



January 2020

## Removal Of Blocking Artifacts From JPEG-Compressed Images Using Neural Network

Md Nurul Amin

Follow this and additional works at: <https://commons.und.edu/theses>

---

### Recommended Citation

Amin, Md Nurul, "Removal Of Blocking Artifacts From JPEG-Compressed Images Using Neural Network" (2020). *Theses and Dissertations*. 3089.  
<https://commons.und.edu/theses/3089>

This Thesis is brought to you for free and open access by the Theses, Dissertations, and Senior Projects at UND Scholarly Commons. It has been accepted for inclusion in Theses and Dissertations by an authorized administrator of UND Scholarly Commons. For more information, please contact [und.commons@library.und.edu](mailto:und.commons@library.und.edu).

REMOVAL OF BLOCKING ARTIFACTS FROM JPEG-COMPRESSED IMAGES  
USING NEURAL NETWORK

by

Md Nurul Amin

Bachelor of Science, Khulna University of Engineering & Technology, 2012

A Thesis

Submitted to the Graduate Faculty

of the

University of North Dakota

In partial fulfillment of the requirements

for the degree of

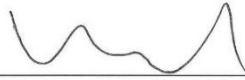
Master of Science

Grand Forks, North Dakota

May

2020

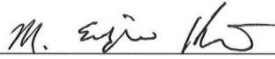
This thesis is submitted by Md Nurul Amin for partial fulfillment of the requirements for the degree of Master of Science from the University of North Dakota. The following committee members have read the document and approved.



Dr. Ronald Marsh, Chairperson



Dr. Wen-Chen Hu



Dr. Eunjin Kim

This thesis is being submitted by the appointed advisory committee as having met all the requirements of the School of Graduate Studies at the University of North Dakota and is hereby approved.



Dr. Chris Nelson

Associate Dean of the School of Graduate Studies

04/01/20 \_\_\_\_\_ Date

## PERMISSION

Title            Removal of blocking artifacts from Jpeg-Compressed images using neural network.

Department    Computer Science

Degree            Master of Science

In presenting this thesis in partial fulfillment of the requirements for a graduate degree from the University of North Dakota, I agree that the library of this University shall make it freely available for inspection. I further agree that permission for extensive copying for scholarly purposes may be granted by the professor who supervised my thesis work or, in his absence, by the chairperson of the department or the dean of the graduate school. It is understood that any copying or publication or other use of this thesis or part thereof for financial gain shall not be allowed without my written permission. It is also understood that due recognition shall be given to me and the University of North Dakota in any scholarly use which may be made of any material to my thesis.

Md Nurul Amin

February 21, 2019

## TABLE OF CONTENTS

LIST OF FIGURES .....	v
LIST OF TABLES .....	vi
LIST OF ABBREVIATIONS .....	vii
ACKNOWLEDGEMENTS.....	viii
ABSTRACT.....	ix
CHAPTER	
I. INTRODUCTION.....	1
II. RELATED RESEARCH ON REMOVING ARTIFACTS FROM COMPRESSED IMAGE .....	6
III. IMPLEMENTING NEURAL NETWORK IN JPEG IMAGE POST-PROCESSING TO IMPROVE QUALITY .....	25
IV. EXPERIMENTAL RESULTS .....	38
V. CONCLUSION .....	43
REFERENCES .....	50

## LIST OF FIGURES

Figure	Page
1. Loss of pixel data in lossy compression .....	3
2. Example of blocking artifacts.....	7
3. Gibb’s oscillation.....	8
4. (a) Staircase Noise, (b) Corner outlier.....	10
5. Adjacent blocks of pixels.....	16
6. Lena, before and after .....	20
7. Flowchart of the original Alpha blend algorithm .....	22
8. The Prewitt filter .....	22
9. An Artificial Neural Network .....	27
10. The 8x8 centered pixel block surrounded by a 12x12 pixel block .....	30
11. The Neural Network training configuration.....	30
12. Two overlapping 12X12 pixel blocks from input image .....	31
13. The trained Neural Network working process .....	32
14. SNNS “Manager” Panel .....	33
15. SNNS “Bignet” Panel .....	34
16. SNNS “Control” Panel .....	35
17. SNNS “Graph” Panel .....	36
18. (a) Lena with a high compression input JPEG image (0.150 bpp)	
(b) Neural network output image .....	41
19. (a) Lena with a low compression input JPEG image (0.430 bpp)	
(b) Neural network output image .....	42
20. Graph with overfitting in validation curve .....	47

## LIST OF TABLES

Figure	Page
1. Results of SNNS neural network .....	39

## LIST OF ABBREVIATIONS

Index	Page
1. GPS (Global Positioning System) .....	1
2. JPEG (Joint Photographic Experts Group) .....	1
3. ISO (International Organization for Standardization) .....	2
4. ITU-T (International Telecommunication Group) .....	2
5. BDCT (Block-based Discrete Cosine Transform) .....	3
6. DCT (Discrete Cosine Transformation) .....	3
7. MSSIM (Mean Structural Similarity Index Measure) .....	4
8. HVS (Human Visual System) .....	12
9. CVG (Classified Vector Quantization) .....	17
10. PSNR (Peak Signal to Noise Ratio) .....	23
11. MSE (Mean Square Error) .....	26
12. SNNS (Stuttgart Neural Network Simulator) .....	32
13. FSIM (Feature Similarity Index) .....	49
14. SSIM (Structural Similarity Index) .....	49



## ACKNOWLEDGEMENTS

I want to appreciate my advisory committee members for their guidance and support during my Master's program at University of North Dakota. I express my sincere gratitude to my advisor and chairperson Prof. Dr. Ronald Marsh for the continuous support of Master's study, for his patience, motivation and extensive knowledge. I am grateful to my committee members Dr. Wen-Chen Hu and Dr. Eunjin Kim. Without their generous guidance and time, the thesis would not be completed. My sincere thanks to Prof. Dr. Ronald Marsh to give me the opportunity to work in his research team and introducing this excellent research area. Through his guidance I found advanced research areas and solving critical problems in image processing. I appreciate his guidance and encouragement in my academic years at UND and working on this thesis. I am also thankful to Dr. Wen-Chen Hu and Dr. Eunjin Kim in agreeing to be in my thesis committee as well as giving me advise and support in my academic life at UND.

Finally, I must express my very profound gratitude to my dad Md Mofizul Islam, my mom Razia Begum and my siblings for providing me with unflinching support and continuous encouragement throughout my years of study and through the process of researching and writing this thesis. This accomplishment would not have been possible without the persons I mentioned here.

## ABSTRACT

The goal of this research was to develop a neural network that will produce considerable improvement in the quality of JPEG compressed images, irrespective of compression level present in the images. In order to develop a computationally efficient algorithm for reducing blocky and Gibbs oscillation artifacts from JPEG compressed images, we integrated artificial intelligence to remove blocky and Gibbs oscillation artifacts. In this approach, alpha blend filter [7] was used to post process JPEG compressed images to reduce noise and artifacts without losing image details. Here alpha blending was controlled by a limit factor that considers the amount of compression present, and any local information derived from Prewitt filter application in the input JPEG image. The outcome of modified alpha blend was improved by a trained neural network and compared with various other published works [7][9][11][14][20][23][30][32][33][35][37] where authors used post compression filtering methods.

## **CHAPTER I**

### **INTRODUCTION**

Image compression refers to the minimization of the size of a graphics file, by reducing the number of bytes in the image while keeping the quality and visual integrity of image in a standard format. Image compression is required mostly for data transmission and storage purposes where size of data impacts cost.

Faster transmission of data is getting more important. High-speed transfer of data in graphical format, via remote video communication, data from GPS (Global Positioning System) and geo-stationary satellites, or instant image sharing hosts over the internet all need image compression. A smaller file will require less bandwidth to transfer the data over a network. Another benefit of reducing data size is that it helps to store image within minimal space. As image capturing technology is very easy to access, people are using vast number of images in regular contexts. As such, compression of images before storing them on the disk is still a matter of research.

There can be lossy and lossless image compression. According to the name, lossy compression commonly losses data from the original image during compression, which might reduce the quality of image after being uncompressed. JPEG is one of the most popular lossy compression methodologies. Its standard is defined by the Joint Photographic Experts Group (JPEG) [1] where acronym makes the general name.

On the other hand, in lossless compression the data from original image is preserved after compression. Some commonly used methods of lossless encoding include run length encoding, which counts the number of bits replaced by taking identical and consecutive data elements. It replaces with a single value of the element and counts the replacement numbers. BMP [2] files effectively uses this method. DEFLATE [3] is another compression technique which uses Huffman coding and LZ77[2,4] which is suitable for PNG files.

In digital images JPEG is extensively used, although it's a lossy image compression method. When using with this algorithm, users can set the compression ratio in an image, which helps to choose the file size and image quality. The standard for the JPEG was created by two active groups, ISO (International Organization for Standardization) and ITU-T (International Telecommunication Group). The actual standard for JPEG algorithm was formally known as ISO/IEC IS 10918-1 | ITU-T Recommendation T.87 [1].

The JPEG algorithm, widely used for digital images and videos, works best when applied to photographs and paintings of realistic scenes. Pictures and videos need to have smooth variation of tone and color. Hence, JPEG does not fit well for lines, icons or any other type of textual contexts where the sharp contrasts between adjacent pixels might cause noticeable artifacts. Because of the lossy nature, JPEG is not good for medical applications and scientific visualization. Because in these cases the user might need exact reproduction of original image data.

JPEG leads to loss of image data after compression and creates artifacts in the uncompressed image. Figure 1 best illustrates the example of loss of data from

JPEG compression which produces artifacts and distortions. The left images a & c have higher amount of compression than the right images b & d. As a result, the images b & d shows a significant amount of loss of details comparing with rest of the image.

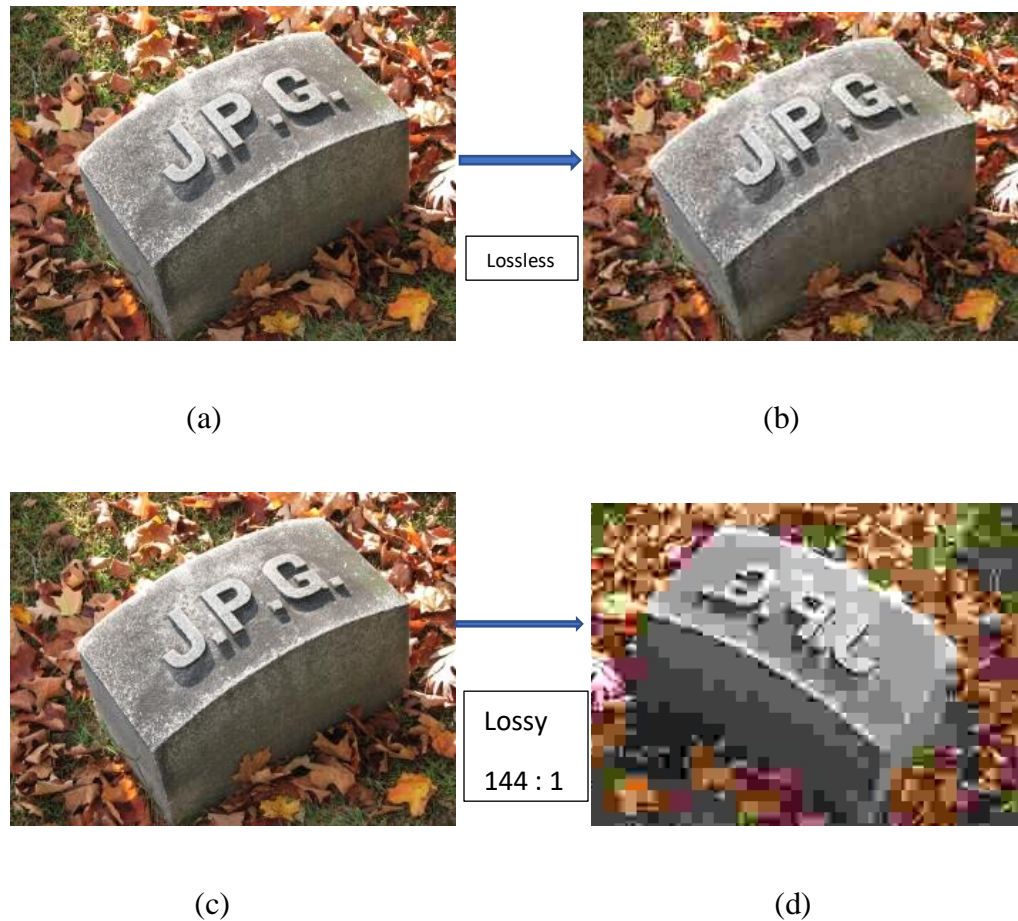


Figure 1. Loss of pixel data in lossy compression; Source: wikipedia.org

JPEG uses a Block-based Discrete Cosine Transform (BDCT) coding scheme. In JPEG an image is first divided into 8X8 non-overlapping blocks. Every block is then transformed using a Discrete Cosine Transformation (DCT), followed by quantization and variable length coding. Discrete cosine transformation is a Fourier-like-transform in which the basis function is only made up of cosines. Like

discrete Fourier transform, the DCT also operates on discrete and finite sequences. JPEG encoding is flexible for users allowing for tradeoffs between image quality and file size (compression ratio) which provides some control over the compression process. Higher compression ratios result in undesirable visual artifacts in the decoded image such as blockiness and Gibbs oscillations [5] (ringing artifacts near strong edges).

At this point of discussion, we need to find a way to reduce visual artifacts in the decoded images. Post processing is done frequently for this purpose. One approach is to use an adaptive spatial filter [6] or adaptive fuzzy post-filtering [6]. These techniques commonly involve classification and low pass filtering. Normally these techniques classify each block having strong edges or weak edges and then apply a set of predefined spatial filters. The effectiveness of this method is highly dependent on block characteristics and on specific filter design.

Riddhiman et al. [7] developed a post-processing algorithm that created a better-quality JPEG compressed image irrespective of the compression level. First, an adaptive low pass filter was applied as it can be done on any image irrespective of compression level. Second, an adaptive computationally efficient algorithm was used for reducing artifacts while preserving the level of quality in the images. Third, the original image was blended with a smoothed version of re-constructed image which is referred to as an alpha blend because blending is controlled by a limit factor that considers the amount of compression present and any local edge information derived from a Prewitt filter application. In addition, the value of the blending co-efficient ( $\alpha$ ) is derived from the local Mean Structural Similarity Index

Measure (MSSIM). The blending co-efficient is adjusted by a factor which considers amount of compression present in JPEG image.

The aim of this research was to improve on the results of Riddhiman's approach through artificial intelligence integration. To achieve this goal, a neural network was trained and applied. For making the training dataset, standard images (Lena, Peppers, and Baboon) with compression levels of 5% to 95% were used. Pixel values in the decompressed alpha blend post processed image were modified by the trained neural network. Post and preprocessing are two different type of approaches, but both can be used to improve image quality after JPEG compression. In this research, post processing was done on JPEG compressed images due to simplicity in implementation and less computational overhead.

Using a neural network to enhance image quality via post processing of the decompressed image is an advanced idea, as it does not require any prior uncompressed image data and it can work on any type of compressed image. Once trained the network can be used for any image without considering compression level.

## **CHAPTER II**

### **RELATED RESEARCH ON REMOVING ARTIFACTS FROM COMPRESSED IMAGE**

JPEG is a block transform algorithm. The core principle of this type of algorithm creates artifacts in the compressed image because it compresses images using a block transformation. In JPEG, the Discrete Cosine Transform (DCT) is applied to  $8 \times 8$  blocks of pixels, followed by a quantization of the transform coefficient of each block. The coefficients are quantized based on compression ratio. The greater the compression, the coefficients are quantized more coarsely. As quantization is applied to each  $8 \times 8$  block separately, the DCT coefficients of each block are quantized individually which causes discontinuities in color and luminosity between neighboring blocks. This phenomenon is especially common in the areas of the image where there is a lack of complex detail which can camouflage the artifacts. These artifacts are commonly referred to as blocking artifacts [30], as demonstrated in the figure 2. The left and right part of figure 2 are highly compressed and decompressed versions of the same image respectively. It's clear that the compressed image has a lot of blocking artifacts.





Figure: 2 Example of blocking artifacts; Source: Wikimedia commons

Another type of artifacts is ringing which is caused by loss of coefficients. To illustrate how this occurs, a short account of Gibbs's phenomenon is a required. Albert Michelson devised a method around 1898 to compute the Fourier series decomposition of a periodic signal. He then reproduced the original signal by resynthesizing those components from the decomposition. He found that resynthesizing a square wave always gave rise to oscillations at discontinuities. In fact, this result was consistent to that of several other contemporary mathematicians. But later, in 1899, J. Willard Gibbs shown that contrary to popular belief, these oscillations are not originally generated from the device's mechanical flaws. The overshoot of discontinuity is mathematical: no matter how many higher harmonics are considered. The oscillations never die down; they just approach a finite limit. This is known as Gibbs's Oscillation.

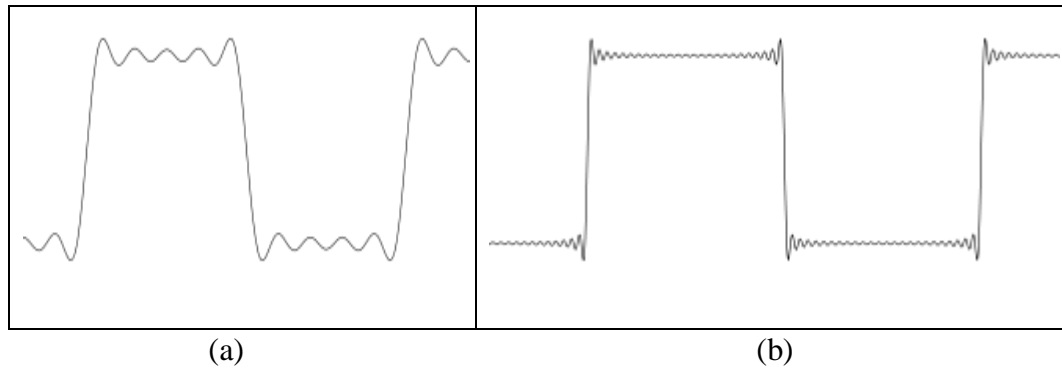


Figure: 3 Gibbs' oscillation; Source: Wikimedia commons

In Figure 3, the left-side figure, 3(a) has less high frequency than the right-side figure 3(b) wave. The high frequency component makes the right-side wave more square shaped. The high frequencies represent sharp contrast edges where there is a lot of pixel intensity variation. So, if we lose or truncate high frequencies during JPEG compression then the image will have ringing artifacts.

Modern displays use 24-bit pixel encoding which uses 8 bits to encode each color band (R, G, B). Because JPEG compresses each color band separately, JPEG does not work well for color representations with high correlation between bands (RGB bands). JPEG works well with color representations that have low correlation between bands (like YCbCr).

The JPEG encoding process consists of several steps:

1. Color Space Transformation: First, convert the image color space from RGB to YCbCr. This is done to allow greater compression as brightness information (Y), which is more important for perceptual quality of the image, is confined to a single channel.

2. Down sample: Reduce the resolution of the two chroma bands (Cb & Cr) by a factor of 2.
3. Block Splitting: Split the image into non-overlapping blocks of 8×8 pixels.
4. Cosine Range Transform: As we will work with cosine series, 128 needs to be subtracted from every value of this 8×8 block such that input values are both positive and negative as the DCT ranges from -1 to +1.
5. Discrete Cosine Transform: Each block is converted to a frequency-domain representation, using a normalized, two-dimensional type-II discrete cosine transform (DCT).

$$G_{u,v} = \sum_{x=0}^7 \sum_{y=0}^7 \alpha(u)\alpha(v)g_{x,y} \cos \left[ \frac{\pi}{8} (x+1/2) u \right] \cos \left[ \frac{\pi}{8} (y+1/2) v \right]$$

Where  $u = 0, \dots, N-1$  and  $v = 0, \dots, N-1$ .

6. Quantization: The high frequency components can be eliminated without much loss of image quality and we can reduce the number of bits needed to store the values from the DCT. The 8×8 DCT results are divided by values in a quantization table which are chosen to save the low frequencies and discard the high frequency components. When the DCT coefficients are divided by corresponding values from quantization table, ringing artifacts can occur. Quantization may also reduce the number of bits used to represent the coefficients, which may result in blocking artifacts.
7. Entropy coding: The outcome of the previous steps are compressed with a variant of Huffman encoding which is lossless compression.

Lee et al. [22] proposed a post processing technique that reduces blocking artifacts, staircase noise and corner outliers. Staircase noises are the result of image edges

which exist in the transform block of an image. This causes the edge to degrade and result in a formation of step-like artifacts along the image edge, as shown in Figure 4(a). Corner outliers are formed when, after the quantization of the DCT coefficients, the corner-most pixel in a transform block has a far greater or smaller value than its neighboring block. Figure 4(b) highlights a corner outlier pixel, which is easily distinguishable in figure 4(a) as its pixel value is far different than its neighboring pixels.

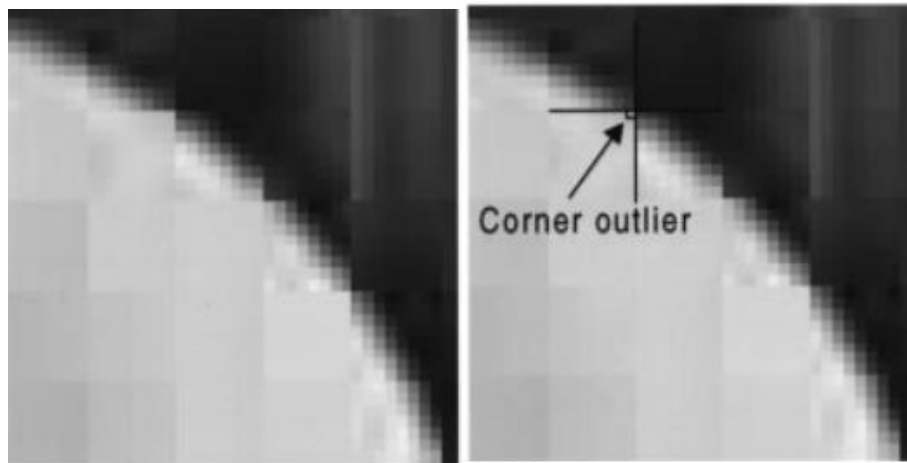


Figure 4 (a). Staircase Noise, (b). Corner outlier; Source: Lee et al [22]

In their algorithm they used a four-step process. First, they used an edge detector operator and a Sobel operator to create an absolute gradient magnitude of the image. Then an edge map was done by thresholding the gradient image obtained from the Sobel filter. This is followed by a classification of the compressed image into an edge area or monotone area based on the edge map. Similarly, a local edge map is also generated for each 8x8 transform block. A global and local threshold value was then calculated.

If the  $n^{\text{th}}$  block does not contain much variation, then the local threshold is close to the global threshold. But if it contains enough variation, the ratio  $\frac{\partial n}{mn}$  increases and generates local threshold value which is much smaller than global counterpart. Where  $\partial n$  is the standard deviation and  $mn$  is the mean of the  $n^{\text{th}}$  block of the gradient image generated by the Sobel filter. Secondly, a smoothing filter was used to remove blocking artifacts and staircase noise along the edges of the image. This algorithm applies 1-D directional smoothing filter along edges on all points on the edge map. The direction of all edges on edge map is calculated as:

$$\theta_e(x, y) = Q[\theta(x, y)] - 90^\circ$$

Where,  $\theta_e(x, y)$  is the direction of edge for location  $(x, y)$ ,  $Q$  is the quantization factor and  $\theta(x, y)$  is the direction of gradient vector calculated by applying Sobel filter. The purpose of this filter is to remove staircase noise generated along edges in the image.

After that an adaptive 2-D low pass filter was applied to the image. This process removed blocking artifacts which were produced within areas of image that does not have too much variance.

According to their filtering, if the center pixel of  $5 \times 5$  block, contains an image edge, then no filtering is required. The procedure to check if an edge exists in the center from global or local edge map of a block is the following:

- a. If no edge pixel is contained within the block, center or anywhere, then average mean filtering is applied to the block by convoluting with a Kernel.

- b. If the block contains an edge pixel, not necessarily in the central point but the neighboring pixels close to center then a weighted average filter is applied to the block by setting the pixels with edges and their neighboring pixels close to 0, followed by convoluting the remaining pixels of the block with the weights from the kernel and calculating the average of the block.

Finally, a  $2 \times 2$  block window is used to detect and remove any corner outliers in the image.

Singh et al. [13] proposes a novel procedure that models a blocking artifact between two neighboring blocks of pixels. Based on the Human Visual System (HVS), using the model, they detect the presence of any blocking artifacts and removed them by filtering adaptively.

The HVS model is based on the human perception process for images, luminosity and color. In video and image processing, visualization techniques for angel of view, resolution, sensitivity and detail of the image is often used to take advantage of human vision capabilities. The model suggests that the human eye can perceive changes in luminosity better than changes in color, but cannot easily perceive high frequency details of an image, thus allowing high frequency components to be quantized without noticing loss in image data.

For two adjacent blocks,  $b_1$  and  $b_2$ , after DCT quantization, because of independent quantization of each block, an artifact can be created between this pair. The artifacts can be simulated as a new block created from the existing two blocks  $b_1$  and  $b_2$ .

The step function is defined as:

$$s(i, j) = \begin{cases} -1/8, & \forall i \in [0,7], j \in [0,7] \\ 1/8, & \forall i \in [0,7], j \in [0,7] \end{cases}$$

And the new block is derived as:

$$b(i, j) = \beta s(i, j) + \mu + r(i, j), \forall i, j \in [0,7]$$

where,  $\beta$  is the amplitude of  $s$ , the 2-d step function,  $\mu$  is the mean of the block  $b_1$  and  $b_2$ , and  $r$  is the residual block which describes the local activity around the block edge.

The classification of blocks as smooth or non-smooth was done based on frequency properties. If two blocks,  $b_1$  and  $b_2$ , have similar frequency properties and block  $b$ , which is comprised of edges between  $b_1$  and  $b_2$ , does not contain high frequency components, then  $b_1$  and  $b_2$  are classified as smooth regions and vice versa.

This above method is not applicable for a non-smooth block, because it will increase artifacts present in the compressed image. In this case, a smoothing sigma filter is used. The sigma filter smooths noise by averaging neighborhood pixels based on sigma probability of Gaussian distribution. A  $5 \times 5$  window was used for this filter.

Singh et al. [27] further improves the above work by classifying the blocks as smooth, non-smooth and intermediate, and using an adaptive pixel replacement algorithm. This approach improves on previous attempts in preserving details of the image with minimum loss of image data. This technique reduced the complexity and computational overhead by a considerable extent.

Filtering is applied to the compressed image based on classification of the area of image, and its frequency properties. The smooth regions contain low frequency components, non-smooth areas contain high frequency and the regions classified as intermediate contain mid-range frequency components. To determine whether an area of two adjacent 8×8 blocks, b1 and b2, is smooth, non-smooth or intermediate, the activity across block boundary is measured by taking eight pixels into account, four on either side of the block boundary. The variation in pixels within b is calculated as:

$$A(p) = \sum_{k=1}^7 \phi(p_k - p_{k+1})$$

$$\phi(\Delta p) = \begin{cases} 0, & \Delta p \leq \text{threshold} \\ 1, & \text{otherwise} \end{cases}$$

where, A(p) is the block boundary activity and p are the pixel intensities of the block b. The number of pixels to be modified while filtering depends on the type of region being filtered.

Saig et al. [1] proposed the use of decimation filters and optimal interpolation in block coders. Like many other algorithms which depend on a block-based approach, typically block coders are high speed and low-complexity, and perform reasonably well, but suffer from creation of artifacts in the decoded image while processing images of low bitrate.

The proposed algorithm works in two parts: initially, determining an optimal framework for an interpolation filter (g), and finally, determining an optimal



framework for the decimation filter (f) with respect to the previously determined optimal interpolation filter (g).

If we apply the optimal interpolation (g) and decimation (f) filters in the block coder algorithm, we will obtain a reasonable improvement in quality over the original decoded image.

Kieu et al. [2] proposed a technique using a ternary polynomial surface to reduce blocking artifacts, created in the low activity regions of the image. This recovered the image details lost during quantization by compensating the DCT co-efficient.

In ternary surface modeling, the image intensity for each 2×2 macro block was calculated, and linear programming techniques are applied to minimize the difference between pixel values across the block boundary of the macro block. The quantization error resulted in the image after the JPEG compression.

Aboud [1] presented a simple adaptive low-pass filtering approach that reduces blocking artifacts in the image without degrading the quality. This approach exploits a property of HVS, in that human vision is more sensitive to blocking artifacts present in smoother areas rather than those that have a lot of activity. This approach classifies regions of image into highly smooth, relatively smooth and highly detailed and then applies strong filtering and weak filtering respectively.

To classify different regions of the image, the function below is used:

$$count = \Phi(v_0 - v_1) + \Phi(v_1 - v_2) + \Phi(v_2 - v_3) + \Phi(v_4 - v_5) + \Phi(v_5 - v_6) + \Phi(v_6 - v_7)$$

$$\Phi(\Delta) = \begin{cases} 0, & \Delta \leq threshold \\ 1, & otherwise \end{cases}$$

where,  $V_0$  to  $V_7$  are adjacent values along the edges in  $8 \times 8$  block, as shown in figure 5. If the count equals 6, the classified area is very smooth. If the count falls between 1 and 5, then the area is classified as relatively smooth. If the count is 0, the area is classified as highly detailed.

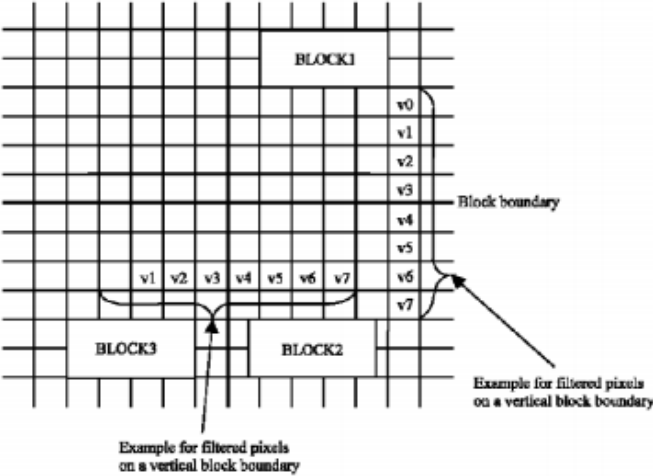


Figure 5. Adjacent blocks of pixels

For highly detailed blocks of an image, a low amount of filtering is applied using a higher factor ( $a = 0.5$ ). A moderate amount of filtering is applied using a moderate factor ( $a = 0.4$ ) to relative smooth blocks. Finally, the blocks which are very smooth, strong filtering is applied using a low factor ( $a = 0.3$ ). Filtering is applied to horizontal and vertical block boundaries respectively. Post processing is commonly applied in two ways. The algorithm first applies the filtering along the vertical boundaries of the block, followed by the horizontal boundaries.

After a count has been calculated, if it is 6, the above equations are calculated for  $V_3$  and  $V_4$  using  $h(n)$  for  $a = 0.3$ ,  $v_2$  and  $v_5$  using  $a = 0.4$  and  $v_1$  and  $v_6$  using  $a = 0.5$ . If the count falls between 1 and 5, indicating the region to be relatively smooth,

V3 and V4 are calculated using  $a = 0.4$  and V2 and V5 are calculated using  $a = 0.5$ . If count is 0, then the region is complex in nature and only V3 and V4 are calculated using  $a = 0.5$ .

The second algorithm follows all the steps of the first one, differing only when dealing with the pixels that are filtered both vertically and horizontally. The horizontal and vertical filtering is applied independent of the each other for those pixels and the mean of the two values is chosen.

Liaw et al. [12] proposed an approach for image deblocking and restoration based on Classified Vector Quantization (CVQ). In CVQ, code words (stored in a codebook) are generated from a set of training vectors and used for both the encoding and decoding process. Before applying CVQ, a deblocking algorithm was applied to the image. The algorithm worked in two steps: classification and deblocking. The image was down sampled into  $8 \times 8$  blocks for classification into smooth and non-smooth based on each block's DCT coefficients. Following the classification, deblocking is applied to the block  $B_{m,n}$  and its neighbors. If  $B_{m,n}$  and its neighboring block  $B_{m+1,n}$  are smooth, then a 1-D filter  $\{1,1,2,4,2,1,1\}$  is applied to one half of the block  $B_{m,n}$  and the adjacent half of the neighboring blocks  $B_{m+1,n}$ .

Traditionally, vector quantization uses a single codebook for both encoding and decoding, and the generalized Lloyd algorithm [12] is used to generate this codebook. Once both codebooks are ready, then restoration of the image is applied via further classification. The images are broken into  $4 \times 4$  blocks, and the mean of the pixel intensities of each block is calculated and is used to classify the blocks into different sub-categories: uniform, texture and edges. Further classification is

required if the block is non-uniform to determine whether it falls under an edge class or a texture class. The edge orientation is changed if the block belongs to the edge class.

The process of restoration of the image, which consists of encoding and decoding, then starts. For encoding processing, the mean of a block of the image is determined and, the block is classified and sub-classified into a respective category. For non-edge class, a codeword is determined from the corresponding class of codewords in the codebook. In case the block does not belong to any edge-class, then the edge direction is calculated. The applicable codeword is retrieved from the codebook based on the type, class and direction. When a suitable codeword is found, it is subtracted from the input block to calculate a differential vector, the codeword index and the differential vector are recorded. If no suitable codeword is found, then the block is not changed. This process is repeated until all blocks have been encoded. Decoding for image restoration works similarly. The mean values are obtained from indices of recorded blocks. The class of the block is determined from mean value. For non-edge type blocks, a respective codeword is determined from the respective codewords in the codebook, which is determined from class and block information. If the block belongs to an edge-class, then an edge orientation is calculated. The relative codeword is retrieved from the codebook, which is determined from class and block information. This is done repeatedly until all blocks are decoded. The image quality can be restored by combining encoding and decoding blocks of compressed image.

Chou et al. [9] presented a simple post-processing algorithm that attempts to estimate the quantization error present in the JPEG compressed images. They modeled each DC coefficient (of the DCT) of all the blocks of an image as Gaussian random variables, and each AC coefficient (of the DCT) as zero-mean Laplacian random variables. They applied a probabilistic framework to estimate the mean square error of each DCT coefficient. The estimated quantization error for each  $n \times n$  block is the mean squared quantization error in the DCT domain which is equal to the mean squared error in the spatial domain. They determine row and column vectors of discontinuing pixel intensities across the image. They used a threshold value determined from the previous mean square error. They then attempted to identify the discontinuities caused by quantization error. After identifying any anomaly in pixel intensities, a new pixel intensity was calculated. The new intensity for relevant pixels was determined by using a proportionality constant derived from the threshold value and the mean of the image.

As shown in the picture 6, the algorithm has low complexity and returns impressive results from the Lena test image. The compressed image is on the left and filtered result is on the right.



Figure 6. Lena, before and after

Riddhiman et al. [7], proposed an alpha-blend algorithm which is computationally efficient for reducing blocky and Gibbs oscillation artifacts from JPEG compressed image. The goal of the alpha-blend filter is to reduce noise and artifacts in the image without losing image details. An adaptive limit value was calculated from the compression ratio of the image being filtered to decide whether a pixel needs to be altered or not. A Prewitt filter was used to derive an edge map. After edge map creation, the original image was down sampled into  $12 \times 12$  blocks of pixels centered on  $8 \times 8$ -pixel blocks. For each  $12 \times 12$  block, the corresponding pixels are compared with a limit value. For an image edge pixel, no processing is applied. However, for pixels less than the limit value, pixel alteration is done as follows.

$$M[i][j] = \text{ROUND} ((1.0-\alpha) * M[i][j] + \alpha * \text{LowPass}[i][j])$$

where,  $\alpha$  ranges between 0 and 1 and is calculated from the MSSIM (Mean Square Similarity Index Measure), LowPass is the image (M) after application of a low pass filter. The  $12 \times 12$  block window in the image, moves horizontally and

vertically until it reaches the extreme edge of the image. Then it moves back to its original position and shifts one pixel down vertically followed by a horizontal shift again. This is done repeatedly until all pixels in the image have been processed.

The original alpha blend filter can be summarized by the flowchart in figure 7. First a low-pass filter is applied to the image and the result is stored separately. The next step is to use a two pass Prewitt operator creating the edge map. The Prewitt filter is one of the most popular edge detection operators. It uses vertical and horizontal kernels to calculate the approximate derivation of image pixels in both horizontal and vertical directions.

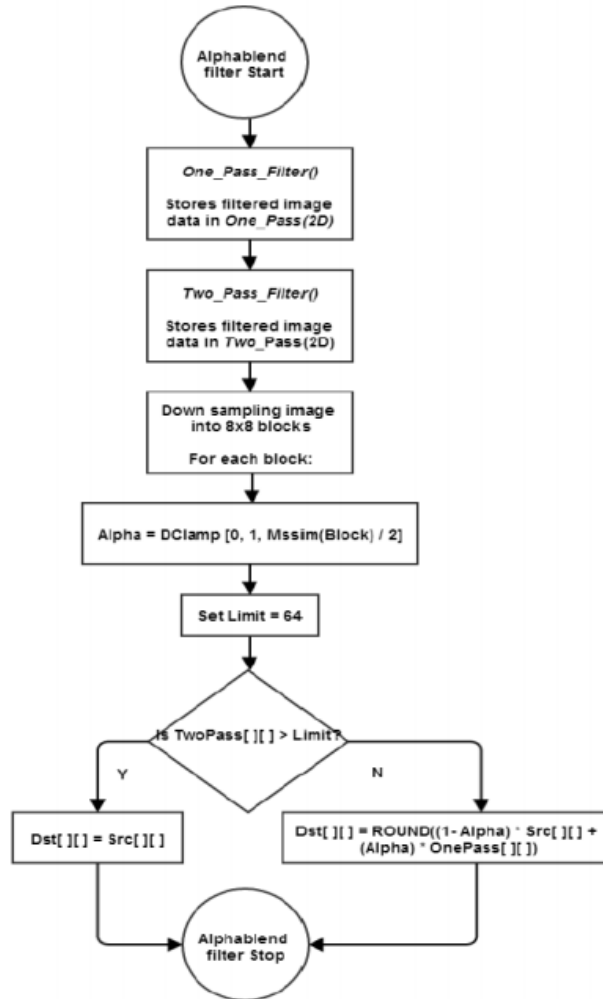


Figure 7. Flowchart of the alpha blend algorithm

$$G_x = \begin{bmatrix} -1 & 0 & +1 \\ -1 & 0 & +1 \\ -1 & 0 & +1 \end{bmatrix} * A \quad G_y = \begin{bmatrix} -1 & -1 & -1 \\ 0 & 0 & 0 \\ +1 & +1 & +1 \end{bmatrix} * A$$

Figure 8: The Prewitt filter.

In Figure 8,  $G_x$  is the horizontal and  $G_y$  is the vertical gradient kernel of the Prewitt filter. The  $*$  operator implies a convolution operation between the image ( $A$ ) and the matrix ( $G_x$  and  $G_y$ ). The magnitude of the gradient for each pixel of  $A$  is derived as:



$$\text{result } [x] [y] = \sqrt{G_x^2 + G_y^2}$$

Their initial algorithm used the original image to calculate the MSSIM. They changed the algorithm such that it now uses the compressed image instead of the original image to calculate the MSSIM. After that, a test image set was created based on compression ratio, ranging from 5% to 95 %. Then  $\alpha$  value was determined as follows:

$$\alpha = \text{CLAMP} \left( 0.0, 1.0, \frac{\text{MSSIM}(x,y,M)}{C} \right)$$

where, alpha ( $\alpha$ ), ranges between 0 and 1 and is calculated from the MSSIM of the block containing pixel i and j in image M. C is the image compression ratio constant. The value of C ranged between 0 and 10, incremented by 0.1. The PSNR (Peak Signal to Noise Ratio) values of all results for each value of limit (from 1 to 255) was recorded and the best result was selected for each image. The objective of this iteration was to determine a best value of the constant (C) for a given compression level. Allowing a function to be created for different bpp (bits per pixel) values of the images, as the bpp might vary based on the compression level of an image. A brute force approach was used, to calculate C in the alpha function taking the adaptive limit into consideration. The value of C ranged between 0 and 10, incremented by 0.1, and applied to all images in the test set. The resulting data was curve fitted using the Matlab curve fitting tool. This resulted in:

$$C = 5.243 * e^{-((bpp - 2.414) 1.224)^2} + 3.374 * e^{-((bpp - 1.057) 0.8201)^2}$$

A second brute force method was applied, with respect to the new adaptive C value. In this stage, the value limit which had been fixed at 64, was varied from 1 to 255, and was applied to all images in test set. The denominator C in this function was dynamically calculated from the compression ratio of the image. The PSNR values for all the results for each value limit from 1 to 255 was recorded. Then, the best result was selected for each image. The best result for every compression level was selected. The resulting data was curve fitted using the Matlab curve fitting tool. This resulted in:

$$\text{Limit} = 34.12 * \text{bpp}^{-0.8432} + 42.11$$

Finally, their results showed significant improvement of images after post processing. Their overall goal was to derive values for C and L that would allow the alpha-blend algorithm to adjust images of different compression ratios which would reduce blockiness and artifacts.

### **CHAPTER III**

#### **IMPLEMENTING NEURAL NETWORK IN JPEG IMAGE POST-PROCESSING TO IMPROVE QUALITY**

The main objective of image compression is to reduce storage and transmission cost while maintaining image quality. DCT transformation is widely used for compression of images due to relative ease of implementation. JPEG image compression is based on 8×8 non-overlapping blocks of pixels, where each block is transformed, quantized, and encoded independently. In high compression ratios, high frequency components are removed from these blocks, which causes artifacts.

We used a neural network to restore degraded JPEG compressed images. The neural network was trained on standard original images: Lena, Baboon, and Peppers. Then the trained network is used to post-process any other JPEG image to improve quality.

For image quality assessment the Mean Square Similarity Index Measure (MSSIM) and Peak Signal to Noise Ratio (PSNR) are used. PSNR is defined as ratio between maximum possible power of a signal and maximum power of intermingled noise that corrupts the original signal. It is used for image processing experiments as a measure of quality for reconstructed images from lossy algorithms. Here, in our experiment, the original signal comes from the uncompressed prior image data and noise is the accumulation of artifacts caused by the loss of data due to compression.

For an image of dimension  $m \times n$ :

$$MSE = \frac{1}{mn} \sum_{i=0}^{m-1} \sum_{j=0}^{n-1} [I(i, j) - K(i, j)]^2$$

MSE (Mean Square Error) is the error estimate between the original image I and the final processing image K. From this equation, PSNR can be calculated as:

$$PSNR = 10 \log_{10} \left( \frac{MAX_1^2}{MSE} \right)$$

where, MAX is maximum pixel value in original image I.

MSSIM (Mean Square Similarity Index Measure) [13], is the method of measuring similarity between two images. MSSIM, as a full reference metric, references the uncompressed data of the original image to determine quality of compressed image.

SSIM is calculated over  $n \times n$  blocks of pixels using the following equation:

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

Using the above equation, MSSIM of an image can be derived by calculating the mean SSIM of all  $n \times n$  blocks in the image.

Here,  $\mu_x$  is the average pixel value of image pixel block x,  $\mu_y$  is the average pixel value of image pixel block y,  $\sigma_x^2$  is the variance of image pixel block x,  $\sigma_y^2$  is the variance of image pixel block y,  $\sigma_{xy}$  is the covariance of image pixel blocks x and y, and c1 and c2 are constants.

A neural network consists of inter-connected nodes called neurons. Each input neuron takes one piece of input data, in this case, one pixel from an input image

and applies a computation to generate results. The inter-connections have numerical weights which are initially set up with random numbers.

A neural network consists of connections between the layers. Each connection provides the input to the next layer by passing the output from current layer. Each connection must have some weight to represent the relative importance. A neuron can have many input and output connections. A sample neural network is presented in Figure 9.

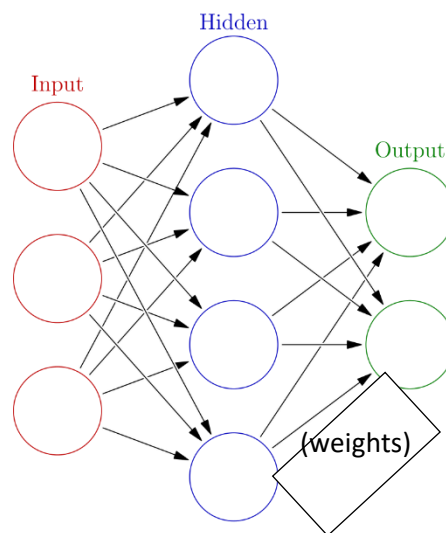


Figure 9: An Artificial Neural Network

The learning process adapts the neural network to produce an expected output by observing samples from a training dataset. While learning, the neural network adjusts the connection weights to improve the accuracy of the result. The accuracy is measured by an error function. The error function compares the network output

and pre-defined expected output to measure the error. Learning stops when the error does not significantly change in next iteration of training.

There are two types of learning, supervised and unsupervised. In our research we used supervised learning with back-propagation. In supervised learning, a paired input and output set is used. The goal is to produce a desired output for a given input. Learning involves adjusting the weights of the network to improve the output of network. Back-propagation supervised learning has following schema:

Input --> Forward Calls --> Error Function --> Derivative of Error --> Back-propagation of Errors

Here, the input consists of the values given to the input layer, forward calls are function calls during the forward pass of the data which take the input values to the output layer via hidden layers, the error function compares the network output with the expected output, derivation of error comes from calculating the derivative of the error function with respect to the weights of the network, Back-propagation of errors calculates the gradient of the error function with respect to the neural network's weights. Finally, after each stage we modify the weights according to:

$$\text{New weight} = \text{Old weight} - \text{Derivative} * \text{Learning rate}$$

The error function in classic back propagation is the mean square error like following equation.

$$E(X, \theta) = \frac{1}{2N} \sum_{i=1}^N (Z_i - y_i)^2$$

where  $y_i$  is the targeted value for input-output pair  $(x_i, y_i)$  and  $y_i$  is output from the network on input  $x_i$ . Backpropagation attempts to minimize the above error

function with respect to the neural network's weights by calculating for each weight update by the following equation :

$$\Delta w_{ij}^k = -\alpha \frac{\partial E(X, \theta)}{\partial w_{ij}^k}$$

where  $w_{ij}^k$  is the weight for node  $j$  in layer  $k$  for incoming node  $i$ ,  $\alpha$  is the neural network learning rate,  $\partial$  function defines changes.

As weights are changed in each iteration (epoch), the value of the error function is changed. The error function used in our case is the Mean Square Error, which represents the average square error between the network output and expected output.

As JPEG image compression is based on  $8 \times 8$  non-overlapping blocks of pixels, we can divide the whole image into  $8 \times 8$  blocks of non-overlapping sub images. For adjusting pixels, from the degraded image to the original image, we consider partially overlapping  $12 \times 12$  blocks of pixels centered on non-overlapping  $8 \times 8$  blocks. Figure 10 presents a  $12 \times 12$ -pixel block centered about an  $8 \times 8$  block.

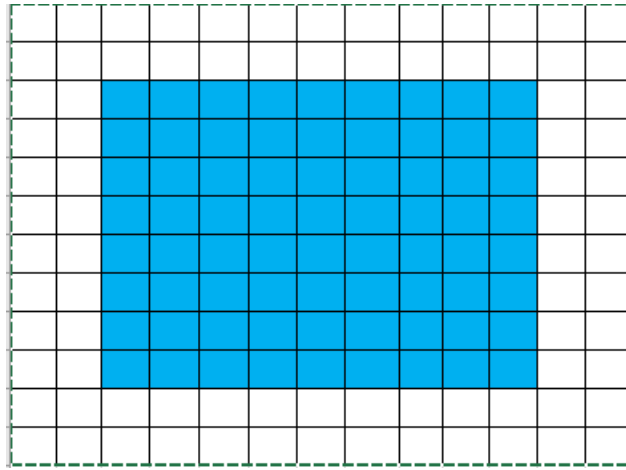


Figure 10: The 8×8 centered pixel block surrounded by a 12×12-pixel block

Before starting the neural network training process, we need train and test datasets. The test and train datasets were generated from the alpha blend output images and original (uncompressed) images. For our experiment we chose Lena, Peppers, and Baboon pictures to train the network. Figure 11 shows neural network training configuration based on Alpha blend output image and Original uncompressed image.

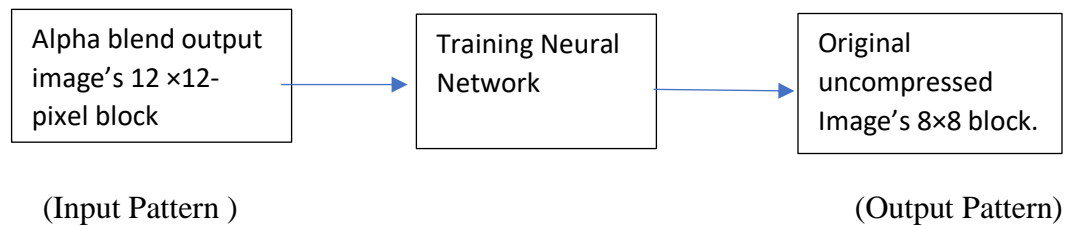


Figure 11: The Neural Network training configuration

At first, an input and output pattern dataset file named “pixel.data” was made from the original and degraded images. While making the dataset file, a partially overlapping 12×12-pixel block window was chosen and was moved through all pixels in each alpha-blend output image. From this 12×12 block window, 144-pixel



values were taken as input. For selecting output, an  $8 \times 8$  non-overlapping block pixel window, centered on the corresponding  $12 \times 12$  overlapping block, were taken from the original image. Figure 12 has two overlapping  $12 \times 12$ -pixel blocks where each is centered on non-overlapping  $8 \times 8$ -pixel blocks.

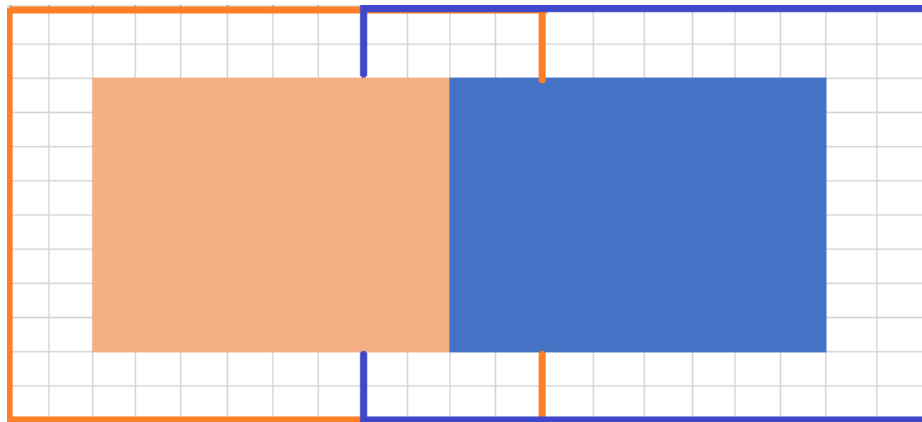


Figure 12: Two overlapping  $12 \times 12$ -pixel blocks from input image.

The orange and blue solid colors represent two  $8 \times 8$  non-overlapping blocks. The orange and blue outlined boundaries represent the  $12 \times 12$  overlapping pixel blocks. The  $12 \times 12$  block pixel values are the input values and  $8 \times 8$  block pixel values are the output values in the train or test patterns. The  $12 \times 12$  block pixel values were taken from the degraded image and then recorded as input data for a pattern in the “pixel.data” file. The  $8 \times 8$ -pixel block values were taken from the original image and then recorded as the desired output data for a pattern in the “pixel.data” file.

Finally, after moving the two windows from left to right and top to bottom the “pixel.data” file had all possible input and output patterns. Then the “pixel.data” file was randomly split into test and train pattern files. The train.pat file has the

training dataset and the test.pat file has the validation dataset. For our research, we set the number of input units to 144 ( $12 \times 12$ ). We set the number of output units to 64 ( $8 \times 8$ ).

Our work attempts to decide what values the center pixels of the  $12 \times 12$  blocks (the  $8 \times 8$  blocks) of a degraded image should have by comparing the trained pixel block from the neural network. Figure 13 shows trained neural network working process on reconstructing improved images with input from Alpha blend output images.

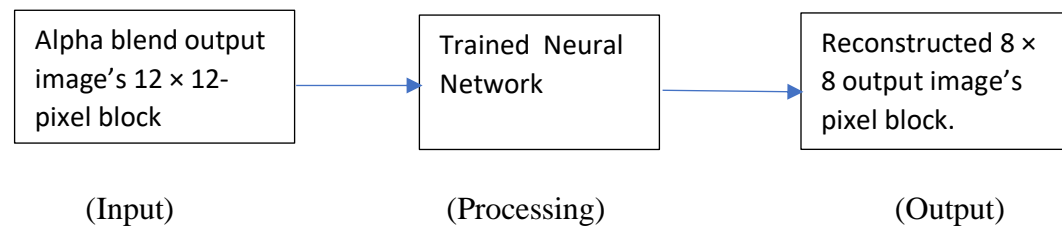


Figure 13: The trained Neural Network working process

We used SNNS (Stuttgart Neural Network Simulator) as artificial intelligence tool in our experiment. For this purpose, we first need to make test and train files, which is already done. Now, the network needs to be trained from the train file and validated by the test file.

SNNS is a neural network simulator developed by the Universität Stuttgart. It can be used to generate, train, test and visualize artificial neural networks. This simulator has a graphical user interface, a simulator kernel, and a compiler to generate neural networks from a high-level network description language.

To train a network, we first load test.pat and train.pat files in into SNNS. After that, other required parameters (in “control panel” and “bignet”) are filled up to start

training a network and testing for respective cases. SNNS can configure a neural network based on the input parameters specified in the “bignet” and “control” panel by going through the Manager Panel of the SNNS simulator GUI. By clicking the BIGNET button in the Manager Panel (figure 10) and selecting “general” from list of options, we would get a window like figure 14.

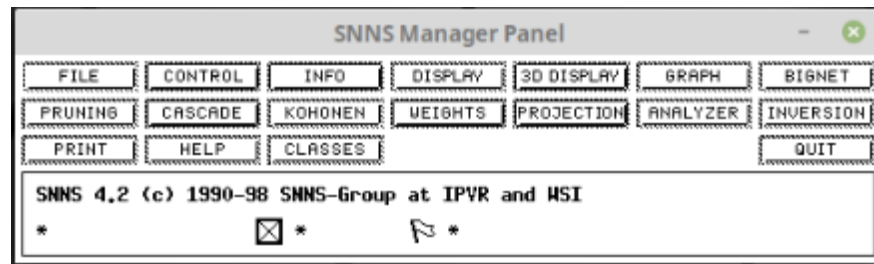


Figure 14: SNNS “Manager” Panel

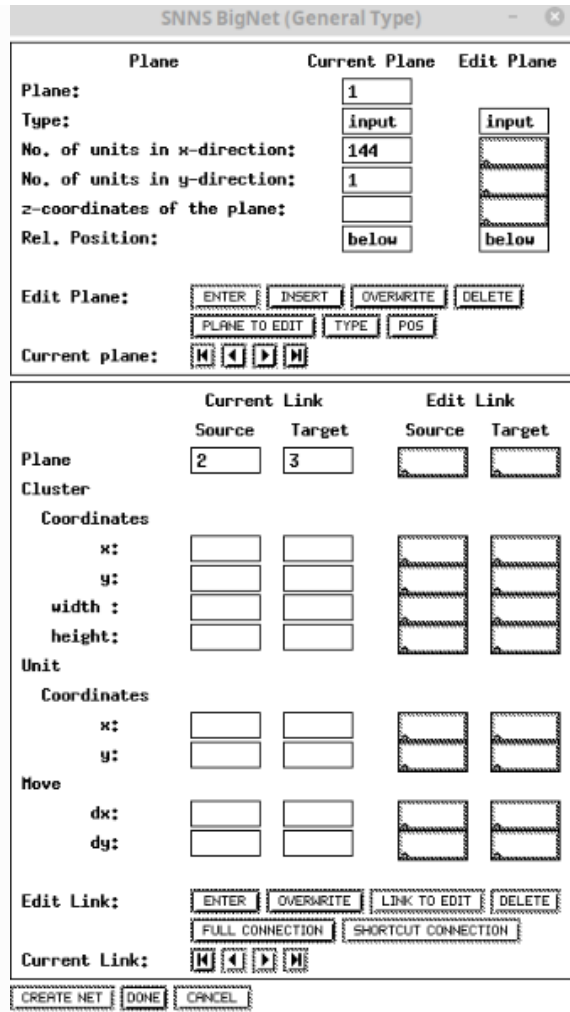


Figure 15: SNNS “Bignet” Panel

In the top section of the figure 15, we would define the nodes of the network, the bottom section is to define the connections. With this interface anyone can create customized layers of a network.

The number of input nodes represent the number of independent variables in the problem, the number of hidden layers is the measure of the nonlinearity of the function, and the number of output nodes is the number of dependent variables. As already mentioned, we need 144 input nodes and 64 output nodes. There was one

hidden layer with 96 nodes. Figure 16 shows the SNNS Control Panel setting for our research.

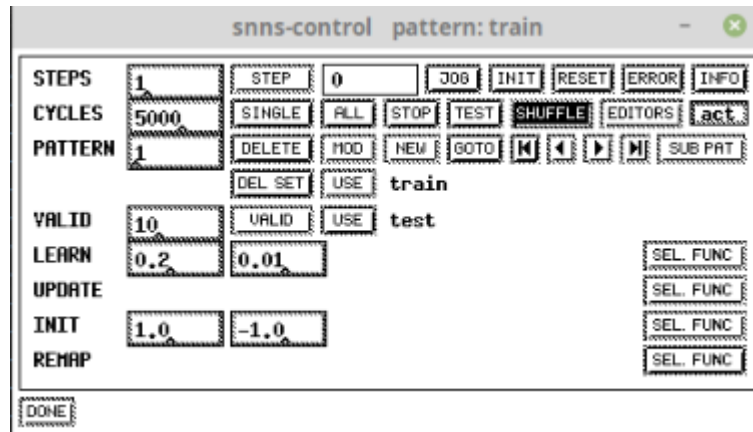


Figure 16: SNNS “Control” Panel

We then click the CONTROL button in the Manager Panel. The most important buttons here are CYCLES, VALID, LEARN, SHUFFLE, INIT, ALL and USE. The ALL button starts the training using all the patterns in the current training set. The editable field next to VALID has currently 10, which means every 10 epochs SNNS will show the validation patterns to the network and will calculate the SSE for the validation pattern set. The USE button is to select train and test files from the selection list. The field next to LEARN button is for learning rate which we set to 0.2 and the second field is the momentum which was set to 0.01. These values were selected as they were that recommended in the SNNS manual. The number of epochs SNNS would use is determined by the value in the editable CYCLES field. If the SHUFFLE button is highlighted, then SNNS will use random patterns from the train file before each cycle starts. We used 5000 cycles (epochs) and shuffled

the patterns from the train and test files. The INIT button will initialize training and set the weights to small random values.

Each iteration is defined as a cycle (epoch). The connection between nodes are called weights. Backpropagation is a learning algorithm that iteratively alters the value of the weights until the error function is minimized. The error function being minimized also depends on the user. In SNNS there is no build-in option for selecting the error function, by default it uses the Mean Square Error for measuring the error which we will utilize in our experiment.

Figure 17 shows the SNNS graph window, which shows the training and testing errors. After applying random train and test patterns to the network, when there was little change in the training error (MSE) the training was allowed to stop.

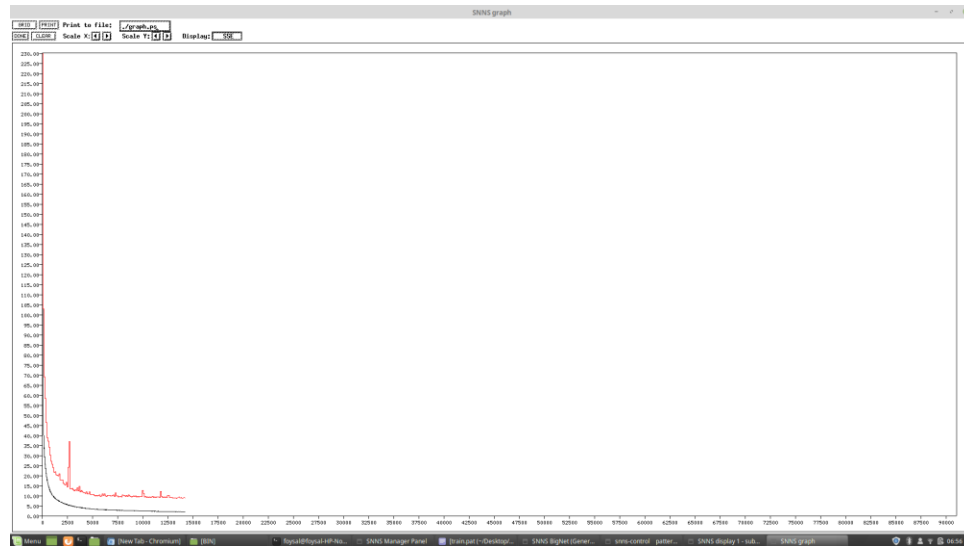


Figure 17: SNNS “Graph” Panel

Once the snns stopped running epochs, we saved the trained network for further usage in our code.

At this stage, we have a neural network trained from the original image and alpha blend output image. This network will take  $12 \times 12$  overlapping blocks of pixels as input from the alpha blend output image and, based on the training, will make an improved  $8 \times 8$  centered block of pixels respective to the  $12 \times 12$ -pixel blocks input from alpha blend output image. This  $12 \times 12$  block window was moved from top left to bottom right corner of both the input image and the under-construction output image by adjusting the centered  $8 \times 8$  block.

## **CHAPTER IV**

### **EXPERIMENTAL RESULTS**

To train the neural network, we used three images of Lena with compression levels of 8%, 19% and 37%, one image of Peppers with a compression level of 40% and one image of Baboon with a compression level of 50%. Then, we used the trained neural network on Reddhiran's alpha blend output images. We used 11 Lena JPEG images with compression levels from 5% to 95% as input to the alpha blend process with the same bit-per-pixel value for each image that Reddhiran used. Reddhiran kept bit-per-pixel values the same in the images to allow direct comparison from other published papers. The output of each input image from SNNS neural network showed improvement over the same image from previous experimental results in PSNR (Peak Signal to Noise Ratio) and MSE (Mean Square Error) values. Table 1 shows the results after neural network integration. For assessment of image quality, Mean Square Error (MSE) and Peak Signal to Noise Ratio (PSNR) were used. The higher the PSNR, the better the quality of images in post-processing. Likewise, the lower the MSE, the better the quality in post-processing.

The results obtained by several other authors, Reddhiran's alpha blend, and the neural network are also displayed in table 1. To compare results from the trained neural network, the input images were used with compression ratios similar in bit-per-pixel values to the images used in published results.



Table 1. Results of SNNS neural network

paper	Com- pression	original		alpha blend		%	SNNS		%
no	bpp	PSNR	MSE	PSNR	MSE	increase	PSNR	MSE	increase
11	0.158	28.77	83.48	29.64	68.35	3.02	29.85	65.23	3.75
14	0.15	28.77	83.48	29.64	68.35	3.02	29.85	65.23	3.75
30	0.15	28.77	83.48	29.64	68.35	3.02	29.85	65.23	3.75
32	0.169	29.34	73.18	30.17	60.4	2.83	30.41	57.25	3.65
33	0.16	29.34	73.18	30.17	60.4	2.83	30.41	57.25	3.65
11	0.188	29.82	65.55	30.61	54.63	2.65	30.87	51.6	3.52
23	0.189	29.82	65.55	30.61	54.63	2.65	30.87	51.6	3.52
32	0.187	29.82	65.55	30.61	54.63	2.65	30.87	51.6	3.52
32	0.209	30.28	59	31	49.88	2.38	31.28	46.86	3.3
35	0.2	30.28	59	31	49.88	2.38	31.28	46.86	3.3
20	0.217	30.63	54.33	31.34	46.19	2.32	31.64	43.21	3.3
9	0.24	31.3	46.61	31.95	40.11	2.08	32.27	37.32	3.1
11	0.24	31.3	46.61	31.95	40.11	2.08	32.27	37.32	3.1
14	0.24	31.3	46.61	31.95	40.11	2.08	32.27	37.32	3.1
30	0.24	31.3	46.61	31.95	40.11	2.08	32.27	37.32	3.1
37	0.25	31.3	46.61	31.95	40.11	2.08	32.27	37.32	3.1
11	0.318	32.65	34.16	33.17	30.28	1.59	33.53	27.91	2.7
35	0.3	32.65	34.16	33.17	30.28	1.59	33.53	27.91	2.7
14	0.43	34.13	24.24	34.49	22.32	1.05	35.04	19.75	2.67
37	0.5	34.75	21	35.07	19.5	0.92	35.56	17.5	2.33
11	0.626	35.66	17.06	35.94	15.98	0.79	36.2	15.12	1.51
11	0.997	37.62	10.86	37.8	10.4	0.48	37.88	10.04	0.69

In the above table, the leftmost “paper no” column gives the published paper reference numbers, the “Compression bpp” column shows the bit-per-pixel (compression) value of the image, the “original PSNR” column lists the actual PSNR values obtained from the respective paper’s experiment. The “MSE” column to the right of “original PSNR” columns are the “MSE” values obtained from the respective paper’s experiment. The “alpha blend PSNR” column shows the PSNR values obtained from Reddhiran’s experiment. The “MSE” column, to the right of

“alpha blend PSNR” column, lists the MSE values obtained with Reddhiran’s alpha blend filter. Reddhiran calculated the percentage of improvement using following equation:

$$(\text{Percentage}) \% \text{ increase} = \frac{\text{alpha blend PSNR} - \text{original PSNR}}{\text{original PSNR}} \times 100$$

The “(percentage) % increase” column displays the improvement of Reddhiran’s alpha blend PSNR over the original paper’s PSNR. The “SNNS PSNR” column shows the PSNR values obtained from the trained neural network’s output image for the corresponding compression level. The “MSE” column to the right of “SNNS PSNR” column lists the MSE values for each image obtained from the trained neural network. We calculated percentage of improvement using the following equation:

$$(\text{Percentage}) \% \text{ increase} = \frac{\text{SNNS PSNR} - \text{original PSNR}}{\text{original PSNR}} \times 100$$

The “(percentage) % increase” column to the right “SNNS PSNR” column lists the improvement of PSNR obtained from the trained neural network over respective published paper’s results.

As shown in Table 1, the “SNNS PSNR” column values are always higher compared with the other PSNR columns. From, the “(percentage) % increase” column on the left side of “SNNS PSNR” column, we can see Reddhiran’s alpha blend filter’s PSNR improvement over the respective published paper’s PSNR. The “(percentage) % increase” column on the right most side of the table, shows a significant improvement of the neural network output image’s PSNR values over

corresponding originally published PSNR values. In some cases, for SNNs PSNR, it improved by 3.75% with the lowest improved by 0.69%. It is clear from table 1, that improvement does not increase PSNR much for low compressed images but works significantly better in highly compressed images. Overall, the proposed method always has a higher PSNR, and a lower MSE compared with any other method presented in table 1.

Example images with different compression levels are shown in Figure 18 with a high compression JPEG image. Figure 19 shows a low compression JPEG image:



(a)

(b)

Figure 18. (a) Lena with a high compression input JPEG image (0.150 bpp)

(b) Neural network output image



(a)

(b)

Figure 19. (a) Lena with a low compression input JPEG image (0.430 bpp)

(b) Neural network output image

Careful examination of the figures shows that reconstructed images from the neural network have more high frequency detail than the JPEG compressed images.

## **CHAPTER V**

### **CONCLUSION**

Image processing is a very important part in data compression which is used for storing information in devices and transmitting data over communication networks. There are two types of image compression- lossy and lossless but the most popular compression algorithm is JPEG, which is specified as lossy. We wanted to do research in reconstructing image pixels with their original values which were lost during image compression process.

JPEG is very popular in image compression, because it can produce compression ratios up to 10:1 with very little deterioration of image quality. JPEG uses a block based Discrete Cosine Transformation (DCT), dividing the image into 8x8 blocks and then transforming each block using the DCT.

Our goal in this research was to implement a neural network for post-processing of JPEG images. Reddhiyan developed an algorithm which produced considerable improvement in the quality of post-processed JPEG images, irrespective of level of compression present in the image. He developed the alpha blend algorithm because of low computational complexity, and ease of implementation. Alpha blend can be applied to any image with any level of compression.

In our algorithm, we took the alpha blend output as an input to a neural network and trained the neural network to compare these input image pixel values with the expected pixel values to increase quality. Once we had a trained neural network, it could be used to improve JPEG compressed images after processing by the alpha blend filter. We saw a significant improvement in PSNR and MSE values of images over other methods, after restoring them using a trained neural network, which has been presented in chapter 4.

In chapter 4, we saw that the PSNR values from different published papers had always been lower and that the MSE had always been higher than our SNNS trained neural network's PSNR and MSE respectively. The higher the PSNR ratio the better the quality of images after post-processing. The lower the MSE the better the quality of images. The SNNS trained neural network output image's PSNR, (Table 1, chapter 4) resulted in an improvement over the original published paper output image's PSNR as well as the Alpha blend output image's PSNR. The improvement was 3.75 % for the best, 0.69% for the lowest, with an average of 3.3%.

Further improvement can be expected through more refinements in the neural network. We used 7 to 8 JPEG images with different level of compressions from 5% to 95%. For improving results, training image file sets can be changed to include more compression levels, so that the neural network would learn more about pixel values of a compressed image for a certain level of compression.

For further improvement of the results, the number of hidden layer nodes can be changed. The number of hidden layer nodes while training the neural network can be a vital factor in the performance of the trained neural network. We wanted

to find a trained neural network which would work best in improving PSNR and MSE. Training a neural network with different numbers of hidden layer nodes is a very time-consuming task. For each test experiment, with a new number of hidden layer nodes, it would take more than 6-7 hours to finish. In some cases, it took almost a day. That's why we did not change the number of hidden layer nodes, more than 100 times in the test experiments. In our experiments, we changed the number of hidden layer nodes ranging from 11 to 196 (11,12,13,18,32,34,36,40,42,44,45,46,47,48,50, 70,72,74, 90, 92, 94,96, 98, 100, 104, 108, 144, 160 and 196). Then, we reconstructed the images each time from the new trained neural network and noted the PSNR and MSE values for each hidden layer nodes number change. Finally, we found that 96 hidden layer nodes produced the best-trained neural network in improving PSNR and MSE values of the images of all referenced compression levels of Table 1. For rest of the number of hidden layer nodes other than 96, the output of trained neural network images PSNR and MSE values did not improve for all referenced compression levels of images from Table 1. Images with high compression levels had some improvement in MSE and PSNR values for rest of the number of hidden layers other than 96. But for low compression levels, it did not improve at all for rest of the number of hidden layers other than 96.

For each experimental test, while setting the dimension of overlapping pixel blocks, we had to set the same dimension in two different places of the code. Once we did this while making training and test files and again in reconstructing an image from the trained neural network. Here, if we change the dimension of overlapping

blocks in trained and test file generation, then the dimension of overlapping blocks must be changed while reconstructing an image from the trained neural network.

In test 1, instead of  $12 \times 12$  input pixel blocks, we chose  $14 \times 14$  input pixel blocks and kept the output non overlapping pixel blocks unchanged to  $8 \times 8$  for generating neural network training files from the alpha blend output images and original (uncompressed) images. But we did not see any improvement in PSNR and MSE values after using the  $14 \times 14$  input pixel blocks. In test 2, we changed both the input and output pixel block dimensions. We used  $8 \times 8$  input pixel blocks, instead of  $12 \times 12$  input pixel blocks and  $3 \times 3$  non overlapping output pixel blocks instead of  $8 \times 8$  non overlapping output pixel blocks while generating the neural network training files. But in both cases, the resulting output images (from the trained neural network with input from the alpha blend algorithm) had more blocky artifacts. The output images also had lower PSNR and higher MSE values than both alpha blend and other published paper's output images.

In test 2, as JPEG works by dividing the original image into  $8 \times 8$  blocks, the  $3 \times 3$  output pixel blocks inside the  $8 \times 8$  input pixels block did not generate the entire block information for improvement because it was smaller than  $8 \times 8$ . The reason is, JPEG compression algorithm works by dividing the original image into  $8 \times 8$  non-overlapping blocks and encodes each block individually through several steps, which leads to block information loss if the output pixel block dimension is changed to any dimension other than  $8 \times 8$ .

Changing the numbers of epochs (iterations), while training the neural network, can change the result. In our experiment, we used 5000 epochs every time. If the



number of epochs is extended to 10000, it may give different results as the number of training iterations would be changed. But, while training the neural network, we need to make sure that over fitting is not happening as overfitting causes errors in predicting and generalizing new input data. In figure 13, we can see that the validation curve started to fall initially. Then due to over fitting the validation curve started to raise at the early stopping point. To stop over fitting, the training process should stop at the number of iterations where early stopping point is been indicated in the figure 13.

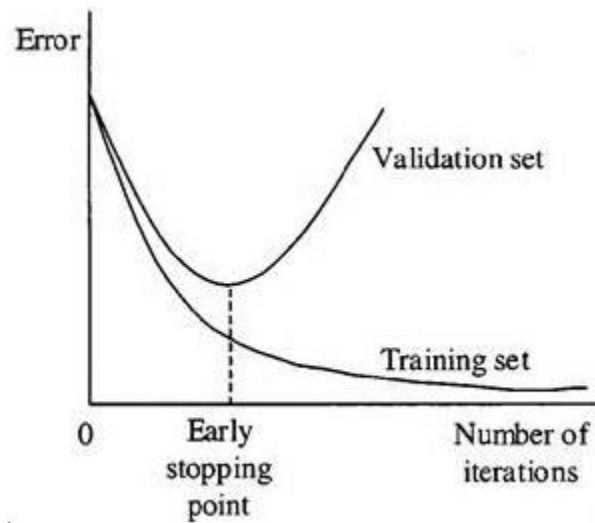


Figure 20: Graph with overfitting in validation curve

(source: elitedatascience.com)

As already mentioned before, each experiment took 6-7 hours in average using 5000 epochs. In the case of 10000 epochs, it took the double amount of time, but did not result in better PSNR and MSE values, as compared to 5000 epochs.

In our experiment, we used grayscale images. The objects in a color image reflect different combinations of light wavelengths from red, green and blue color channels. While modifying pixel values in JPEG compressed color images, we need to balance the combinations of red, green and blue color light intensity to a specific color. The color image uses 24 bits per pixel to represent the exact color of that pixel through encoding intensity levels ranging from 0 to 255 of red, green and blue light wavelengths for each color channel. For using color images in the experiment, we must do some changes in the alpha blend algorithm. The reason we have to make changes is because the bits-per-pixel value must be calculated separately. Color images have 24 bits per-pixel, 8 bits per each channel band of red, green and blue whereas grayscale images have only 8-bits per pixel. After this, train and test files need to be generated from the original RGB image and the alpha blend output RGB image to train a neural network. The test and train files will work on each color channel individually for each pixel in training the neural network. So, the trained neural network would be able to modify input RGB image's color pixels individually to expected values. Hence, the output of neural network should improve PSNR and MSE values of JPEG compressed color images.

Finally, our results showed a clear improvement of Lena images over any compression level ranging from 5% to 95%. The image samples with blocking and ringing artifacts were smoothed after post-processing with the trained neural

networks. For evaluating results, we used PSNR and MSE. Besides these, for further experiments, the Feature Similarity Index (FSIM) and Structural Similarity Index (SSIM) can be used to measure improvement in results.

## REFERENCES

1. An explanation of Dflate algorithm, <http://www.zlib.net/feldspar.html>
2. S. Chen, Z. He, and P.M. Grant, "Artificial neural network visual model for image quality enhancement," *Neurocomputing* 30, pp 339-346, 2000.
3. Du ñng T. Võ, Truong Q. Nguyen, Sehoon Yea, and Anthony Vetro, "Adaptive Fuzzy Filtering for Artifact Reduction in Compressed Images and Videos," *IEEE Transactions on Image Processing*, Vol. 18, No. 6, 2009, pp 1166-1178.
4. DEFLATE Compressed data format specification, <http://www.ietf.org/rfc/rfc1951.txt>
5. Yao Nie, Hao-Song Kong, Anthony Vetro, Huifang Sun, and Kenneth E. Barner, "Fast adaptive fuzzy postfiltering for coding artifacts removal in interlaced video," in *Proc. IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP) (2)*, 2005, pp 993-996.
6. Xiangchao Gan, Alan Wee-Chung Liew, and Hong Yan, "A smoothness constraint set based on local statistics of BDCT coefficients for image postprocessing," *Image and Vision Computing*, (23) 2005, pp 721-737.
7. Ronald Marsh, Riddhiman Goswami, "Alphablend - a self-adjusting algorithm for reducing artifacts in JPEG compressed images", 2016 IEEE International Conference on Electro Information Technology (EIT), DOI: 10.1109/EIT.2016.7535259
8. Y. Tsaig, M. Elad, G.H. Golub, and P. Milanfar, "Optimal framework for low bit-rate block coders", in *Proc. International Conference on Image Processing (ICIP) (2)*, 2003, pp.219-222.
9. Kieu, V.T. and Nguyen, D.T. "Surface fitting approach for reducing blocking artifacts in low bitrate DCT decoded images," in *Proc. International Conference on Image Processing (ICIP) (3)*, 2001, pp150-153.
10. Francois Alter, Sylvain Durand, and Jacques Froment, "Deblocking DCT-based compressed images with weighted total variation", in *Proc. IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP) (3)* 2004, pp 221-224.
11. Zou, Ju Jia, "Reducing artifacts in BDCT-coded images by adaptive pixel-adjustment," in *Proc. 17th International conference on Pattern Recognition (ICPR) (1)*, 2004, pp 508-511
12. Abboud, "Deblocking in BDCT image and video coding using a simple and effective method," *Information technology Journal* 5 (3), 2006, pp 422-426. ISSN 1812-5638.
13. Tai-Chiu Hsung and Daniel Pak-Kong Lun, "Application of singularity detection for the deblocking of JPEG decoded images," *IEEE Transactions on Circuits and Systems-II: Analog and Digital Signal Processing*, Vol. 45, No. 5, 1998, pp 640-644.
14. Jim Chou, Matthew Crouse, and Kannan Ramchandran, "A simple algorithm for removing blocking artifacts in block-transform coded images," *IEEE Signal Processing Letters*, Vol. 5, No. 2, 1998 pp 33-35.

15. G.A. Triantafyllidis, M. Varnuska, D. Sampson, D. Tzovaras, and M.G. Strintzis, "An efficient algorithm for the enhancement of JPEG-coded images," *Computers & Graphics* 27, 2003, pp 529-534.
16. Chaminda Weerasinghe, Alan Wee-Chung Liew, and Hong Yan, "Artifact reduction in compressed images based on region homogeneity constraints using the projection onto convex sets algorithm," *IEEE Transactions on Circuits and Systems for Video Technology*, Vol. 12, No. 10, 2002 pp 891-897.
17. Yi-Ching Liaw, Winston Lo, and Jim Z.C. Lai, "Image restoration of compressed images using classified vector quantization," *Pattern Recognition* (35), 2002, pp 329-340.
18. Sukhwinder Singh, Vinod Kumar, and H.K Verma, "Reduction of blocking artifacts in JPEG compressed images," *Digital Signal Processing* 17, 2007, pp 225-243.
19. Bahadir K. Gunturk, Yucel Altunbasak, and Russell M. Mersereau, "Multiframe blockingartifact reduction for transform-coded video," *IEEE Transactions on Circuits and Systems for Video Technology*, Vol. 12, No. 4, 2002, pp 276-282.
20. Shizhong Liu and Alan C. Bovik, "Efficient DCT-domain blind measurement and reduction of blocking artifacts," *IEEE Transactions on Circuits and Systems for Video Technology*, Vol. 12, No. 12, 2002, pp 1139-1149.
21. Camelia Popa, Aural Vlaicu, Mihaela Gordon, and Bogdan Orza, "Fuzzy contrast enhancement for images in the compressed domain," in *Proc. International Multiconference on Computer Science and Information Technology*, 2007, ISSN 1896-7094, pp 161-170.
22. Jinshan Tang, Jeonghoon Kim, and Eli Peli, "Image enhancement in the JPEG domain for people with vision impairment," *IEEE Transactions on Biomedical Engineering*, Vol. 51, No. 11 2004, pp 2013-2023.
23. Xiangchao Gan, Alan Wee-Chung Liew, and Hong Yan, "Blocking artifact reduction in compressed images based on edge-adaptive quadrangle meshes," *Journal of Visual Communication and Image Representation*, 14, 2003, pp 492-507.
24. Zhou Wang, Bovik, and B.L. Evan, "Blind measurement of blocking artifacts in images," in *Proc. International Conference on Image Processing*, Vol. 3, 2000, pp 981-984.
25. I.O. Kirenko, R. Muijs, and L. Shao, "Coding artifact reduction using non-reference block grid visibility measure," in *Proc. IEEE International Conference on Multimedia and Expo*, 2006, ISBN 1-4244-0366-7, pp 469-472.
26. Ling Shao and I. Kirenko, "Coding artifact reduction based on local entropy analysis," *IEEE Transactions on Consumer Electronics*, Vol. 53, Issue 2, ISSN: 0098-3063, 2007, pp 691-696.
27. Y.L. Lee, H.C. Kim, and H.W. Park, "Blocking effect reduction of JPEG images by signal adaptive filtering," *IEEE Transactions on Image Processing*, Vol. 7, No. 2, 1998, pp 229-234.
28. R. Samaduni, A. Sundararajan, and A. Said, "Deringing and deblocking DCT compression artifacts with efficient shifted transforms," *Proc. IEEE Int. Conf. on Image Processing, (ICIP '04)*, Singapore, vol. 3, pp.1799-1802, Oct. 2004.

29. Amjed S. Al-Fahoum and Ali M. Reza, "Combined edge crispiness and statistical differencing for deblocking JPEG compressed images," *IEEE Transactions on Image Processing*, Vol. 10, No. 9, 2001, pp 1288-1298.
30. Kiryung Lee, Dong Sik Kim, and Taejeong Kim, "Regression-Based Prediction for Blocking Artifact Reduction in JPEG-Compressed Images," *IEEE Transactions on Image Processing*, Vol. 14, No. 1, 2005, pp 36-48.
31. Jim Z.C. Lai, Yi-Ching Liaw, and Winston Lo, "Artifact reduction of JPEG coded images using mean-removed classified vector quantization," *Signal Processing*, 82, 2002, pp 1375- 1388.
32. Jagroop Singh, Sukhwinder Singh, Dilbag Singh, and Moin Uddin, "A signal adaptive filter for blocking effect reduction of JPEG compressed images," *International Journal of Electronic and Communications* 65, 2011, pp 827-839.
33. Hyuk Choi and Taejeong Kim, "Blocking-Artifact Reduction in Block-Coded Images Using Wavelet-Based Subband Decomposition," *IEEE Transactions on Circuits and Systems for Video Technology*, Vol. 10, No. 5, 2000, pp 801-805.
34. George A. Triantafyllidis, Dimitrios Tzovaras, and Michael Gerassimos Strintzis, "Blocking Artifact Detection and Reduction in Compressed Data," *IEEE Transactions on Circuits and Systems for Video Technology*, Vol. 12, No. 10, 2002, pp 877-890.
35. Zhen Li and Edward J. Delp, "Block Artifact Reduction Using a Transform-Domain Markov Random Field Model," *IEEE Transactions on Circuits and Systems for Video Technology*, Vol. 15, No. 12, 2005, pp 1583-1593.
36. LZ&& Compression algorithm, <http://msdn.microsoft.com/en-us/library/ee916854.aspx>
37. JPEG, <http://www.jpeg.org/jpeg/index.html>

