2018 ES 156 Project

Marguerite Basta
John A. Paulson School of Engineering and Applied Sciences
Harvard University
Email: mbasta@college.harvard.edu

***Abstract*—The field of image processing has rapidly evolved from the most simple of processing methods to now containing complex schemes like deep neural networks optimized for noise removal, compression, recognition and more. The goal of this paper is to analyze and recreate several of these image processing routines. Our main area of focus is the exploration of the various uses of both the 2-D Fourier Transform and the 2-D Discrete Cosine Transform. We lastly take a brief look at the increasingly prevalent use of neural networks in image processing.**

I. Introduction

Even the most complex image processing schemes are derived from several foundational elements of basic signal processing. In this project, our work is split into three main sections that recreate and analyze how these elements construct image processing routines. The first explores the use of the two-dimensional Fourier Transform for tasks like basic noise removal and image compression. We then proceed in the following section to explore and compare the two-dimensional Discrete Cosine Transform for similar, but more refined applications. In the final section of the paper, we look at the use of neural networks, which are designed to learn the same transforms and carry out these tasks.

The prevalence of certain image processing applications has led to the generation of significant amounts of advanced and related work. Various forms of the methods we implement can be found in mainstream packages from *numpy*, *scipy*, and *opencv*. In depth, comparative analysis of these methods can also be found in a variety of papers [1][2][3]. Specifically, the use of convolutional neural networks in image processing has in recent years generated significant amounts of relevant research [4][5][6][7].

We lastly note that this paper attempts to adhere to the most traditionally used notation for the relevant material. All specific definitions are still included within the main body of the paper with their notations outlined.

II. 2-D Fourier for Image Processing

We first examine the 2-D Fourier Transform as an image processing technique for the removal of noise and the compression of 512×512 black and white images.

A. Image Noise Removal with 2-D Fourier

The image processing routine in the noise removal portion of this section consists of two main parts.

The first part down-samples the original 512×512 image to 256×256 pixels and makes it noisy. This is done by first sampling every other pixel of the original image then adding a variable amount of Gaussian noise. The corresponding function is *noise_image()* from *part1_i.py*.

The second part attempts to reconstruct the full image by removing noise from the output of the prior part and rescaling it to 512×512 pixels. The noise removal is done by checking if the gray-scale value of each pixel is within the same standard deviation as its neighbors. The value of any pixel that does not fit this criteria is then changed to the average of neighboring values. The image is then rescaled to 512×512 by doubling the denoised pixels. The corresponding function is *filter_image()* from *part1_i.py*. Examples of the output are displayed in Figure 1.

In order to analyze the performance of the denoiser, its reconstruction error is compared to reconstruction error of denoising methods from mainstream packages on the same test image. Figure 2 compares the performance of the denoiser with the opencv *fastNlMeansDenoising()* and the numpy *gaussian_filter()* for different levels of added Gaussian noise. We can see from the results that our denoiser is competitive with standard methods.

Definition 2.1: Reconstruction error for original image $x[m,n]$ reconstructed image $\tilde{x}_\alpha[m,n]$

$$E_\alpha = \frac{1}{M}\frac{1}{N}\sum_{m=0}^{M-1}\sum_{n=0}^{N-1}(\tilde{x}_\alpha[m,n] - x[m,n])^2$$

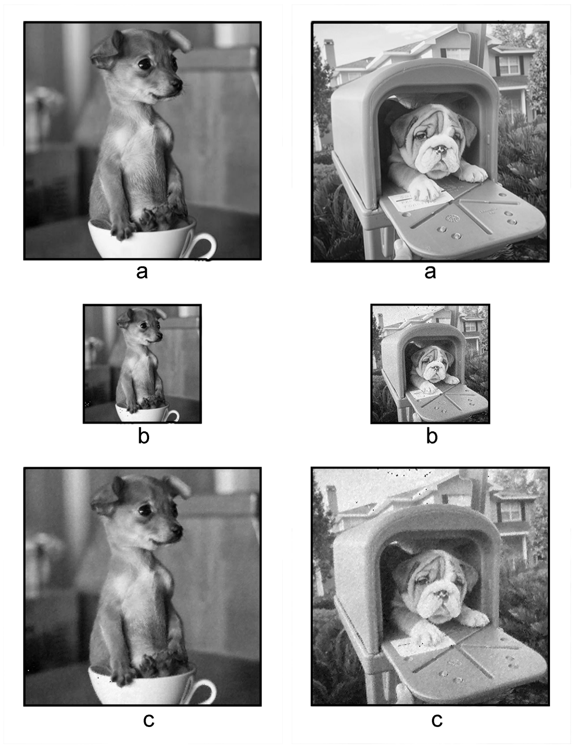


Fig. 1: Original image (a), down-sampled image with added noise (b), and reconstructed/filtered image (c) for added Gaussian noise of $\mathcal{N} \sim (0.0, 4.0)$ (left) and $\mathcal{N} \sim (0.0, 8.0)$ (right)

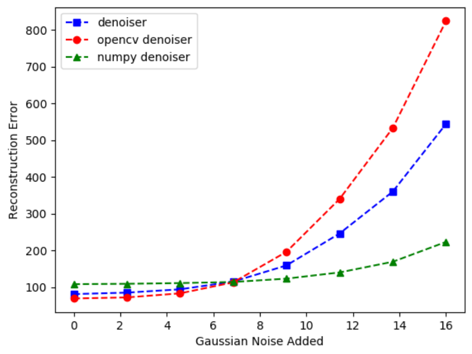


Fig. 2: Reconstruction error vs. added noise for our own denoiser (blue squares), the opencv *fastNlMeansDenoising* (red circles) and the numpy *gaussian_filter* (green triangles)

B. *Fourier Image Compressor*

As in the previous portion, the image processing routine in the compression aspect of this section consists of two main parts.

The first part computes the 2D Discrete Fourier Transform (DFT) of the inputed image in order to find its frequency domain. This transform is produced using the imported *cv2.dft()* function from opencv. Here, use of the opencv DFT is selected over the standard numpy FFT as its performance is generally more optimal over arrays with sizes that are powers of two. Depending on the specified α input value, the largest $\alpha\%$ of the transform coefficients are then maintained while the rest are set to zero.

The second part of the compression portion uses the inverse 2D Discrete Fourier Transform (IDFT) to produce an image from the subset of the previously computed DFT coefficients. The gray-scale values of the reconstructed image are taken to be the normalized magnitudes of the inverse transform. Examples of the output can be seen in Figure 3.

The compressor can then be analyzed by the trade-off between of the reconstruction quality and compression rate. For $0 < \alpha < 100$ and the reference images shown in Figure 1, the reconstruction error (1) is plotted as a function of α.

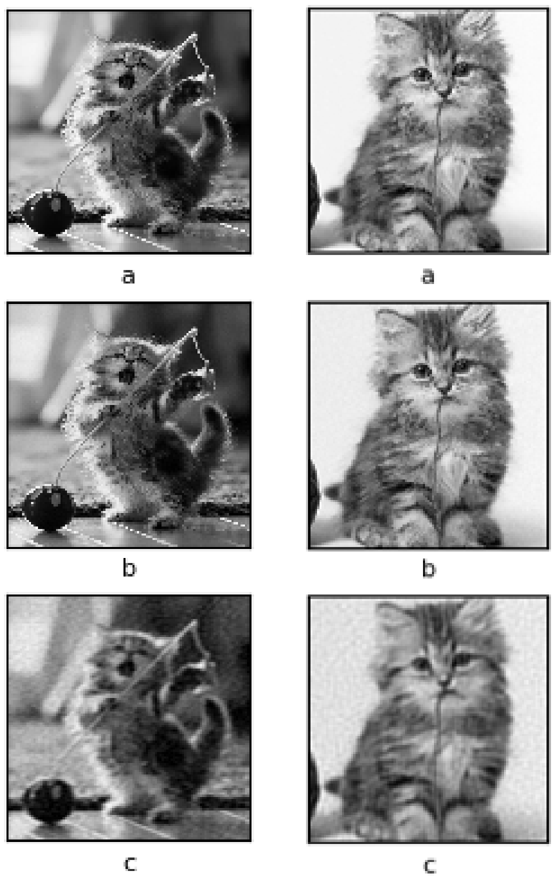


Fig. 3: Original images (a), reconstructed images with $\alpha = 5\%$ (b) and reconstructed images with $\alpha = .01\%$ (c)

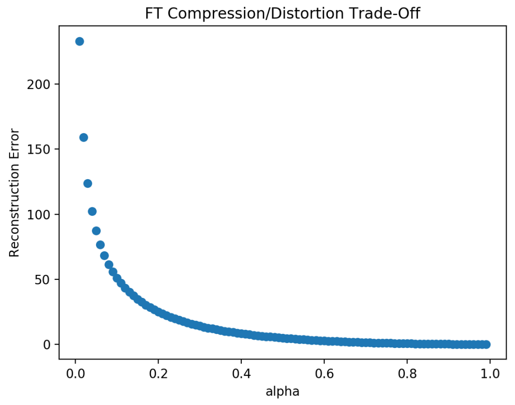


Fig. 4: Reconstruction error plotted as a function of α for the original sample image 1 (Figure 2 left)

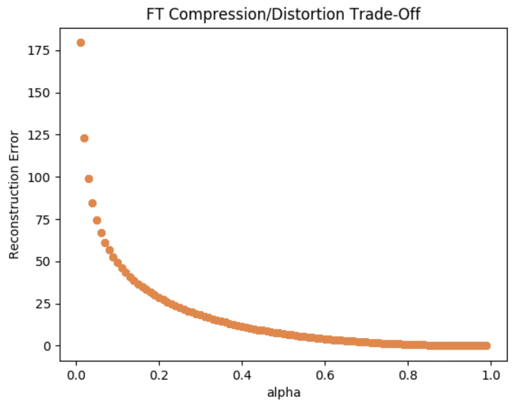


Fig. 5: Reconstruction error plotted as a function of α for the original sample image 1 (Figure 2 right)

III. Image Processing, DCT and JPEG

We follow our examination of the DFT with a look at the 2-D Discrete Cosine Transform (2-D DCT). We compare the performance of the two transforms in slightly more advanced versions of the previously seen image processing routines. In standard applications, the DCT is generally preferred for several significant reasons including its lower complexity, its oftentimes better approximation of signal behavior, and the fact that its frequency decomposition more closely follows that of the human visual system [2]. The main methods we will consider with the DCT considered are compression with partitioning, compression with partitioning and quantization, and reconstruction from both these schemes.

A. *Overview of the DCT*

For an initial overview of the DCT, we first outline its definition then show it to be an inner product in two dimensions.

Definition 3.1: The 2-D DCT: $x_C(k,l)$

$$x_C(k,l) := \frac{2}{\sqrt{MN}} \sum_{m=0}^{M-1} \sum_{n=0}^{N-1} x(m,n)\alpha_k C_{kl,MN}(m,n)$$

where $C_{kl,MN}(m,n)$ *is defined as:*

$$C_{kl,MN}(m,n) := \alpha_k cos[\tfrac{k\pi}{2M}(2m+1)]\alpha_l cos[\tfrac{l\pi}{2N}(2n+1)]$$

and α_k, α_l *are defined as:*

$$\alpha_k, \alpha_l := \begin{cases} \frac{1}{\sqrt{2}} & k,l = 0 \\ 1 & o.w \end{cases}$$

Proof 3.1: The 2-D DCT as the inner product $\langle x, C_{kl,MN} \rangle$

where A := the matrix of $C_{kl,MN}$

$$A = \begin{bmatrix} \alpha_0\alpha_0 cos[\frac{k\pi}{2M}]cos[\frac{l\pi}{2N}] & . \; . \; . & \alpha_0\alpha_{N-1} cos[\frac{k\pi}{2M}]cos[\frac{l\pi(2(N-1)+1)}{2N}] \\ . & & . \\ . & & . \\ . & & . \\ \alpha_{M-1}\alpha_0 cos[\frac{k\pi(2(M-1)+1)}{2M}]cos[\frac{l\pi}{2N}] & . \; . \; . & \alpha_{M-1}\alpha_{N-1} cos[\frac{k\pi(2(M-1)+1)}{2M}]cos[\frac{l\pi}{2N}]cos[\frac{l\pi(2(N-1)+1)}{2N}] \end{bmatrix}$$

and where B := the matrix of x

$$B = \begin{bmatrix} x(0,0) & . \; . \; . & x(N-1,0) \\ . & & . \\ . & & . \\ . & & . \\ x(M-1,0) & . \; . \; . & x(N-1,M-1) \end{bmatrix}$$

$$\langle A, B \rangle = \sum_{m=0}^{M-1} \sum_{n=0}^{N-1} a_{mn} b_{mn} \propto x_c(k,l)$$

B. Implementation of the 2-D DCT

The function *compute_dct()* from *part2.py* is then implemented following Definition 3.1. The function takes as input a two-dimensional signal, with variable dimensions but with the default form of a 35×35 pixel gray-scale image, and computes its two-dimensional DCT. For testing purposes, the output of the function is compared to the output of two nested one-dimensional DCT functions imported from *scipy.fftpack.* This follows from the fact that the 2-D DCT is able to be obtained from taking the one-dimensional transform along the columns of the 2-D signal and then along the rows of the result [1].

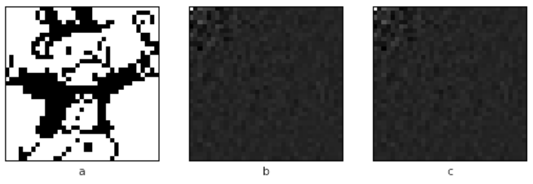
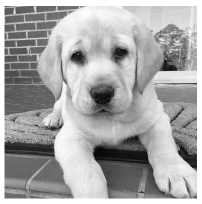


Fig. 6: The original image (a), the DCT of the image computed from *compute_dct()* (b) and the DCT of the image computed with *scipy.fftpack* functions (c)

C. Image Compression with the DFT and DCT

We next explore image compression with both the DFT and the DCT. In order to deal with the computational difficulty in 2-D of using DFT and DCT representations for high dimensional signals, the signal is partitioned. The compression is therefore done by splitting the signal into 8×8 patches, each of which maps to its K^2 largest coefficients. In order to deal with specified values of K^2 that are not perfect squares, the largest coefficients are stored with dimensions $k_1 \times k_2$. Here, normally $k_1, k_2 = \sqrt{K^2}$, but for non-perfect square values, k_1, k_2 are merely a pair of specified factors for K^2. The functions for the compression are *partitioned_dft()* and *partitioned_dct()* from *part2_ii.py.* A sample image with two sample patches and their corresponding transforms for varying values of K^2 can be seen in Figure 7.



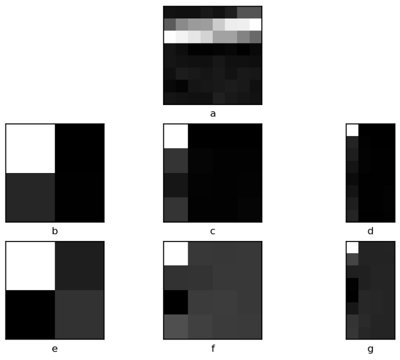


Fig. 7: The sample photo (top) with sample patch and transforms for the upper left corner patch (bottom). The corresponding DFT for K^2 = 4, 16, 32 is shown in images b, c, d. The corresponding DCT for K^2 = 4, 16, 32 is shown in images e, f, g.

D. Quantization:

We next explore a basic form of JPEG image compression using the previous partitioning method paired with quantization. After again computing and storing the DCT for each 8 × 8 patch of the image, the coefficient values are then rounded according to the quantization scheme defined below.

Definition 3.2: Basic quantization:

$\hat{X}_{ij}(k,l) = \text{round}\left[\frac{X_{ij}(k,l)}{Q(k,l)}\right]$

where $X_{ij}(k,l) = X_C(x_{ij})$, and x_{ij} is the $(i,j)_{th}$ block of the image.

and Q is the standard JPEG quantization matrix

$$Q_L = \begin{bmatrix} 16 & 11 & 10 & 16 & 24 & 40 & 51 & 61 \\ 12 & 12 & 14 & 19 & 26 & 58 & 60 & 55 \\ 14 & 13 & 16 & 24 & 40 & 57 & 69 & 56 \\ 14 & 17 & 22 & 29 & 51 & 87 & 80 & 62 \\ 18 & 22 & 37 & 56 & 68 & 109 & 103 & 77 \\ 24 & 36 & 55 & 64 & 81 & 104 & 113 & 92 \\ 49 & 64 & 78 & 87 & 103 & 121 & 120 & 101 \\ 72 & 92 & 95 & 98 & 112 & 100 & 103 & 99 \end{bmatrix}$$

This process is summarized by an overview of the standard JPEG process.

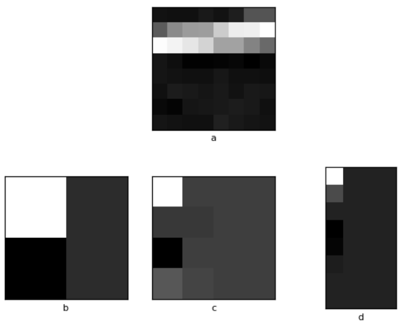


Fig. 8: The upper left corner patch (a) for the sample image in Figure 7 and the corresponding quantized DCT for K^2 = 4 (b), 16 (c), 32 (d).

E. Image Reconstruction with Partitioning and Quantization

Following the examination of the compression schemes in parts (C) and (D), we now explore reconstruction of the original image from these various schemes.

In order to reconstruct the image from the partitioned DCT compression in part (C), we first take the inverse DCT (iDCT) on all the patches. For each patch, we apply the imported function from *scipy*, *fftpack.idctn()*, which takes the inverse transform of a n-dimensional signal. The function which implements this patch-wise iDCT computation is *inverse_partitioned()* in *part2_iv.py*. The reconstructed patches are then stitched together to form the fully reconstructed $N \times N$ image. This process is implemented by the function *stitch()* in *part2_iv.py*.

For reconstruction from the quantization scheme in part (D), a similar procedure is followed. First however, the patches are "unquantized" by the procedure defined below.

Definition 3.3: Unquantization:

$X_{ij}(k,l) = \hat{X}_{ij}(k,l) \times Q(k,l)$

where $\hat{X}_{ij}(k,l)$ is the quantized transform and Q is the standard JPEG quantization matrix from Defintion 3.2

Following the computation of the un-quantized coefficients, the iDCT is then computed via the method as the prior reconstruction. The un-quantization is implemented by the function *inverse_quantized()* in *part2_iv.py*, and the image is again stitched together by *stitch.py()* from *part2_iv.py*.

The reconstruction error as a function of K^2 for both compression schemes is displayed in Figure 9. We also includes the PSNR as a function of K^2 in Figure 10. The PSNR is the peak signal-to-noise ratio (in decibels) between the original and compressed images. Here, higher values correspond to better compression quality. It is evident from the plots that quantization adds a negligible amount of error to the reconstruction when compared to the original partitioned DCT compression scheme. Likewise, the PSNR is only impacted for much higher values of K^2. This result demonstrates how there is essentially no quality loss as a trade-off for the computational efficiency of quantization, underscoring the advantages of using it in compression. Samples of these reconstructed outputs for several values of K^2 can be seen in Figure 11.

For purposes of comparison, we can look at the performances of our DCT compression schemes next to the same ones with the DFT. Figures 12 and 13 show the same previous reconstruction error and PSNR for the DCT next to those of the DFT with and without quantization. The results correspond with the traditional preference and superior performance of the DCT in image compression over the DFT.

Lastly, we compare our results to the JPEG standard. Figure 14 displays the reconstruction error of our DCT compression schemes next to the reconstruction error of several levels of JPEG compression quality on the same sample image. Notably, our scheme regularly outperforms the standard JPEG compression.

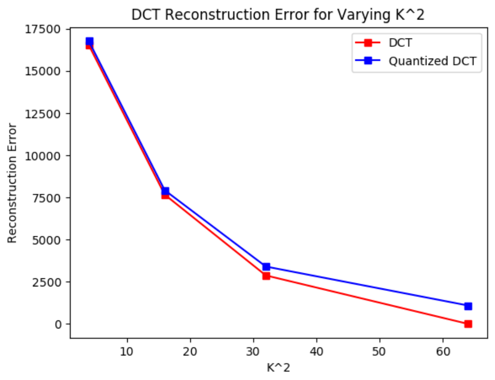


Fig. 9: The reconstruction error for the partitioned DCT and the quantized, partitioned DCT compression schemes vs. K^2.

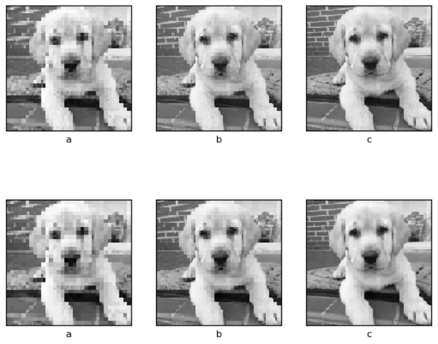


Fig. 11: The reconstructed images for the partitioned DCT (top) and the quantized, partitioned DCT compression schemes (bottom) for K^2 = 4 (a), 16 (b), 32 (c)

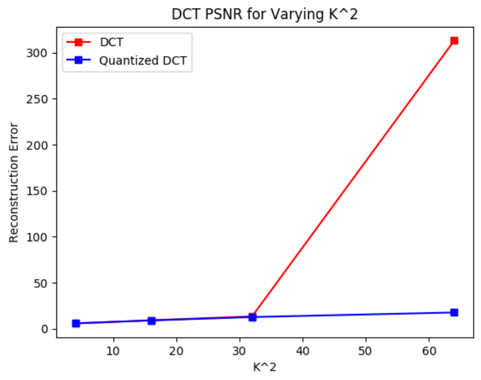


Fig. 10: The PSNR for the partitioned DCT and the quantized, partitioned DCT compression schemes vs. K^2.

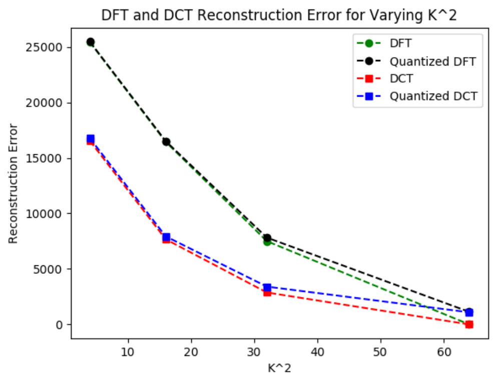


Fig. 12: The reconstruction error for the partitioned DCT, the quantized, partitioned DCT, the partitioned DFT and the quantized, partitioned DFT compression schemes.

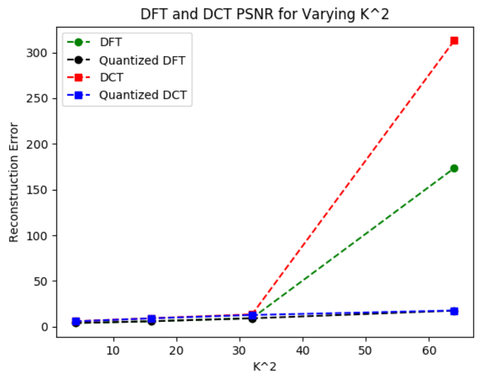


Fig. 13: The PSNR error for the partitioned DCT, the quantized, partitioned DCT, the partitioned DFT, and the quantized, partitioned DFT compression schemes.

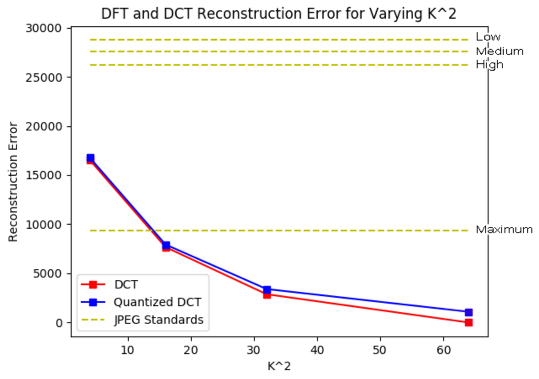


Fig. 14: The reconstruction error for the partitioned DCT compression scheme and the quantized, partitioned DCT compression scheme next to reconstruction error of several levels of JPEG standard compression schemes K^2.

IV. Neural Networks for Image Compression

In the final section of this paper, we take a look at relevant research on the use of neural networks for the previously explored image processing tasks. Compression schemes specifically have made significant advancements with these types of networks. The two main types of models we will examine are DCT optimization neural networks and pixel recurrent neural networks.

A. *Neural Networks for Optimized DCT Compression*

The increasing relevance of the DCT image compression methods, like the basic JPEG Quantization scheme discussed in Section II, has led to significant work in exploring techniques for optimization. Neural networks designed to maximize performance in various stages of the DCT compression are amongst these promising techniques.

One of the most prevalent designs for DCT compression networks focuses on determining the optimal compression ratio for a given image. As is evident from the previous sections, image compression quality at higher compression ratios is reduced. However, the behavior of the DCT makes this loss relativity marginal. Thus, there is a need for finding an optimum DCT compression ratio for a system which produces high quality compressed images while ensuring a relatively minimal time cost. One drawback of this approach has proven to be the costly training time; however, once trained, the time to recognize the optimal compression ratio upon presenting the image to the network has shown promising results [4].

Additional to this compression ratio optimization, other approaches have sought to implement back propagating neural networks (BPNNs) to reduce storage cost. After the initial decomposition, the two stage neural network is designed to reduce the representation space of the DCT coefficient to a space smaller than that of the actual coefficients. Similar to the patching scheme from Section II, following the partitioning of both the image and the decomposition, the networks are trained to represent the image blocks instead of the entire image. By saving the weights and bias of each neuron an image segment can be approximately recreated with the iDCT [5].

B. *Pixel Recurrent Neural Networks*

Departing from the standard DCT compression methods, recent compression schemes that have shown to be extremely promising are ones employing recurrent neural networks (RNNs). Instead of using the DCT transform the image, images are compressed and reconstructed with two neural networks - an encoder and a decoder. Using a Residual Gated Recurrent Unit (Residual GRU layers) with the encoder and decoder, the model iteratively refines the image reconstruction by passing added information between iterations. [6][7]. The RNN which is continually been successfully developed by Google research teams is available through Tensorflow. To take a look at its performance, we run it on the same sample image as our DCT compression schemes from Section II and compare the errors in *part3_I.py*. The error per iteration is shown in Figure 15. We also include the error for the standard measure for high quality JPEG compression on the image. Given the low number of iterations and speed of the model, the results are notable.

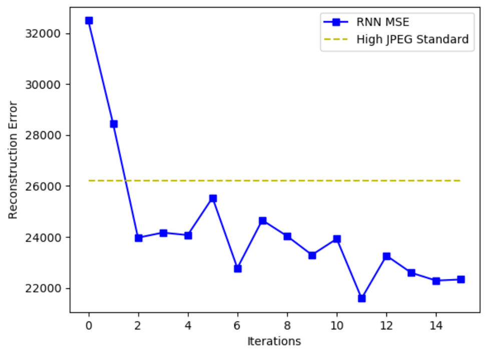


Fig. 15: Reconstruction Error for the images compressed by the RNN vs. iterations

V. Conclusion

The development and relevance of new methodologies for image processing routines has rapidly increased in recent years. In this paper we explored various applications and approaches to some of these routines like noise removal and compression. We started with the most basic methods that directly employed two-dimensional transforms like the DFT and the DCT. From there, we then refined our approach to take time and spacial constraints into consideration by adding incorporating image partitioning and quantization. Lastly, we explored some of the more recent methods in the field, which employ neural networks to optimize performance. In future work, we hope to direct our focus towards the refinement of these networks for an improved performance on more specific tasks.

References

[1] Chao-Yang Pang, Zheng-Wei Zhou and Guang-Can Guo, ”Quantum Discrete Cosine Transform for Image Compression,” *arXiv:quant-ph/0601043*.

[2] Robert M. Gray, IEEE, and David L. Neuhoff, IEEE, ”Quantization,” *IEEE Trans. on Information Theory*, vol. 44, no. 6, 1998.

[3] Anitha S, ”Image Compression Using Discrete Cosine Transform and Discrete Wavelet Transform,” *International Journal of Scientific and Engineering Research*, vol 2, no. 8, 2011.

[4] S.S. Tamboli, and V.R.Udupi, ”Optimum DCT Image Compression Using Neural Networks Arbitration,” *International Journal of Electronics, Communication and Instrumentation Engineering Research and Development (IJECIERD)* , vol 3, no. 1, 2013.

[5] Tarun Kumar, Ritesh Chauhan, ”An Adaptive Two-Stage BPNN-DCT Image Compression Technique,” *International Journal of Electronics Communication and Computer Engineering*, vol 5, no. 3, 2014.

[6] George Toderici, Sean M. OMalley, Sung Jin Hwang, Damien Vincent, David Minnen, Shumeet Baluja, Michele Covell and Rahul Sukthankar, ”Variable Rate Image Compression With Recurrent Neural Networks,” *arXiv:1511.06085v5*, 2016.

[7] George Toderici, Damien Vincent, Nick Johnston, Sung Jin Hwang, Joel Shor Michele Covell, ”Full Resolution Image Compression with Recurrent Neural Networks,” *arXiv:1608.05148v2g*, 2017.