Modeling Image Quantization Tradeoffs for Optimal Compression

Johnathan Chiu

University of California, Berkeley

November 2021

Abstract

All Lossy compression algorithms employ similar compression schemes – frequency domain transform followed by quantization and lossless encoding schemes. They target tradeoffs by quantizating high frequency data to increase compression rates which come at the cost of higher image distortion. We propose a new method of optimizing quantization tables using Deep Learning and a minimax loss function that more accurately measures the tradeoffs between rate and distortion parameters (RD) than previous methods. We design a convolutional neural network (CNN) that learns a mapping between image blocks and quantization tables in an unsupervised manner. By processing images across all channels at once, we can achieve stronger performance by also measuring tradeoffs in information loss between different channels. We initially target optimization on JPEG images but feel that this can be expanded to any lossy compressor.

1 Introduction

In this paper, we attempt to produce optimal quantization tables for lossy compression algorithms, most notably, JPEG. JPEG is a well known standard that has been widely used for many decades. Additionally, experiments are far easier to run given the algorithm is far more simplistic and more widely implemented than newer codecs like JPEG2000, HEIF, AVIF. The concepts still remain the same for RD optimization across all algorithms.

We strive to achieve small file sizes while minimizing visual degradation. In this paper, we regard visual degradation by a number of parameters – image "blockiness", significant color distortion, reduction of image texture.

We can formulate our proposed problem as an optimization one such that we want to maximize visual quality while minimizing file size. Given that this is evidently a non-convex problem, we can rely on neural networks to learn how to produce quantization tables which are locally optimal but still achieve the results that we desire. Additionally, rather than learning to linearly scale the table, we want to focus on each individual quantization index and how it relates to reconstructing the image as a whole. To train the model, we design a loss function that enables unsupervised learning so that our training process becomes the optimization process for each input.

Finally, in this paper, we mainly reference Google's Guetzli [1], MozJPEG [2], and Kraken.io [3] as our baseline metric for optimizers.

2 JPEG Basics

The JPEG algorithm uses the Discrete Cosine Transform (DCT) followed by quantization to compress images [4]. The quantization portion of the compressor is what makes JPEG a lossy compressor. In order to best compress the image, JPEG employs the use of run-length encoding (RLE) followed by Huffman encoding. In specific, the run length encoder is a 0-based run length encoder meaning that the data is compressed in a sequence of tuples. Given the energy compaction properties of DCT, low frequency data and high frequency data are separated in the transformed matrix. Additionally, previous knowledge about human visual perception define that higher frequency details in images are less perceptible to the human eye. The JPEG algorithm exploits the preceding two properties to heavily quantize higher frequency data in an image.

3 Related Works

Research on RD optimization is already well characterized. Additionally, a number of state of the art algorithms were introduced many decades ago [5, 6, 7]. Many popular libraries such as MozJPEG still employ use of these specific works [8]. On the other hand, research in mathematical modeling of the Human Visual System (HVS) is still a large ongoing research topic [9, 10, 11, 12].

The researchers at Google developed an algorithm to encode JPEG images using a closed-loop optimizer that utilizes Butteraugli, their proprietary model of human vision [13]. The developers designed their model to make use of two major optimizations. The first involves making quantization tables coarser which decrease the magnitude of stored coefficients. The second involves directly replacing DCT coefficients in each block to optimize zero run-length encoding. Guetzli produces files even smaller than the typical JPEG encoding, but it is very slow in comparison to the normal JPEG algorithm and can even produce images of quite unacceptable quality, as we see on Page 3 below.

Likewise, MozJPEG targets the problem similar with Trellis Quantization and adaptive quantization. They optimize the global quantization tables using a Dynamic Programming Algorithm introduced by Crouse et. al [14].

The researchers in [15] developed a CNN that generates a map that highlights "semantically-salient" regions in a given image to encode them at a higher quality relative to the rest of the image. They claim the neural network improves compression with by using higher bit rates to encode regions more sensitive to distortion according to the human perspective and lower bit rates elsewhere.

The authors of DeepN-JPEG made successful attempts to classify the impact of frequency bands in an 8x8 frequency block for the decompression of the JPEG algorithm [16]. Zhao et. al introduce a loss function that proves to have better performace by combining the use of SSIM and l_1 loss [17]. Both of the aforementioned papers were large inspiration in our work for modeling our loss function that provides higher quality information for RD optimization.

4 Loss Function

4.1 Multi-scale Structural Similarity Index (MS-SSIM)

Our loss function uses MS-SSIM as a means to measure image distortion. MS-SSIM incorporates use of the regular SSIM metric and low-pass filters up to a scale of M. Each individual low-pass filtered image is considered in the final formula. SSIM is a stronger metric than the traditional Mean-Squared Error (MSE) metrics since SSIM targets smaller local changes rather than the entire image. MS-SSIM between two images, x, y, is represented by the following formula(s):

$$\text{MS-SSIM}(x, y) = \text{SSIM}(x^M, y^M)^{\gamma_M} \Pi_{i=1}^{M-1} \text{CS}(x^i, y^i)^{\gamma_i}$$

where

$$\text{SSIM}(x, y) = \frac{2\mu_x\mu_y + C_1}{\mu_x^2\mu_y^2 + C_1} \text{CS(x, y)}$$

and

$$\text{CS}(x, y) = \frac{2\sigma_{xy} + C_2}{\sigma_x^2 + \sigma_y^2 + C_2}$$

The following variables: σ and μ follow their traditional representation of standard deviation and mean. C_1 and C_2 are arbitrary constants, usually 0.01 and 0.03 respectively. γ is an arbitrary constant that weights the important of each channel.

In addition, MS-SSIM takes into account structural information using the standard deviation of blocks in local regions. Without going into significant technical information we invite interested readers to the original paper for more details [18].

4.2 Entropy Estimation

Computing entropy is a nondifferentiable function since it requires the probability distribution of pixel values. We estimate entropy similar to the work in introduced by Ratnakar et. al [5]. We make the slight modification to average the value across the samples at each coefficient index. In addition, we scale each coefficient index after averaging with a value denoted as α_i inspired by the work from Liu et. al [16]. Our entropy estimation, R, over the quantized coefficients, Q, is thus defined as:

$$R(Q) \approx \frac{1}{N} \sum_{i=0}^{63} \sum_{n=0}^{N-1} \alpha_i Q_n[i]$$

where N is the number of sample blocks taken from the image. Refer to section 5 for more information on how N is chosen.

4.3 Minimax Function

With the following information, we aim to minimize both our rate loss and distortion loss simultaneously. We incorporate the use of l_1 loss similar introduced in the work from Zhao et. al [17]. Thus, we can define our loss function with the following formula:

$$l(x, y) = -\beta \log(\mathcal{L}^{\text{MS-SSIM}}(x, y)) + (1 - \beta)\mathcal{L}^{l_1}(x, y) + \gamma R(Q)$$

β and γ are all hyperparameters that can be tuned. We additionally use a window size of 3 for our MS-SSIM function.

5 Neural Network

5.1 Network Input

Our input consists of some number of samples of blocks in an image. The number of samples is a hyperparameter which can affect the fidelity of the image. We take inspiration from Prakash et. al but feel that it is simpler to find regions of interest (ROI) through simple statistical methods – we sample more blocks with high variance. We also incorporate some randomness in sampling so we get a stronger spread across the image [15]. The reasoning for this stems from the idea that high variance blocks can usually contain the most significant details in the image. So, if the quality following compression on high detail blocks are great, we can expect the rest of the image to have relatively high quality as well.

5.2 Model Architecture

Our network incorporates the use of fully connected and convolutional layers. The fully connected layers are important in relating each frequency index to another in the zigzag-encoded data. The convolutional layers provide information across channels. We illustrate information on the architecture in diagram 1. We start by zigzag encoding the input for each $C \times 8 \times 8$ sample block, where C is the number of channels. We similarly denote the number of sample blocks by S. We then linearly embed the $S \times C \times 64$ zigzag encoded tensor into a $S \times C \times 256$ tensor. This is followed by a 1D convolution which simultaneously downsamples the tensors and reduces the samples to 1 with a resulting shape of $1 \times C \times 64$. From here we undo our zigzag encoding and retrieve our $1 \times C \times 8 \times 8$ block and pass this into another series of convolution layers which gives us the quantization tables with shape $1 \times Q \times 8 \times 8$ where Q is the number of quantization tables we desire.

5.3 Training

We use Deep Learning as a means to optimize over a singular image. Deep Learning works better than traditional optimization techniques. In particular, our model has the ability to learn across the three channels simultaneously and maximize rate by considering trade-offs in separate channels. The model is simply used

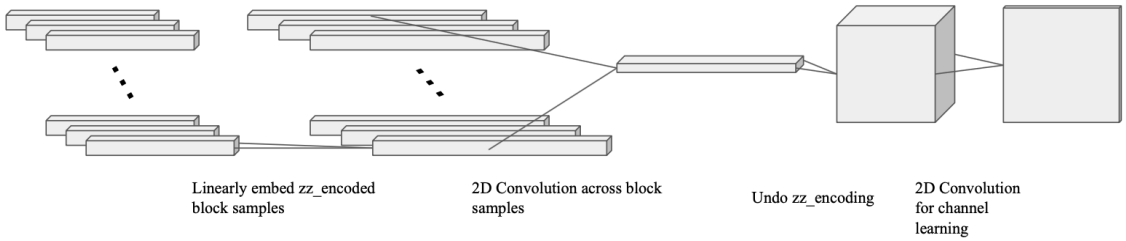

Figure 1: High level model architecture diagram

to optimize and not as a means to learn information for any large number of images. We will attempt to extend this to a universal model trained on larger datasets. We discuss this point further in section 7.

In our training process, each image is optimized over 100 epochs. In addition, we use the Adam Optimizer provided in the PyTorch library to solve our function.

Despite our loss function being a strong model, we have no exact measurement to truly compare the tradeoffs between visual quality and compression rate. To find the best possible image, we require a bit of human supervision. We create bins for each image to fall into. These bins are split by MS-SSIM and we take each image with the lowest rate loss across all the bins. We then allow the user to choose the smallest image file with quality they find acceptable.

5.4 Annealing

As mentioned in 4.3, β and γ are tunable hyperparameters. Rather than setting these constants, we look to use annealing as a means for enabling convergence. We penalize the β term dependent on the MS-SSIM factor – when our MS-SSIM is small (more distortion), we penalize β quite heavily. Otherwise, when we find that our model begins to converge to a sufficient MS-SSIM, we penalize β by $1 - \text{MS-SSIM(x, y)}$ multiplied by some "temperature". And, we penalize γ using the ratios of the entropy for the current solution over original solution also multiplied by the similar "temperature" value. The "temperature" is scaled each epoch in which the model proposes a solution that has an acceptable MS-SSIM value.

6 Results

We tested our model amongst a small set of 10 images with very differing qualities and saw really strong results throughout. We state the results of compression rates along with the respective resulting MS-SSIM below in figures 2 and 3.

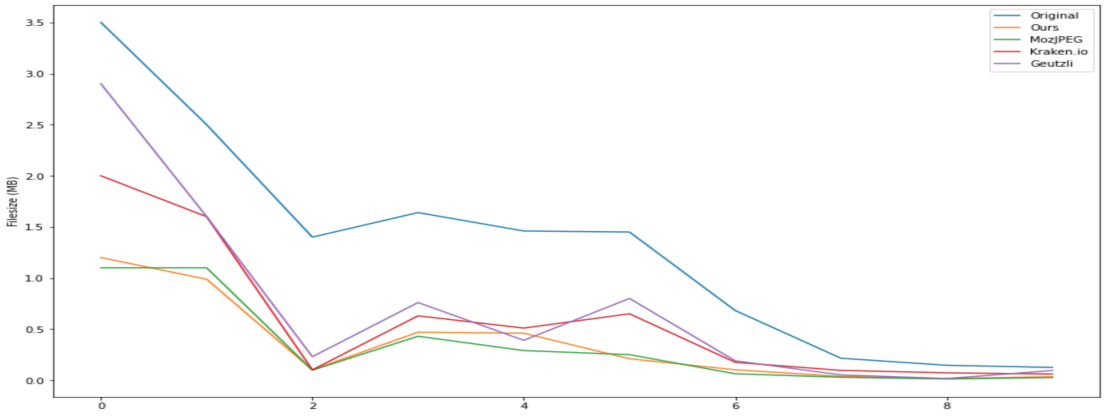

Figure 2: Filesizes across test images

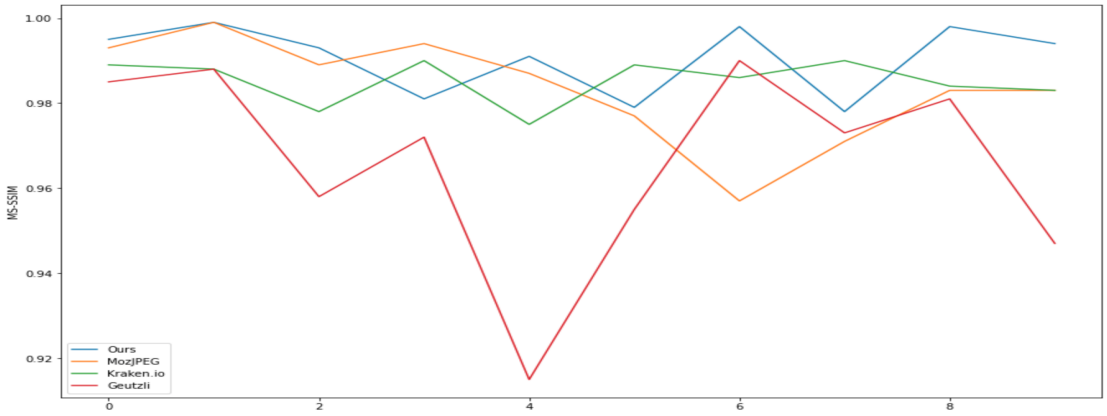

Figure 3: MS-SSIM across test images

Our test set differed in a number of ways: (1) file size, (2) file resolution, (3) subjects, (4) high contrasting colors, and (5) number of channels. We found that our compressor had stronger ability to reduce distortion than the other compressors. We did also find that our algorithm also performed on par with MozJPEG despite their implementations of Trellis Quantization and adaptive thresholding. We additionally include a small number of image examples in the Appendix for comparison.

7 Future Work

Another set of experiments that could show a great deal of promise would be to optimize a significantly larger model to learn and generalize outputs for any image input. This would enable the ability for these models to be implemented in a realworld setting.

The developers of MozJPEG used a number of adaptive thresholding techniques to produce smaller filesizes. We believe that implementing similar techniques will reduce file size exponentially and give us much stronger results. Given that our model alone gives really strong global quantization tables, we believe we can reapply the model on individual 8×8 blocks to give strong local quantization tables that will reduce filesize without compromising image quality.

In section 5.3, we note that we require human supervision to choose the most accurate output. We hope that future works can expand on the HVS modeling and enable a more complete picture of human perception. We also found that a number of machine learning tasks face similar issues with alignment to more human-like understanding of scenes. With this work, our models will only improve and learn more human-like vision qualities. We would thus like to explore our loss function applied to tasks that expand beyond compression and to other topics in the vision space.

8 Conclusion

In this paper, we utilized deep learning as a tool to compressing images according to the JPEG file format. We demonstrate the applicability of deep learning to image compression (and maybe other notable types of compression) and hope that future research extends on developing strong mathematical models for HVS. We believe there are a large number of implications that demonstrate how machine learning models could potentially learn to examine images in the same manner as humans.

It is important to note that our work can truly expand beyond JPEG given the similar inner-workings of all lossy image compressors. We have strong reason to believe similar models can be applied to any given compression algorithm to achieve better results. And, despite the recent advances in other separate image compressors such as HEIF and AVIF, the ubiquity of JPEG still stands.

Finally, though our test set is small, we hope that interested readers will test our work through our open source code repository. We hope this paper provides inspiration to readers to continue similar work and find novel and groundbreaking approaches to applying machine learning to data compression.

References

[1] J. Alakuijala, R. Obryk, O. Stoliarchuk, Z. Szabadka, L. Vandevenne, and J. Wassenberg. Guetzli: Perceptually guided jpeg encoder.

[2] Mozilla. Mozilla/mozjpeg: Improved jpeg encoder, . URL https://github.com/mozilla/mozjpeg.

[3] Nekkra UG. Image optimizer. URL https://kraken.io/.

[4] Ellen Chang, Udara Fernando, and Jane Hu. The discrete cosine transform. URL https://cs.stanford.edu/people/eroberts/courses/soco/projects/data-compression/lossy/jpeg/dct.htm.

[5] M. Livny and Viresh Ratnakar. Rd-opt: an efficient algorithm for optimizing dct quantization tables.

[6] K. Ramchandran and M. Vetterli. Rate-distortion optimal fast thresholding with complete jpeg/mpeg decoder compatibility. *IEEE Transactions on Image Processing*, 3(5):700–704, 1994. doi: 10.1109/83.334973.

[7] A. B. Watson. Dctune: A technique for visual optimization of dct quantization matrices for individual images. *Society for Information Display Digest of Technical Papers*, 24:946–949, 1993.

[8] Mozilla. Image specific quantization tables · issue 182 · mozilla/mozjpeg, . URL https://github.com/mozilla/mozjpeg/issues/182.

[9] Diana Sadykova and Alex Pappachen James. Quality assessment metrics for edge detection and edge-aware filtering: A tutorial review. In *2017 International Conference on Advances in Computing, Communications and Informatics (ICACCI)*, pages 2366–2369. IEEE, 2017.

[10] A Bhown, S Mukherjee, S Yang, and ... Humans are still the best lossy image compressors. *2019 Data â€¦*, 2019. URL https://ieeexplore.ieee.org/abstract/document/8712697/.

[11] Irena Hwang, Shubham Chandak, Kedar Tatwawadi, and Tsachy Weissman. Forget jpeg, how would a person compress a picture?, Oct 2021. URL https://spectrum.ieee.org/forget-jpeg-how-would-a-person-compress-a-picture.

[12] Nanyang Ye, María Pérez-Ortiz, and Rafał K. Mantiuk. Visibility metric for visually lossless image compression. In *2019 Picture Coding Symposium (PCS)*, pages 1–5, 2019. doi: 10.1109/PCS48520.2019.8954560.

[13] google. Projects. URL https://opensource.google/projects/butteraugli.

[14] M. Crouse and K. Ramchandran. Joint thresholding and quantizer selection for transform image coding: entropy-constrained analysis and applications to baseline jpeg. *IEEE Transactions on Image Processing*, 6(2):285–297, 1997. doi: 10.1109/83.551698.

[15] A. Prakash, N. Moran, S. Garber, A. DiLillo, and J. Storer. Semantic perceptual image compression using deep convolution networks.

[16] Zihao Liu, Tao Liu, Wujie Wen, Lei Jiang, Jie Xu, Yanzhi Wang, and Gang Quan. Deepn-jpeg: A deep neural network favorable jpeg-based image compression framework, 2018.

[17] Hang Zhao, Orazio Gallo, Iuri Frosio, and Jan Kautz. Loss functions for image restoration with neural networks. *IEEE Transactions on Computational Imaging*, 3(1):47–57, 2017. doi: 10.1109/TCI.2016.2644865.

[18] Z. Wang, E.P. Simoncelli, and A.C. Bovik. Multiscale structural similarity for image quality assessment. In *The Thrity-Seventh Asilomar Conference on Signals, Systems Computers, 2003*, volume 2, pages 1398–1402 Vol.2, 2003. doi: 10.1109/ACSSC.2003.1292216.

Appendix

Figure 4: Test image 9 (original filesize: 146 kB), left to right: MozJPEG (12 kB), Ours (17 kB), Kraken.io (72 kB), Guetzli (16 kB)

Figure 5: Test image 8 (original filesize: 215 kB), left to right: MozJPEG (29 kB), Ours (37 kB), Kraken.io (72 kB), Guetzli (52 kB)

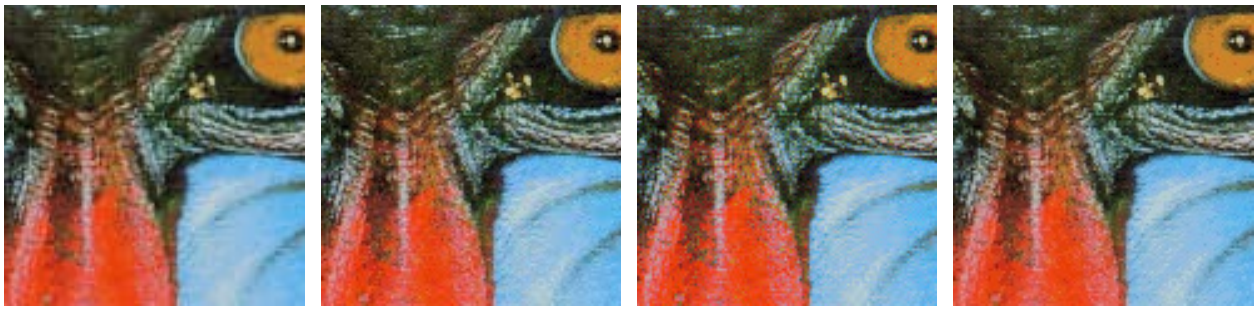

Figure 6: Test image 7 file (original filesize: 682 kB), left to right: MozJPEG (63 kB), Ours (102 kB), Kraken.io (175 kB), Guetzli (190 kB)

Figure 7: Not part of test set (original filesize: 3.7 MB), left to right: MozJPEG (1.1 MB), Ours (1.2 MB). The file dimensions were too large for Kraken.io nor Guetzli to generate