range(1, n + 1):
    # Display original
    ax = plt.subplot(2, n, i)
    plt.imshow(X_test[i].reshape(32, 32,3))
    plt.gray()
    ax.get_xaxis().set_visible(False)
    ax.get_yaxis().set_visible(False)

    # Display reconstruction
    ax = plt.subplot(2, n, i + n)
    plt.imshow(decoded_imgs[i].reshape(32, 32,3))
    plt.gray()
    ax.get_xaxis().set_visible(False)
    ax.get_yaxis().set_visible(False)
plt.show()

```



The compressed results are then compared with different state of the art quality metrics such as SSIM, MSE and PSNR with the help of skimage library.

```
from skimage.metrics import structural_similarity as ssim
from skimage.metrics import mean_squared_error as mse
from skimage.metrics import peak_signal_noise_ratio as psnr

def calculate_mse(imageA, imageB):
    return mse(imageA, imageB)

def calculate_psnr(imageA, imageB):
    return psnr(imageA, imageB)

def calculate_ssim(imageA, imageB):
    return ssim(imageA, imageB, multichannel=True)

def compression_ratio(original, compressed):
    return (original.size) / (compressed.size)

def compare_images(original, compressed, title):
    # compute the mean squared error, peak signal noise ratio
    and structural similarity

    fig, axes = plt.subplots(nrows=1, ncols=2, figsize=(10, 4),
                             sharex=True, sharey=True)
    ax = axes.ravel()
    cal_mse = calculate_mse(original, compressed)
    cal_psnr = calculate_psnr(original, compressed)
    cal_ssim = calculate_ssim(original, compressed)
    # setup the figure
    ax[0].imshow(original, cmap=plt.cm.gray)
    ax[0].set_xlabel(f'MSE: {calculate_mse(original, original):.3f}, SSIM: {calculate_ssim(original, original):.3f}')
    ax[0].set_title('Original image')

    ax[1].imshow(compressed, cmap=plt.cm.gray)
```



```

ax[1].set_xlabel(f'MSE: {cal_mse:.3f}, PSNR: {cal_psnr:.3f},
                SSIM: {cal_ssim:.3f}')
ax[1].set_title('Compressed image')

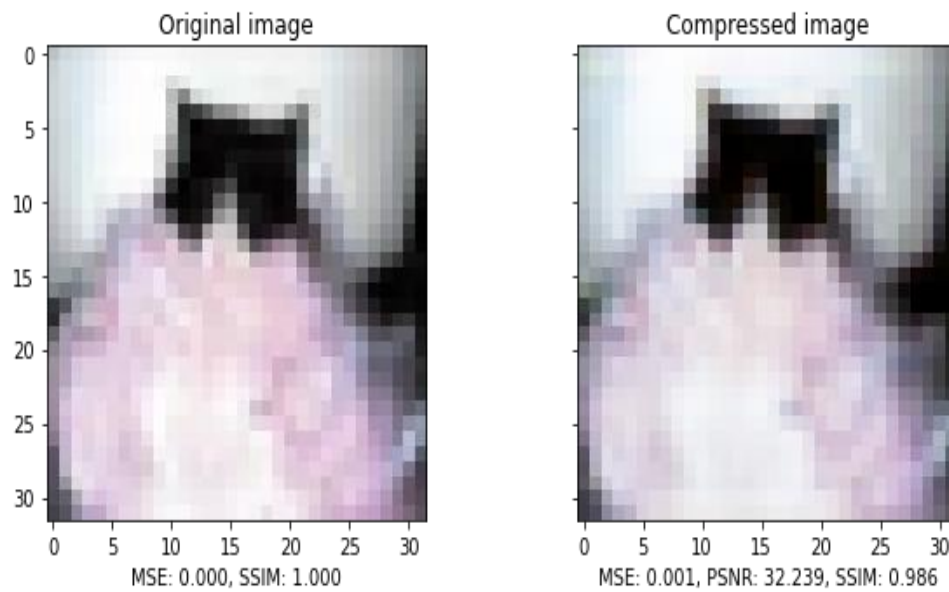
plt.show()

# test quality results with random images
# initialize the figure
fig = plt.figure("Images")
index = np.random.randint(len(X_test))
original = np.squeeze(X_test[index])
compressed = np.squeeze(decoded_imgs[index])
images = ("Original", original), ("Compressed", compressed)
# loop over the images
for (i, (name, image)) in enumerate(images):
    # show the image
    ax = fig.add_subplot(1, 3, i + 1)
    ax.set_title(name)
    plt.imshow(image, cmap = plt.cm.gray)
    plt.axis("off")
# show the figure
plt.show()

#compare image quality
compare_images(original, compressed, "Original vs. Compressed")

#calculate compression ratio
cr = "{:.4f}".format(compression_ratio(original, compressed))
print(cr)

```



To learn a generalizable encoding and decoding model, the model needs to be sensitive enough to recreate the original observation, yet insensitive enough to the training data. For that noise is added to the input data, but the uncorrupted data remains as a target output.

```
# add some gaussian noise
def apply_gaussian_noise(X, sigma=0.1):
    noise = np.random.normal(loc=0.0, scale=sigma, size=X.shape)
    return X + noise
```

```

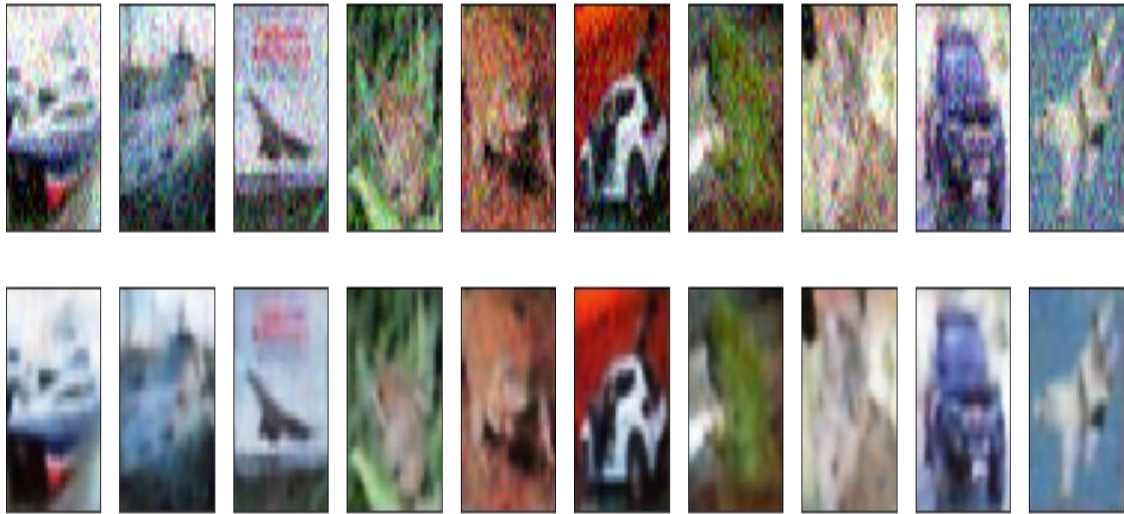
for i in range(4):
    print("Epoch %i/10, Generating corrupted samples..."%(i+1))
    X_train_noise = apply_gaussian_noise(X_train)
    X_test_noise = apply_gaussian_noise(X_test)

    # continue to train our model with new noise-augmented data
    autoencoder.fit(x=X_train_noise, y=X_train, epochs=1,
                    validation_data=[X_test_noise, X_test])

X_test_noise = apply_gaussian_noise(X_test)
n = 10
decoded_imgs = autoencoder.predict(X_test_noise)
plt.figure(figsize=(20, 4))
for i in range(1, n + 1):
    # Display original
    ax = plt.subplot(2, n, i)
    plt.imshow(X_test_noise[i])
    plt.gray()
    ax.get_xaxis().set_visible(False)
    ax.get_yaxis().set_visible(False)

    # Display reconstruction
    ax = plt.subplot(2, n, i + n)
    plt.imshow(decoded_imgs[i])
    plt.gray()
    ax.get_xaxis().set_visible(False)
    ax.get_yaxis().set_visible(False)
plt.show()

```



An image is taken random from the testing sample set and DFT compression is applied. At first all the necessary libraries are imported, then it is converted into gray scale image.

```
import numpy as np
import matplotlib.pyplot as plt
import scipy
from skimage import io
from numpy import pi
from numpy import sin
from numpy import zeros
from numpy import r_
from scipy import signal
import matplotlib.pyplot as pylab
```

Then the image is divided into several 8x8 tiles. These 8x8 2D image are then converted into 64x1 one dimensional images. DFT is the applied to 64x1 image to get the amplitude and phase at different frequencies. The smaller items in the DFT results are excluded as they are insensitive to human eyes.

```

#image is read from file path
file_path = "Car.png"
im = io.imread(file_path , True)
imsize = im.shape

dft = zeros(imsize , dtype='complex');
im_dft = zeros(imsize , dtype='complex');

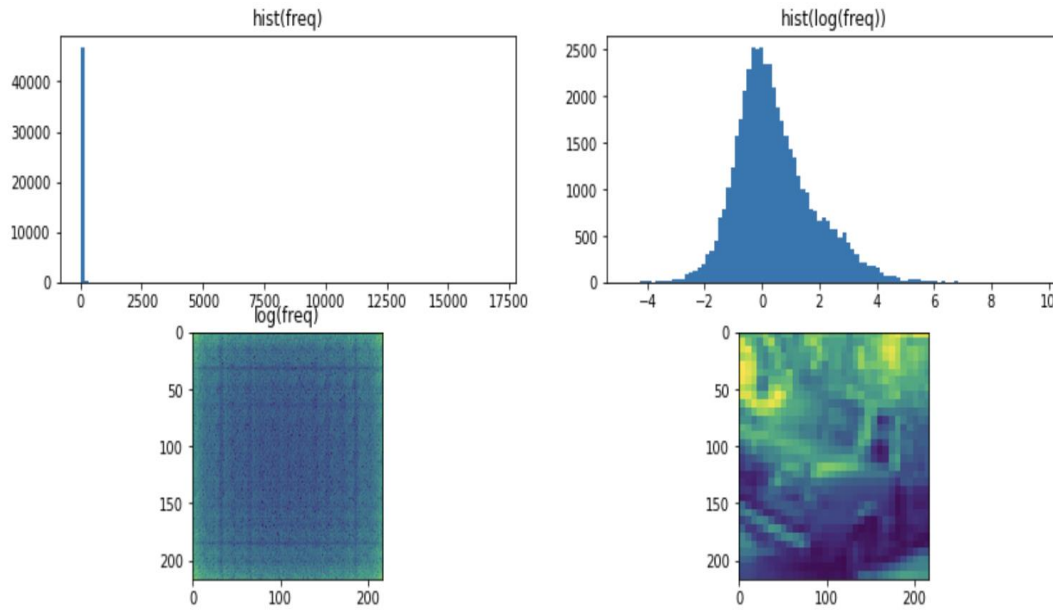
# 8x8 DFT
for i in r_[:imsize[0]:8]:
    for j in r_[:imsize[1]:8]:
        dft[i:(i+8),j:(j+8)] = np.fft.fft2( im[i:(i+8),j:(j+8)]
        )

freq = np.fft.fft2(im)
freq = np.abs(freq)

plt.figure()
fig , ax = plt.subplots(nrows=2, ncols=2, figsize=(14, 6))
ax[0,0].hist(freq.ravel(), bins=100)
ax[0,0].set_title('hist(freq)')
ax[0,1].hist(np.log(freq).ravel(), bins=100)
ax[0,1].set_title('hist(log(freq))')
ax[1,0].imshow(np.log(freq), interpolation="none")
ax[1,0].set_title('log(freq)')
ax[1,1].imshow(im, interpolation="none")
plt.show()

```

Then, the histogram frequency, log frequency and histogram of log frequency are plotted for analysis.



```

# Thresh
thresh = 0.013
dft_thresh = dft * (abs(dft) > (thresh*np.max(abs(dft))))

percent_nonzeros_dft = np.sum( dft_thresh != 0.0 ) / (imsize[0]*
    imsize[1]*1.0)
print("Keeping only %f%% of the DFT coefficients" % (
    percent_nonzeros_dft*100.0))

# 8x8 iDFT
for i in r_[:imsize[0]:8]:
    for j in r_[:imsize[1]:8]:
        im_dft[i:(i+8),j:(j+8)] = np.fft.ifft2( dft_thresh[i:(i
            +8),j:(j+8)] )

plt.figure()
ax = plt.gca()
ax.invert_yaxis()
plt.axis('off')
plt.imshow(abs(im_dft), cmap='gray')
plt.savefig('compressed', bbox_inches = 'tight', pad_inches
    = 0)

```



The output image after DFT compression is measured using PSNR and MSE.

```
def calculate_psnr(img1, img2, max_value=255):  
    """Calculating peak signal-to-noise ratio (PSNR) between  
        two images."""  
    mse = np.mean((np.array(img1, dtype=np.float32) - np.array(  
        img2, dtype=np.float32)) ** 2)  
    if mse == 0:  
        return 100  
    return 20 * np.log10(max_value / (np.sqrt(mse)))  
  
def rmse(a, b):  
    # Root MSE (Mean Squared Error)  
    return np.sqrt(np.mean(np.square(np.subtract(a, b, dtype=np.  
        float32))))  
  
calculate_psnr(im, abs(im_dft))  
rmse(im, abs(im_dft))
```

4.3 TESTING

A set of 10000 images is taken to test the proposed model. At first, the test datasets are normalized and reshaped. Then it is fed into the neural network. The model complies with different settings using adam optimizer. Here, accuracy is used as a quality metric, and loss is measured in terms of mean squared error.

The loss and accuracy of the model are measured, and the best weights are saved in an external file. Regularization techniques are applied for optimization and its performance is evaluated. Weights with the best performance are added to the same model.

To check the denoising capability of the model, gaussian noise with low variance is added to the train and test the dataset. The model is trained, and its best value is added to the model. Quality metrics such as PSNR, MSE, and SSMI are used to measure the measure model's output quality.

4.4 SUMMARY

Various activities such as mapping the flow, coding, and testing are carried out to implement the model. The model uses an autoencoder neural network and is built using Keras. Encoder network extracts the most prominent features of the original data in the smallest possible form and stores it in a way that decoder understands it. Noise is added to the input to make it more generalizable encoding and decoding model. Various state of the art metrics is used to measure the quality of reconstructed image.

CHAPTER 5

RESULTS AND ANALYSIS

5.1 INTRODUCTION

This chapter discusses the results of the proposed model through various aspects. Different quality metrics are used to measure quality of reconstructed image. The proposed method is evaluated with different settings and its performance is also measured.

5.2 RESULTS

The training dataset is run through the model in an incremental fashion. While training the model, Adam is used as an optimizer and loss is calculated in terms of Mean Square Error (MSE) as it has better generalization performances than the Cross Entropy (CE) loss (Hui & Belkin, 2020). Loss values and accuracy scores while training at different iterations is shown below along with the test results.

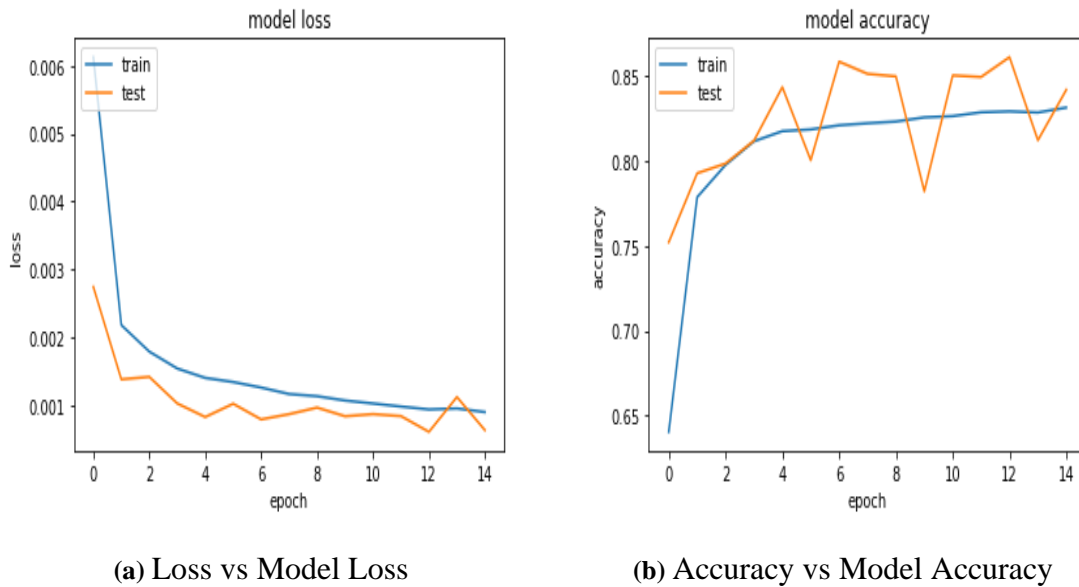


Figure 5.1: Loss vs Model & Accuracy vs Model Accuracy after 15 epochs

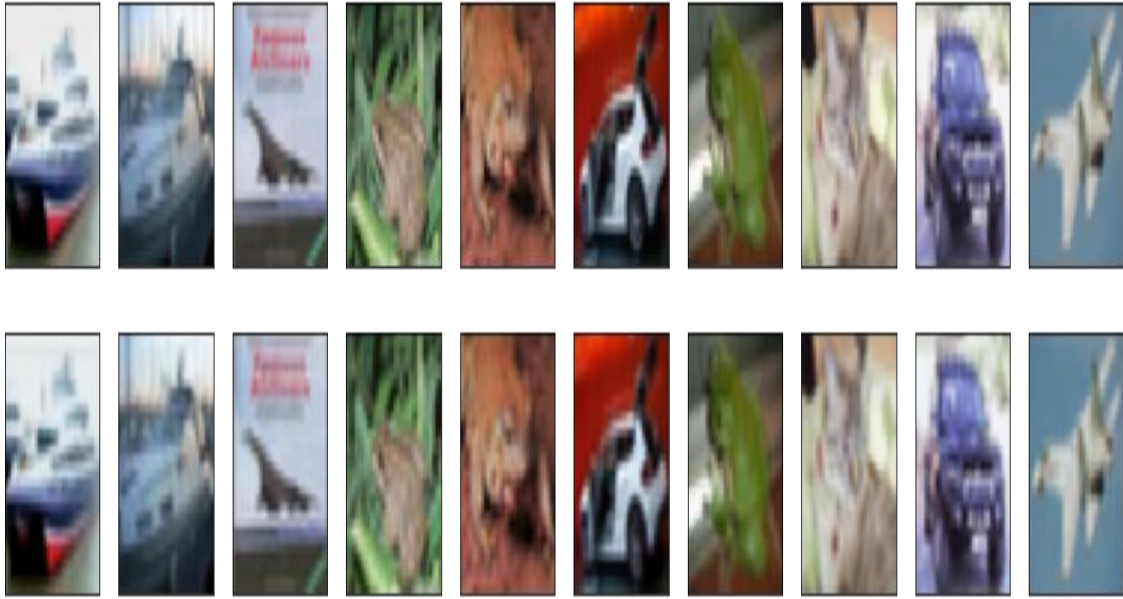


Figure 5.2: Original and reconstructed images over 15 epoches

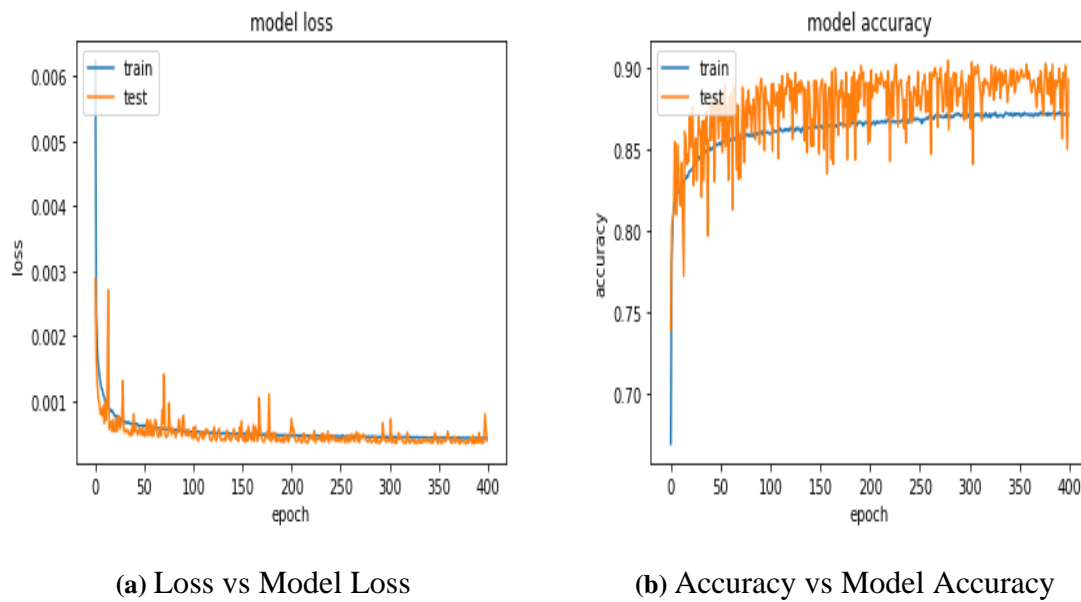


Figure 5.3: Loss vs Model & Accuracy vs Model Accuracy after training 400 epochs

Certain spikes can be seen in Figure 5.6 (a). Here, the loss first decreases, increases, and decreases at last. These spikes are often encountered when training with high learning rates, high order loss functions or small batch sizes (Ede & Beanland, 2020). The batch

size is reduced, and early stopping is added to the model for to avoid overfitting (Rice et al., 2020). A checkpoint is also added to save the best value weights obtained while training. The maximum accuracy value has reached up to 0.90104 within 200 epochs training.

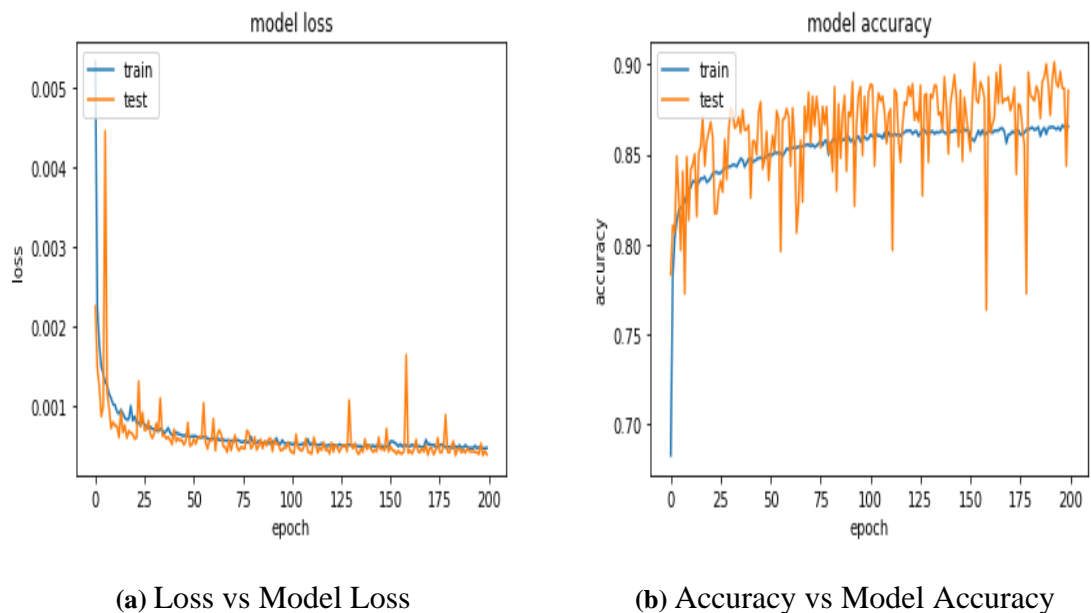


Figure 5.4: Loss vs Model & Accuracy vs Model Accuracy after training 200 epochs

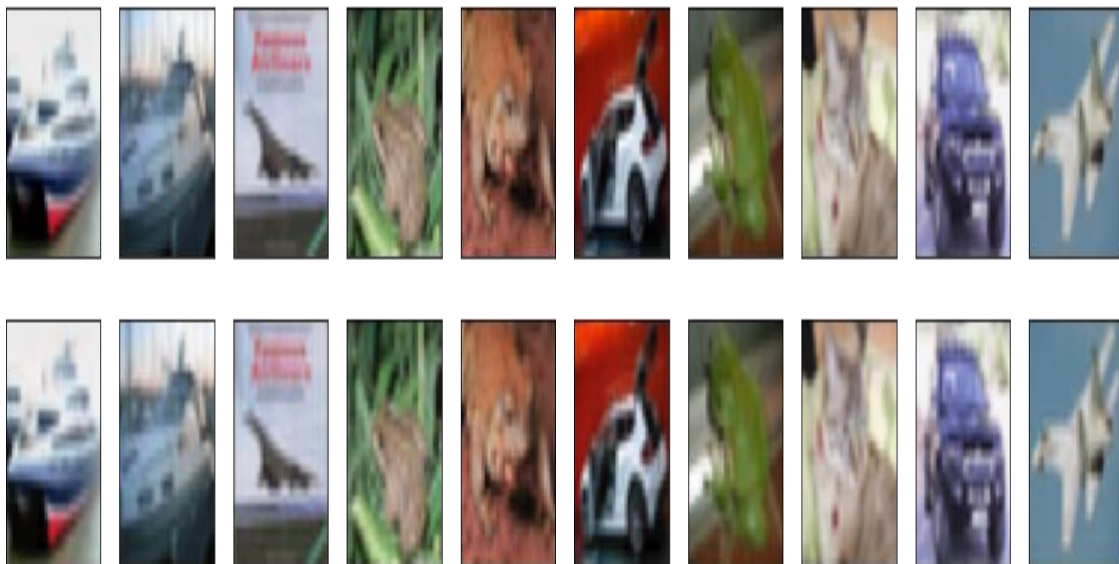
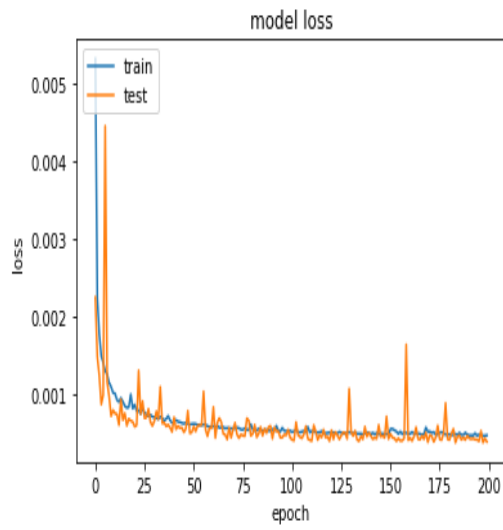
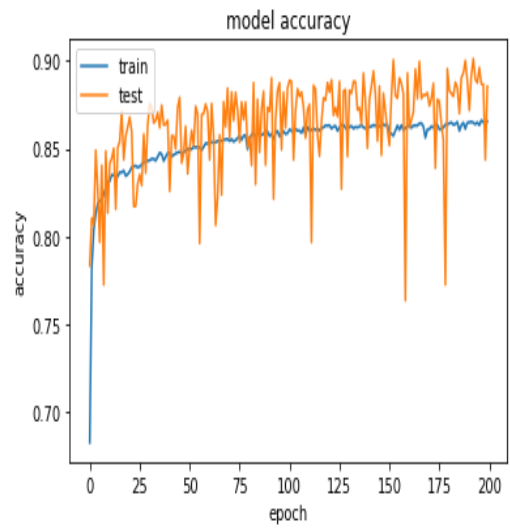


Figure 5.5: Original and reconstructed images over 200 epochs



(a) Loss vs Model Loss



(b) Accuracy vs Model Accuracy

Figure 5.6: Loss vs Model & Accuracy vs Model Accuracy after training 400 epochs

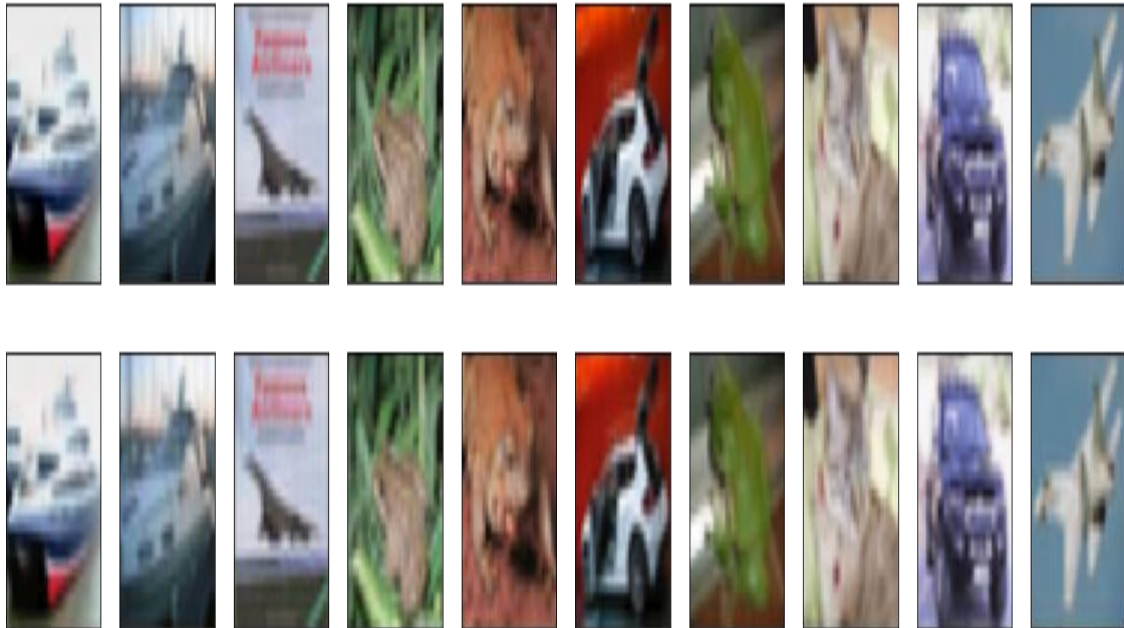


Figure 5.7: Original and reconstructed images over 400 epochs

5.3 EVALUATION OF THE PROPOSED METHOD

The proposed method is evaluated with different settings and its loss is measured. MSE is used a loss function to measure the loss. Table 5.1 shows the comparison of proposed model with different settings.

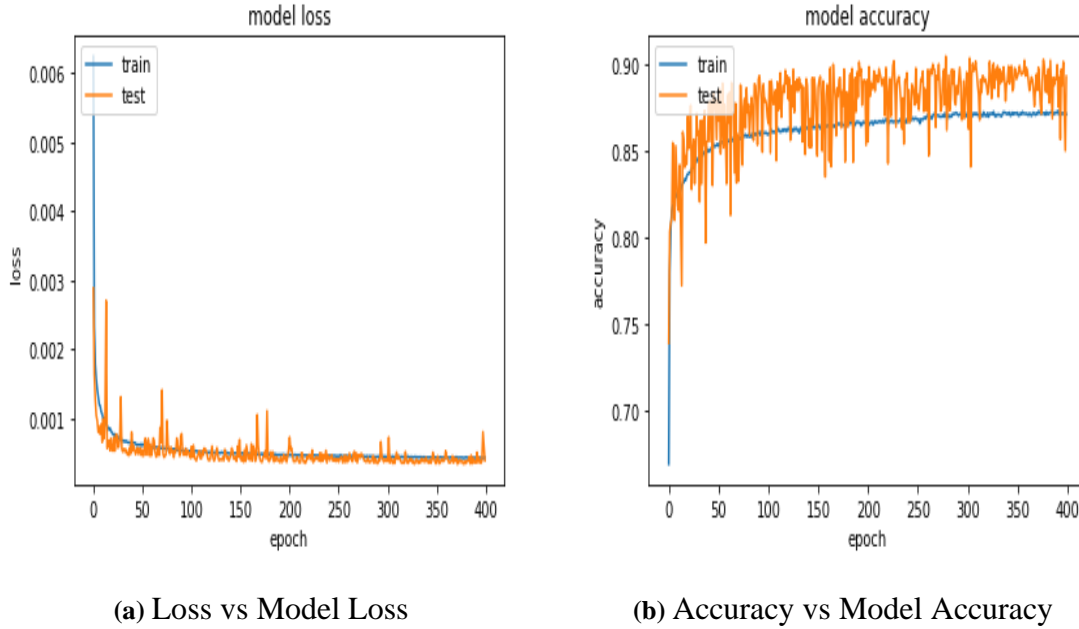


Figure 5.8: Loss vs Model & Accuracy vs Model Accuracy after 15 epochs

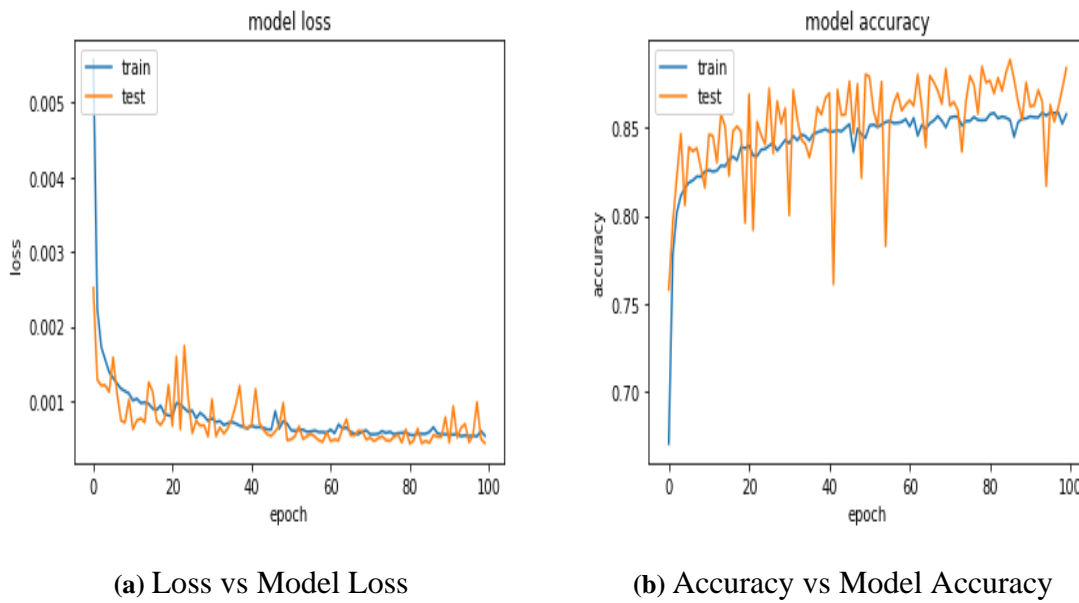
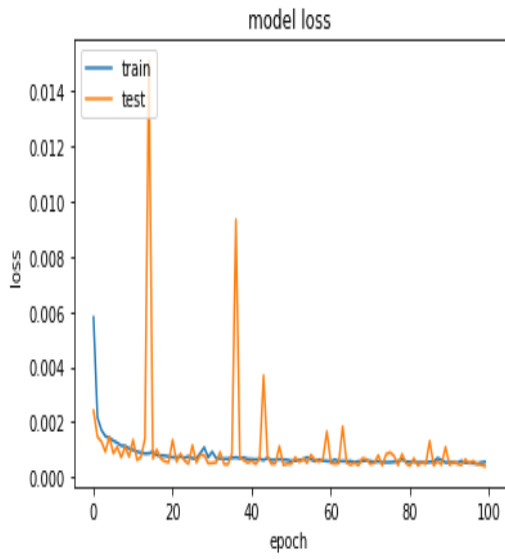
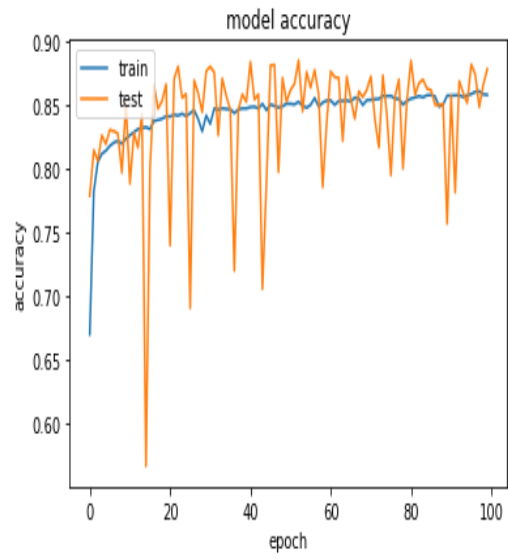


Figure 5.9: Loss vs Model & Accuracy vs Model Accuracy after 100 epochs

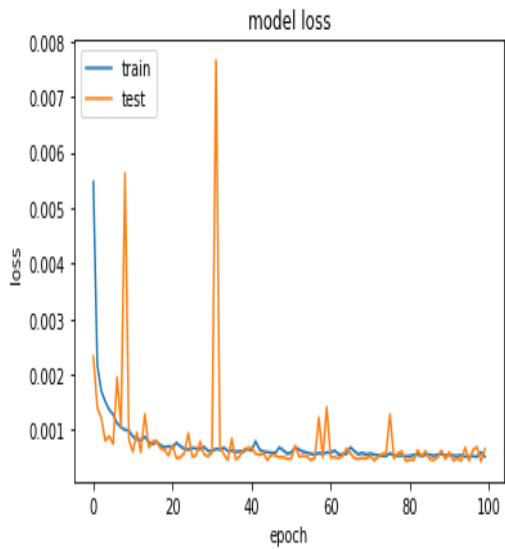


(a) Loss vs Model Loss

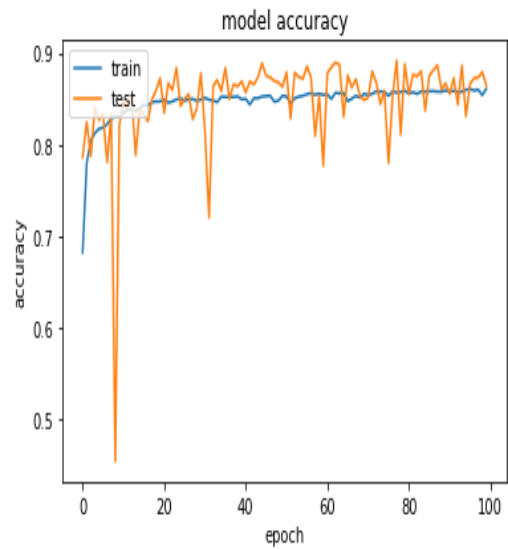


(b) Accuracy vs Model Accuracy

Figure 5.10: Loss vs Model & Accuracy vs Model Accuracy after 100 epochs using batch size of 32

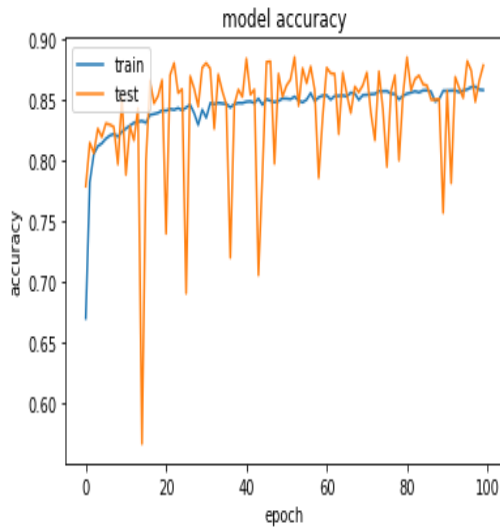


(a) Loss vs Model Loss

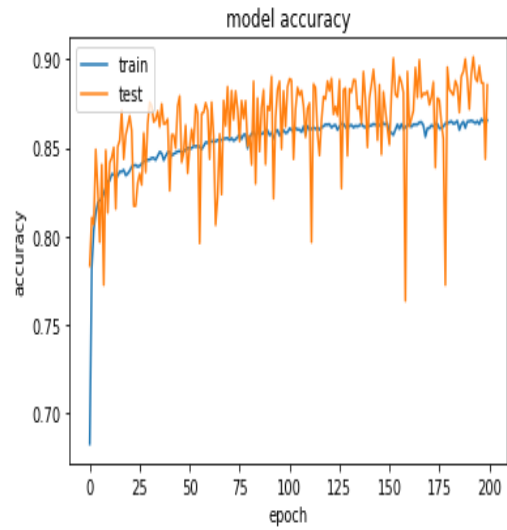


(b) Accuracy vs Model Accuracy

Figure 5.11: Loss vs Model & Accuracy vs Model Accuracy after 100 epochs with regularization



(a) Loss vs Model Loss



(b) Accuracy vs Model Accuracy

Figure 5.12: Loss vs Model & Accuracy vs Model Accuracy after training 200 epochs with regularization

Table 5.1: Comparison of proposed model with different settings

Model (variations)	Loss	Accuracy	Epoches
Model	0.162	80.236	15
Model	0.092	85.352	100
Model	0.089	82.307	200
Model with batch size 32	0.13	84.269	200
Model with normalization	0.07	85.94	100
Model with normalization	0.08	87.22	200
Model with normalization(200 epoches) and noise	0.111	56.74	30

From the comparison, the model shows high accuracy with regularization and has higher loss with low training. With the increasing number of training variation on loss and accuracy can be seen. The model shows low accuracy with the addition of noise value to the model with normalization. The model output of an image is then compared with the JPEG image which uses DFT compression technique using PSNR and SSIM quality metrics.

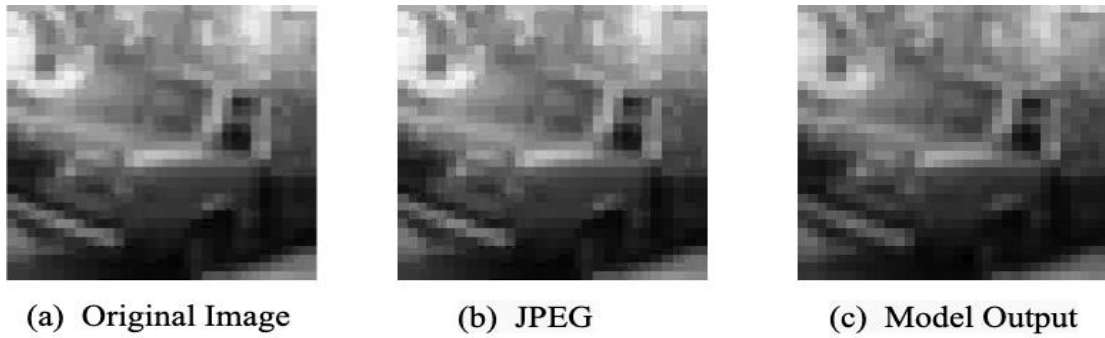


Figure 5.13: Original, JPEG and Model Output image

Table 5.2: Comparison of proposed model output with JPEG






	PSNR	SSIM
JPEG	25.27	0.82
Purposed Model	34.47	0.94

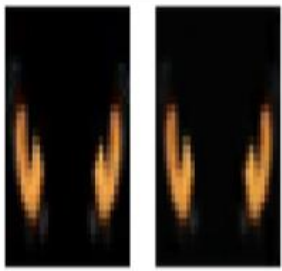
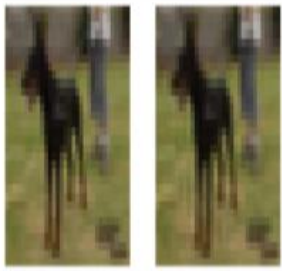



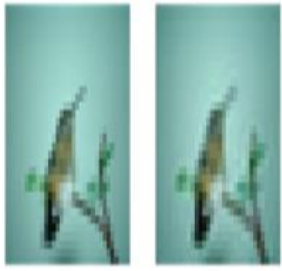
The result from Table 5.2 shows that the purposed model output has better reconstruction than the existing JPEG compression using DFT.







5.4 QUALITY MEASUREMENTS

Various state of the art quality metrics such as MSE, PSNR and SSMI is used to measure the quality of the reconstructed result. The results of the original and reconstructed images of the random sample are shown in Table 5.3. Table 5.4 depicts the result of the noisy and reconstructed image.

Table 5.3: Quality measurement using quality assessment techniques for original vs reconstructed image

Original vs Reconstructed Image	Quality Assessment Techniques		
	MSE	PSNR	SSMI
	0.0002	37.060	0.994
	0.0003	36.015	0.992
	0.0003	34.728	0.991
	0.0002	38.157	0.993
	0.0003	35.626	0.995

Original vs Reconstructed Image	Quality Assessment Techniques		
	MSE	PSNR	SSMI
	0.0004	34.139	0.833
	0.0002	36.940	0.989
	0.0003	35.375	0.987
	0.0004	34.264	0.992
	0.0004	34.319	0.986
	0.0004	34.337	0.984

Original vs Reconstructed Image	Quality Assessment Techniques		
	MSE	PSNR	SSMI
	0.0005	32.897	0.980
	0.0006	32.413	0.989
	0.0003	34.588	0.990
	0.0005	32.809	0.990
	0.0004	34.490	0.992
	0.0003	34.598	0.990

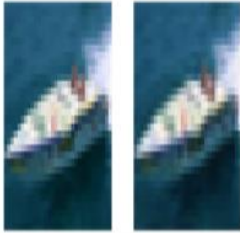


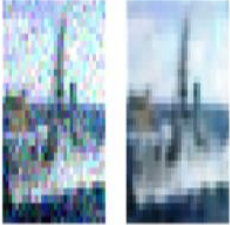

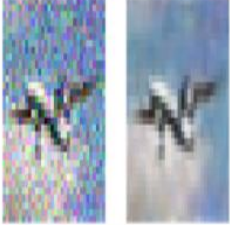

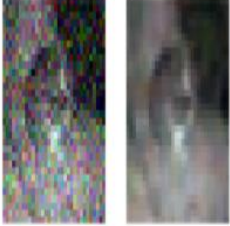



























Original vs Reconstructed Image	Quality Assessment Techniques		
	MSE	PSNR	SSMI
	0.0005	33.199	0.980
	0.0002	36.628	0.990
	0.0007	31.602	0.984

Table 5.4: Quality measurement using quality assessment techniques for noisy vs reconstructed image

Noisy vs Reconstructed Image	Quality Assessment Techniques		
	MSE	PSNR	SSMI
	0.0085	23.498	0.784
	0.0084	23.448	0.708
	0.0085	22.582	0.655
	0.0094	25.064	0.759
	0.0085	22.308	0.628

Noisy vs Reconstructed Image		Quality Assessment Techniques		
		MSE	PSNR	SSMI
		0.0087	23.782	0.654
		0.0093	23.250	0.852
		0.0089	21.923	0.768
		0.0085	21.180	0.636
		0.0091	22.737	0.713
		0.0092	22.589	0.758

Noisy vs Reconstructed Image		Quality Assessment Techniques		
		MSE	PSNR	SSMI
		0.0087	21.099	0.655
		0.0086	21.575	0.678
		0.0093	23.593	0.767
		0.0089	23.116	0.817
		0.0085	23.378	0.755
		0.0086	23.286	0.710

Noisy vs Reconstructed Image	Quality Assessment Techniques		
	MSE	PSNR	SSMI
	0.0087	22.730	0.685
	0.0084	22.239	0.756
	0.0086	23.985	0.776

5.5 SUMMARY

The results obtained using the proposed model with different settings are explored. MSE is used as a loss function to calculate the predicted errors across the training samples. The accuracy of the results is also measured. Different regularization techniques are applied for optimization. The model is also evaluated with noisy data along with other quality metrics: MSE, PSNR, and SSMI and the reconstructed output of model is compared with JPEG image using PSNR and SSIM quality metrics.

CHAPTER 6

DISCUSSION AND CONCLUSIONS

6.1 INTRODUCTION

This chapter discusses the research findings and derives a conclusion based on the results of evaluations. The contribution of this study is discussed in the contribution of the study section and future recommendations are made.

6.2 DISCUSSION AND CONCLUSION

In this dissertation, a CNN model is presented to reduce the construction loss. The experimental evaluation of the proposed model shows that it has better quality results with reduced construction loss.

The proposed model uses an autoencoder network where the encoder layer compresses the input image into encoded representation and the decoder layer decoded the encoded representation using knowledge representation. The magnitude of encoded representation is smaller than input data. The evaluation results show that with regularization, the reconstruction loss dropped significantly, and performance also increased.

The reconstructed results are compared with different state of art quality metrics. The evaluation result shows that CNN techniques achieve good compression with better reconstruction compared to JPEG using DFT compression. In addition, CNN techniques can reconstruct an image from a noisy image, where the noise is tied to the image's high-frequency content can be reconstructed.

6.3 CONTRIBUTION OF THE STUDY

This research makes three contributions. First, the study of the relationship between the border effect problem and DFT compression in relation to JPEG, and various state of art solutions to it. Second, propose a compression technique using CNN, to capture the most

essential elements of the input image to learn a lower-dimensional representation and reconstruct the image based on compression knowledge representation. Third, this proposed technique reconstructs images from a noisy input with moderate quality results.

6.4 FUTURE RECOMMENDATIONS

This research has highlighted several areas where further research can be beneficial. These highlighted areas are mostly mentioned in the literature review. Besides that, several additional areas such as further investigation of regularization techniques along with variations of the autoencoder can be carried out for better performance.

Furthermore, different data sets with different settings can be used to explore to minimize the reconstruction loss. Improving the quality of the noisy image can also be a great topic to investigate.

6.5 SUMMARY

Various aspects of the proposed technique are discussed, and conclusions are made based on the results of the evaluation performed. The research contributions are discussed in the contribution of the study section and based on the findings of the research recommendations are made.

REFERENCES

- Abraham, B., Camps, O. I., & Sznaier, M. (2005). Dynamic texture with fourier descriptors. *Proceedings of the 4th international workshop on texture analysis and synthesis, 1*, 53–58.
- Alzubaidi, L., Zhang, J., Humaidi, A. J., Al-Dujaili, A., Duan, Y., Al-Shamma, O., Santamaría, J., Fadhel, M. A., Al-Amidie, M., & Farhan, L. (2021). Review of deep learning: Concepts, CNN architectures, challenges, applications, future directions. *Journal of Big Data*, 8(1), 53. <https://doi.org/10.1186/s40537-021-00444-8>
- Alexandre, D., Chang, C.-P., Peng, W.-H., & Hang, H.-M. (2019). An autoencoder-based learned image compressor: Description of challenge proposal by nct. <https://doi.org/10.48550/ARXIV.1902.07385>
- Annoni, R., & Forster, C. H. Q. (2012). Experimental comparison of dwt and dft for trajectory representation. *International Conference on Intelligent Data Engineering and Automated Learning*, 670–677.
- Audhkhasi, K., Osoba, O., & Kosko, B. (2016). Noise-enhanced convolutional neural networks. *Neural Networks*, 78, 15–23. <https://doi.org/10.1016/j.neunet.2015.09.014>
- Baig, M. H., Koltun, V., & Torresani, L. (2017). Learning to inpaint for image compression [arXiv:1709.08855]. *arXiv:1709.08855[cs]*. Retrieved March 16, 2022, from <http://arxiv.org/abs/1709.08855>
- Ballé, J., Laparra, V., & Simoncelli, E. P. (2017). End-to-end optimized image compression [arXiv: 1611.01704]. *arXiv:1611.01704 [cs, math]*. Retrieved March 16, 2022, from <http://arxiv.org/abs/1611.01704>
- Bank, D., Koenigstein, N., & Giryes, R. (2020). Autoencoders. <https://doi.org/10.48550/ARXIV.2003.05991>
- Brownlee, J. (2021). How to Choose an Activation Function for Deep Learning. Retrieved July 22, 2022, from <https://machinelearningmastery.com/choose-an-activationfunction-for-deep-learning/>
- Cavigelli, L., Hager, P., & Benini, L. (2017). Cas-cnn: A deep convolutional neural network for image compression artifact suppression. *2017 International Joint Conference on Neural Networks (IJCNN)*. <https://doi.org/10.1109/ijcnn.2017.7965927>

- Cavigelli, L., Magno, M., & Benini, L. (2015). Accelerating real-time embedded scene labeling with convolutional networks. *Proceedings of the 52nd Annual Design Automation Conference*, 1–6. <https://doi.org/10.1145/2744769.2744788>
- Chollet, F. (2017). Xception: Deep learning with depthwise separable convolutions. *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 1800–1807. <https://doi.org/10.1109/CVPR.2017.195>
- De Carvalho, J. R., Duque, C. A., Lima, M. A., Coury, D. V., & Ribeiro, P. F. (2014). A novel dft-based method for spectral analysis under time-varying frequency conditions. *Electric Power Systems Research*, *108*, 74–81.
- Deshmukh, K. R. (2019). *Image compression using neural networks* (Master of Science). San Jose State University. San Jose, CA, USA. <https://doi.org/10.31979/etd.h8mt65ct>
- DeVries, T., & Taylor, G. W. (2017). Improved regularization of convolutional neural networks with cutout. <https://doi.org/10.48550/ARXIV.1708.04552>
- Dong, C., Deng, Y., Loy, C. C., & Tang, X. (2015). Compression artifacts reduction by a deep convolutional network [arXiv: 1504.06993]. *arXiv:1504.06993 [cs]*. Retrieved March 16, 2022, from <http://arxiv.org/abs/1504.06993>
- Dong, C., Loy, C. C., He, K., & Tang, X. (2014). Learning a deep convolutional network for image super-resolution. In D. Fleet, T. Pajdla, B. Schiele, & T. Tuytelaars (Eds.), *Computer vision – eccv 2014* (pp. 184–199). Springer International Publishing.
- Dong, Y., Jiao, W., Long, T., Liu, L., & He, G. (2019). Eliminating the effect of image border with image periodic decomposition for phase correlation based remote sensing image registration. *Sensors*, *19*(10), 2329. <https://doi.org/10.3390/s19102329>
- Dosovitskiy, A., Fischer, P., Ilg, E., Hausser, P., Hazirbas, C., Golkov, V., Smagt, P. v. d., Cremers, D., & Brox, T. (2015). FlowNet: Learning optical flow with convolutional networks. *2015 IEEE International Conference on Computer Vision (ICCV)*, 2758–2766. <https://doi.org/10.1109/ICCV.2015.316>
- Dubey, S. R., Singh, S. K., & Chaudhuri, B. B. (2021). Activation functions in deep learning: A comprehensive survey and benchmark. <https://doi.org/10.48550/ARXIV.2109.14545>
- Dubois, Y., Bloem-Reddy, B., Ullrich, K., & Maddison, C. J. (2022). Lossy compression for lossless prediction.

- Ede, J. M., & Beanland, R. (2020). Adaptive learning rate clipping stabilizes learning. *Machine Learning: Science and Technology*, 1(1), 015011. <https://doi.org/10.1088/2632-2153/ab81e2>
- Ge, P., Lan, C., & Wang, H. (2014). An improvement of image registration based on phase correlation. *Optik*, 125(22), 6709–6712. <https://doi.org/10.1016/j.ijleo.2014.07.086>
- Giannakis, G. B., Bach, F., Cendrillon, R., Mahoney, M., & Neville, J. (2014). Signal processing for big data [from the guest editors]. *IEEE Signal Processing Magazine*, 31(5), 15–16.
- Goel, N., & Gabarda, S. (2016). Lossy and lossless image compression using legendre polynomials. *2016 Conference on Advances in Signal Processing (CASP)*, 315–320. <https://doi.org/10.1109/CASP.2016.7746187>
- Gonzalez, R. C., Woods, R. E., & Masters, B. R. (2009). Digital image processing.
- Gupta, B. C. (2021). Sampling methods. *Statistical quality control: Using minitab, r, jmp and python* (pp. 89–121). <https://doi.org/10.1002/9781119671718.ch4>
- Gustineli, M. (2022). A survey on recently proposed activation functions for deep learning. <https://doi.org/10.48550/ARXIV.2204.02921>
- Hadsell, R., Chopra, S., & LeCun, Y. (2006). Dimensionality Reduction by Learning an Invariant Mapping. *2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition - Volume 2 (CVPR'06)*, 2, 1735–1742. <https://doi.org/10.1109/CVPR.2006.100>
- He, K., Zhang, X., Ren, S., & Sun, J. (2015). Deep residual learning for image recognition. <https://doi.org/10.48550/ARXIV.1512.03385>
- He, X., Xue, F., Ren, X., & You, Y. (2021). Large-scale deep learning optimizations: A comprehensive survey. <https://doi.org/10.48550/ARXIV.2111.00856>
- Ho-Phuoc, T. (2018). Cifar10 to compare visual recognition performance between deep neural networks and humans. <https://doi.org/10.48550/ARXIV.1811.07270>
- Hosseini, M. S., Zhang, J. S., Liu, Z., Fu, A., Su, J., Tuli, M., Hosseini, S., Kadakia, A., Wang, H., & Plataniotis, K. N. (2021). Conet: Channel optimization for convolutional neural networks. <https://doi.org/10.48550/ARXIV.2108.06822>
- Hovden, R., Jiang, Y., Xin, H. L., & Kourkoutis, L. F. (2015). Periodic artifact reduction in fourier transforms of full field atomic resolution images. *Microscopy and Microanalysis*, 21(2), 436–441. <https://doi.org/10.1017/S1431927614014639>
- Huang, G., Liu, Z., van der Maaten, L., & Weinberger, K. Q. (2016). Densely connected

- convolutional networks. <https://doi.org/10.48550/ARXIV.1608.06993>
- Hui, L., & Belkin, M. (2020). Evaluation of neural architectures trained with square loss vs cross-entropy in classification tasks. <https://doi.org/10.48550/ARXIV.2006.07322>
- Hussain, A. A., AL-Khafaji, G. K., & Siddeq, M. M. (2020). Developed JPEG algorithm applied in image compression. *IOP Conference Series: Materials Science and Engineering*, 928(3), 032006. <https://doi.org/10.1088/1757-899x/928/3/032006>
- Johnson, R., & Zhang, T. (2013). Accelerating stochastic gradient descent using predictive variance reduction. In C. Burges, L. Bottou, M. Welling, Z. Ghahramani, & K. Weinberger (Eds.), *Advances in neural information processing systems*. Curran Associates, Inc. <https://proceedings.neurips.cc/paper/2013/file/ac1dd209cbcc5e5d1c6e28598e8cbbe8Paper.pdf>
- Khan, A., Sohail, A., Zahoor, U., & Qureshi, A. S. (2020). A survey of the recent architectures of deep convolutional neural networks. *Artificial Intelligence Review*, 53(8), 5455–5516. <https://doi.org/10.1007/s10462-020-09825-6>
- Kingma, D. P., & Ba, J. (2014). Adam: A method for stochastic optimization. <https://doi.org/10.48550/ARXIV.1412.6980>
- Kiourt, C., Pavlidis, G., & Markantonatou, S. (2020). Deep learning approaches in food recognition. <https://doi.org/10.48550/ARXIV.2004.03357>
- Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012a). Imagenet classification with deep convolutional neural networks. In F. Pereira, C. Burges, L. Bottou, & K. Weinberger (Eds.), *Advances in neural information processing systems*. Curran Associates, Inc. <https://proceedings.neurips.cc/paper/2012/file/c399862d3b9d6b76c8436e924a68c45bPaper.pdf>
- Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012b). Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*, 25.
- Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2017). ImageNet classification with deep convolutional neural networks. *Communications of the ACM*, 60(6), 84–90. <https://doi.org/10.1145/3065386>

- Kudeshia, S., & Potnis, A. A. (2017). Novel design of fft using high radix butterfly of complexvalueddata.2017InternationalConferenceonComputerCommunication and Informatics (ICCCI), 1–5. <https://doi.org/10.1109/ICCCI.2017.8117795>
- Kundu, S., Mostafa, H., Sridhar, S. N., & Sundaresan, S. (2020). Attention-based image upsampling. <https://doi.org/10.48550/ARXIV.2012.09904>
- Kunwar, S. (2017). Image Compression Algorithm and JPEG Standard. *International Journal of Scientific and Research Publications*, 7(12), 150–157. <http://www.ijsrp.org/research-paper-1217/ijsrp-p7224.pdf>
- Kunwar, S. (2018). Jpeg image compression using cnn, 1. <https://doi.org/10.13140/RG.2.2.25600.53762>
- Lai, E. (2004). *Practical digital signal processing for engineers and technicians* [OCLC: ocm51527224]. Newnes.
- LeCun, Y., Jackel, L. D., Bottou, L., Cortes, C., Denker, J. S., Drucker, H., Guyon, I., Muller, U. A., Sackinger, E., Simard, P., et al. (1995). Learning algorithms for classification: A comparison on handwritten digit recognition. *Neural networks: the statistical mechanics perspective*, 261(276), 2.
- Legrand, A., Nieperon, B., Cournier, A., & Trannois, H. (2018). Study of Autoencoder Neural Networks for Anomaly Detection in Connected Buildings. *2018 IEEE Global Conference on Internet of Things (GCIoT)*, 1–5. <https://doi.org/10.1109/GCIoT.2018.8620158>
- Leprince, S., Barbot, S., Ayoub, F., & Avouac, J.-P. (2007). Automatic and precise orthorectification, coregistration, and subpixel correlation of satellite images, application to ground deformation measurements. *IEEE Transactions on Geoscience and Remote Sensing*, 45(6), 1529–1558.
- Li, J., Wang, Y., Xie, H., & Ma, K.-K. (2020). Learning a single model with a wide range of quality factors for jpeg image artifacts removal. *IEEE Transactions on Image Processing*, 29, 8842–8854. <https://doi.org/10.1109/tip.2020.3020389>
- Li, P., & Lo, K.-T. (2019). Joint image encryption and compression schemes based on 16 * 16 dct. *Journal of Visual Communication and Image Representation*, 58, 12–24.
- Li, Z., Yang, W., Peng, S., & Liu, F. (2020). A survey of convolutional neural networks: Analysis, applications, and prospects. <https://doi.org/10.48550/ARXIV.2004.02806>

- Liu, W., Wen, Y., Yu, Z., & Yang, M. (2016). Large-margin softmax loss for convolutional neural networks. <https://doi.org/10.48550/ARXIV.1612.02295>
- Lo, W. W., Yang, X., & Wang, Y. (2019). An xception convolutional neural network for malware classification with transfer learning. *2019 10th IFIP International Conference on New Technologies, Mobility and Security (NTMS)*, 1–5. <https://doi.org/10.1109/NTMS.2019.8763852>
- Long, J., Shelhamer, E., & Darrell, T. (2015). Fully convolutional networks for semantic segmentation. *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 3431–3440. <https://doi.org/10.1109/CVPR.2015.7298965>
- Lu, L. (2020). Dying ReLU and initialization: Theory and numerical examples. *Communications in Computational Physics*, 28(5), 1671–1706. <https://doi.org/10.4208/cicp.oa-2020-0165>
- Maleki, D., Nadalian, S., Derakhshani, M. M., & Sadeghi, M. A. (2018). Blockcnn: A deep network for artifact removal and image compression [arXiv: 1805.11091]. *arXiv:1805.11091 [cs]*. Retrieved March 17, 2022, from <http://arxiv.org/abs/1805.11091>
- Malvar, H. (1999). A modulated complex lapped transform and its applications to audio processing. *1999 IEEE International Conference on Acoustics, Speech, and Signal Processing. Proceedings. ICASSP99 (Cat. No. 99CH36258)*, 3, 1421–1424.
- Mao, X.-J., Shen, C., & Yang, Y.-B. (2016). Image restoration using very deep convolutional encoder-decoder networks with symmetric skip connections [arXiv: 1603.09056]. *arXiv:1603.09056 [cs]*. Retrieved March 16, 2022, from <http://arxiv.org/abs/1603.09056>
- Mathur, M., & Mathur, G. (2012). Image compression using DFT through Fast Fourier Transform Technique. *International Journal of Emerging Trends & Technology in Computer Science*, 1(2), 129–133. <https://www.ijettcs.org/Volume1Issue2/IJETTCS-2012-08-23-086.pdf>
- Naftel, A., & Khalid, S. (2006). Classifying spatiotemporal object trajectories using unsupervised learning in the coefficient feature space. *Multimedia Systems*, 12(3), 227–238.
- Nanni, L., Maguolo, G., & Lumini, A. (2021). Exploiting adam-like optimization algorithms to improve the performance of convolutional neural networks. <https://doi.org/10.48550/ARXIV.2103.14689>

- Niu, Y., Tondi, B., Zhao, Y., & Barni, M. (2020). Primary quantization matrix estimation of double compressed jpeg images via cnn. <https://doi.org/10.1109/LSP.2019.2962997>
- O'Shea, K., & Nash, R. (2015). An Introduction to Convolutional Neural Networks [arXiv: 1511.08458]. *arXiv:1511.08458 [cs]*. Retrieved December 3, 2021, from <http://arxiv.org/abs/1511.08458>
- Pielawski, N., & Wählby, C. (2020). Introducing Hann windows for reducing edge-effects in patch-based image segmentation (J. Zhang, Ed.). *PLOS ONE*, *15*(3), e0229839. <https://doi.org/10.1371/journal.pone.0229839>
- Podder, P., Zaman Khan, T., Haque Khan, M., & Muktadir Rahman, M. (2014). Comparative performance analysis of hamming, hanning and blackman window. *International Journal of Computer Applications*, *96*(18), 1–7. <https://doi.org/10.5120/16891-6927>
- Poor, H. V. (1988). *An introduction to signal detection and estimation* (J. B. Thomas, Ed.). Springer New York. <https://doi.org/10.1007/978-1-4757-3863-6>
- Pouyanfar, S., Sadiq, S., Yan, Y., Tian, H., Tao, Y., Reyes, M. P., Shyu, M.-L., Chen, S.-C., & Iyengar, S. S. (2019). A survey on deep learning: Algorithms, techniques, and applications. *ACM Computing Surveys*, *51*(5), 1–36. <https://doi.org/10.1145/3234150>
- Puchala, D., & Stokfiszewski, K. (2021). Convolutional neural network for image compression with application to jpeg standard. *2021 Data Compression Conference (DCC)*, 361–361. <https://doi.org/10.1109/DCC50243.2021.00048>
- Qian, N. (1999). On the momentum term in gradient descent learning algorithms. *neural Networks*, *12*(1), 145–151. [https://doi.org/10.1016/S08936080\(98\)00116-6](https://doi.org/10.1016/S08936080(98)00116-6)
- Rahman, M. A., Islam, S. M. S., Shin, J., & Islam, M. R. (2018). Histogram alternation based digital image compression using base-2 coding. *2018 Digital Image Computing: Techniques and Applications*, 1–8. <https://doi.org/10.1109/DICTA.2018.8615830>
- Rao, K. R., & Ochoa-Dominguez, H. (2019). *Discrete cosine transform* (Second edition). Taylor & Francis Group, CRC Press.
- Rasheed, M. H., Salih, O. M., Siddeq, M. M., & Rodrigues, M. A. (2020). Image compression based on 2D Discrete Fourier Transform and matrix minimization algorithm. *Array*, *6*, 100024. <https://doi.org/10.1016/j.array.2020.100024>

- Reddi, S., Kale, S., & Kumar, S. (2018). On the convergence of adam and beyond. *International Conference on Learning Representations*.
- Ren, S., He, K., Girshick, R., & Sun, J. (2016). Faster r-cnn: Towards real-time object detection with region proposal networks [arXiv: 1506.01497]. *arXiv:1506.01497 [cs]*. Retrieved September 25, 2021, from <http://arxiv.org/abs/1506.01497>
- Rice, L., Wong, E., & Kolter, J. Z. (2020). Overfitting in adversarially robust deep learning. <https://doi.org/10.48550/ARXIV.2002.11569>
- Rozenwald, M. B., Galitsyna, A. A., Sapunov, G. V., Khrameeva, E. E., & Gelfand, M. S. (2020). A machine learning framework for the prediction of chromatin folding in drosophila using epigenetic features. *PeerJ Computer Science*, 6, e307.
- Santurkar, S., Budden, D., & Shavit, N. (2017). Generative Compression [arXiv: 1703.01467]. *arXiv:1703.01467 [cs]*. Retrieved January 4, 2022, from <http://arxiv.org/abs/1703.01467>
- Santurkar, S., Tsipras, D., Ilyas, A., & Madry, A. (2018). How does batch normalization help optimization? <https://doi.org/10.48550/ARXIV.1805.11604>
- Sara, U., Akter, M., & Uddin, M. S. (2019). Image quality assessment through fsim, ssim, mse and psnr—a comparative study. *Journal of Computer and Communications*, 07(03), 8–18. <https://doi.org/10.4236/jcc.2019.73002>
- Schroff, F., Kalenichenko, D., & Philbin, J. (2015). FaceNet: A unified embedding for face recognition and clustering. *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 815–823. <https://doi.org/10.1109/CVPR.2015.7298682>
- Sevgi, L. (2014). Fourier transform and fourier series. *Electromagnetic modeling and simulation* (pp. 71–94). <https://doi.org/10.1002/9781118716410.ch4>
- Seyyarer, E., Uçkan, T., Hark, C., Ayata, F., İnan, M., & Karci, A. (2019). Applications and comparisons of optimization algorithms used in convolutional neural networks. *2019 International Artificial Intelligence and Data Processing Symposium (IDAP)*, 1–6. <https://doi.org/10.1109/IDAP.2019.8875929>
- Shorten, C., & Khoshgoftaar, T. M. (2019). A survey on Image Data Augmentation for Deep Learning. *Journal of Big Data*, 6(1), 60. <https://doi.org/10.1186/s40537019-0197-0>
- Shrivastav, A. (2021). Different types of CNN models. Retrieved July 21, 2022, from <https://iq.opengenus.org/different-types-of-cnn-models/>

- Shui, P.-L. (2009). Image denoising using 2-d separable oversampled dft modulated filter banks. *IET Image processing*, 3(3), 163–173.
- Siddeq, M., & Rodrigues, M. (2014a). A novel image compression algorithm for high resolution 3d reconstruction. *3D Research*, 5(2), 7.
- Siddeq, M., & Rodrigues, M. (2014b). A new 2d image compression technique for 3d surface reconstruction.
- Siddeq, M. M., & Al-Khafaji, G. (2013). Applied minimized matrix size algorithm on the transformed images by dct and dwt used for image compression. *International Journal of Computer Applications*, 70(15).
- Simonyan, K., & Zisserman, A. (2014). Very deep convolutional networks for large-scale image recognition. <https://doi.org/10.48550/ARXIV.1409.1556>
- SMPTE. (2020). The future of the JPEG standard. Retrieved March 30, 2022, from <https://www.smpte.org/blog/the-future-of-the-jpeg-standard>
- Steck, H., & Garcia, D. G. (2021). On the regularization of autoencoders. <https://doi.org/10.48550/ARXIV.2110.11402>
- Sujitha, B., Parvathy, V. S., Lydia, E. L., Rani, P., Polkowski, Z., & Shankar, K. (2021). Optimal deep learning based image compression technique for data transmission on industrial Internet of things applications. *Transactions on Emerging Telecommunications Technologies*, 32(7). <https://doi.org/10.1002/ett.3976>
- Sun, S., Cao, Z., Zhu, H., & Zhao, J. (2019). A survey of optimization methods from a machine learning perspective. <https://doi.org/10.48550/ARXIV.1906.06821>
- Svoboda, P., Hradis, M., Barina, D., & Zemcik, P. (2016). Compression artifacts removal using convolutional neural networks [arXiv: 1605.00366]. *arXiv:1605.00366 [cs]*. Retrieved March 16, 2022, from <http://arxiv.org/abs/1605.00366>
- Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V., & Rabinovich, A. (2014). Going deeper with convolutions. <https://doi.org/10.48550/ARXIV.1409.4842>
- Tian, J., & Ma, K.-K. (2011). A survey on super-resolution imaging. *Signal, Image and Video Processing*, 5(3), 329–342.
- Tzimiropoulos, G., Argyriou, V., Zafeiriou, S., & Stathaki, T. (2010). Robust fft-based scale-invariant image registration with image gradients. *IEEE transactions on pattern analysis and machine intelligence*, 32(10), 1899–1906.

- van der Walt, S., Schönberger, J. L., Nunez-Iglesias, J., Boulogne, F., Warner, J. D., Yager, N., Gouillart, E., Yu, T., & the scikit-image contributors. (2014). Scikit-image: Image processing in Python. *PeerJ*, 2, e453. <https://doi.org/10.7717/peerj.453>
- W3Techs. (2022). Usage statistics of image file formats for websites. Retrieved March 2, 2022, from https://w3techs.com/technologies/overview/image_format
- Wallace, G. K. (1991). The JPEG still picture compression standard. *Communications of the ACM*, 34(4), 30–44. <https://doi.org/10.1145/103085.103089>
- Wan, S., Wu, T.-Y., Hsu, H.-W., Wong, W. H., & Lee, C.-Y. (2020). Feature consistency training with jpeg compressed images. *IEEE Transactions on Circuits and Systems for Video Technology*, 30(12), 4769–4780. <https://doi.org/10.1109/TCSVT.2019.2959815>
- Wang, Z., Bovik, A., & Lu, L. (2021). *Why is image quality assessment so difficult?* https://live.ece.utexas.edu/publications/2002/zw_icassp2002_whyqa.pdf
- Wen, Y., Zhang, K., Li, Z., & Qiao, Y. (2016). A discriminative feature learning approach for deep face recognition. In B. Leibe, J. Matas, N. Sebe, & M. Welling (Eds.), *Computer vision – eccv 2016* (pp. 499–515). Springer International Publishing.
- Xie, S., Girshick, R., Dollár, P., Tu, Z., & He, K. (2016). Aggregated residual transformations for deep neural networks. <https://doi.org/10.48550/ARXIV.1611.05431>
- Yadav, S., & Shukla, S. (2016). Analysis of k-fold cross-validation over hold-out validation on colossal datasets for quality classification. *2016 IEEE 6th International Conference on Advanced Computing (IACC)*, 78–83. <https://doi.org/10.1109/IACC.2016.25>
- Yuan, S., & Hu, J. (2019). Research on image compression technology based on huffman coding. *Journal of Visual Communication and Image Representation*, 59, 33–38.
- Zeiler, M. D., & Fergus, R. (2013). Visualizing and understanding convolutional networks. <https://doi.org/10.48550/ARXIV.1311.2901>
- Zeiler, M. D., & Fergus, R. (2014). Visualizing and Understanding Convolutional Networks. In D. Fleet, T. Pajdla, B. Schiele, & T. Tuytelaars (Eds.), *Computer Vision – ECCV 2014* (pp. 818–833). Springer International Publishing. https://doi.org/10.1007/978-3-319-10590-1_53
- Zhang, K., Zuo, W., Chen, Y., Meng, D., & Zhang, L. (2017). Beyond a gaussian denoiser: Residual learning of deep cnn for image denoising [arXiv: 1608.03981].

IEEE Transactions on Image Processing, 26(7), 3142–3155.
<https://doi.org/10.1109/TIP.2017.2662206>