Computationally Efficient Neural Image Compression

Nick Johnston   Elad Eban   Ariel Gordon   Johannes Ballé
Google Research
nickj@google.com   elade@google.com   gariel@google.com   jballe@google.com

Abstract

Image compression using neural networks have reached or exceeded non-neural methods (such as JPEG, WebP, BPG). While these networks are state of the art in rate-distortion performance, computational feasibility of these models remains a challenge. We apply automatic network optimization techniques to reduce the computational complexity of a popular architecture used in neural image compression, analyze the decoder complexity in execution run-time and explore the trade-offs between two distortion metrics, rate-distortion performance and run-time performance to design and research more computationally efficient neural image compression. We find that our method decreases the decoder run-time requirements by over 50% for a state-of-the-art neural architecture.

1. Introduction

In recent years, tremendous progress has been made in the field of neural image compression [23, 22, 6, 20, 7, 16, 4, 15]. Each advancement produced another additional milestone in rate-distortion performance with respect to classic image compression codecs [24, 12, 10, 8] until they were first match or exceeded in Mean-Squared Error (MSE) or Multiscale Structural Similarity(MS-SSIM) [25] in [7].

Despite this success in the field of neural image compression, neural methods have not be widely deployed. A number of explanations can be found: lack of confidence in metrics, difficulty and lead time needed to ratify and adopt a new standard, portability of the technology or run-time requirements. With the exception of [20], run-time metrics have not been well documented in neural image compression literature, and there is virtually no prior work explaining in detail how the model architecture was arrived at. No research documents their method for choosing architecture. In this paper, we focus on this architecture search and how it relates to both rate–distortion and run-time.

We focus on the run-time performance of decoders in this work because of the asymmetric nature of compression. Images are generally compressed once and decompressed many times. Since an image could be viewed millions of times and on compute constrained devices, like smart phones, then high performance decoding is mandatory.

Due to the ubiquity and performance requirements around image decoders, we feel a study and understanding of where neural image compression performance currently is and how to improve the decoding performance is integral to mainstream acceptance and deployment of this rich field of research.

In this work, we:

1. Analyze the complexity of a common neural architecture's decoder.

2. Optimize Generalized Divisive Normalization (GDN, an activation function commonly used in neural image compression) [5] to reduce run-time at no performance loss.

3. Apply a regularization technique to optimize decoder architecture for Computationally Efficient Neural Image Compression (CENIC) models.

4. Analyze the trade offs of these learned architectures with respect to rate–distortion performance.

In Section 1.1 we discuss previous work related to neural image compression, performance and architecture search. We introduce the methods used in Section 2 including a baseline performance of the Mean-Scale Hyperprior architecture [16] (referred to as Mean-Scale), the GDN variant, and a description of the regularization technique focused on computation reduction. In Section 3 we describe the training procedure for our networks along with initial rate-distortion curves for the entire loss function sweep. In Section 4 we analyze and discuss the performance characteristics of the newly learned architectures and how they related to the rate-distortion performance. We found that it is easier to design high-performance compute-constrained models trained for MS-SSIM than for MSE. Finally, in Section 5 we conclude with recommendations on how to apply these methods to neural image compression networks along with offering additional insights that can be achieved through future work.

1.1. Previous Work

Image compression using neural networks has quickly become a rich field of research in recent years with [23], [22] and [6]. [15] has recently achieved state of the art in rate-distortion, which is based on the combined auto-regressive and hyperprior work introduced in [16].

While most research has focused on improving the rate-distortion curve on image compression performance, less research has been done on designing the end-to-end neural networks to be feasible in the applications space. [20] showed not only state of the art performance in MS-SSIM, but was also one of the few works to report neural compression and decompression runtimes, although the lack of details given on implementation and architecture make it difficult to understand what architectural details contribute to better runtime performance. [4] improved on the work of [7] and demonstrated the importance of deterministic computation needed for cross-device deployment, necessary for practical application of this research.

Neural architecture search [29, 19] and model compression and pruning techniques [27, 14, 26, 2, 17, 11] are playing an increasing role in the research of various computer vision and machine learning problems. However, to our knowledge, such techniques have not been utilized in the past for image compression models. We use an implementation of MorphNet [11] which has already shown to optimize classification network by balancing classification tasks with compute on much deeper architectures.

2. Mean-Scale Architecture

We are using the *Mean-Scale* Hyperprior network as described in [16], but without the context model. We omit the context model for two reasons, the first being simplicity in testing on a baseline network that is already better than BPG [8] and the second being the difficulty of implementing a computationally efficient autoregressive context model over latents. Some other work uses much deeper architectures. However, we choose this shallower network for analyzing run-time characteristics, because it is already more likely to be computationally more efficient than a deeper architecture.

Because the Group Lasso Regularization removes channels from the network architecture, we purposefully start out with an over-parameterized network with the knowledge that we have less computation in the overall learned final architectures. Throughout this paper we refer to the over-parameterized networks as *Larger Mean-Scale* and the learned final architectures as *Computationally Efficient Neural Image Compression (CENIC)* models. Our description of the Mean-Scale architecture (Figure 1) is described in Table 1 and the Larger Mean-Scale architecture in Table 2.
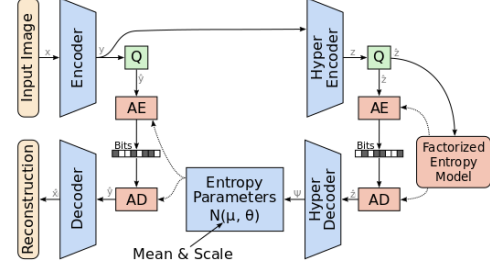

Figure 1. This is the Mean-Scale architecture we use as our baseline. Figure reproduced with permission from [16].

Encoder	Hyper Encoder	Hyper Decoder	Decoder
5x5conv,2,192	3x3conv,1,320	5x5deconv,2,320	5x5deconv,2,192
5x5conv,2,192	5x5conv,2,320	5x5deconv,2,320	5x5deconv,2,192
5x5conv,2,192	5x5conv,2,320	3x3deconv,1,320	5x5deconv,2,192
5x5conv,2,192			5x5deconv,1,3

Table 1. Description of the Mean-Scale encoder, hyper encoder, hyper decoder and decoder network architectures. We use the notation of CxC[conv or deconv],S,F to describe either a convolutional or deconvolutional layer with a kernel of CxC, a stride of S with F output channels.

Encoder	Hyper Encoder	Hyper Decoder	Decoder
5x5conv,2,320	3x3conv,1,640	5x5deconv,2,640	5x5deconv,2,320
5x5conv,2,320	5x5conv,2,640	5x5deconv,2,640	5x5deconv,2,320
5x5conv,2,320	5x5conv,2,320	3x3deconv,1,320	5x5deconv,2,320
5x5conv,2,320			5x5deconv,1,3

Table 2. Description of the Larger Mean-Scale encoder, hyper encoder, hyper decoder and decoder network architectures. The hyper encoder and decoder networks are identical for the mean and scale prediction. We use the notation of CxC[conv or deconv],S,F to describe either a convolutional or deconvolutional layer with a kernel of CxC, a stride of S with F output channels.

2.1. GDN activation function

Divisive normalization (DN) [9] is a multivariate nonlinearity designed to model the responses of sensory neurons in biological systems, and has been shown to play a role in efficient information processing. GDN was introduced in [5] as a probabilistic flow model, generalizing models such as independent subspace analysis and sparse coding. As an activation function, it is defined as:

$$z_i = \frac{x_i}{\left(\beta_i + \sum_j \gamma_{ij} |x_j|^{\alpha_{ij}}\right)^{\varepsilon_i}}, \tag{1}$$

where x_i and z_i denote the input and output vectors, respectively, α_{ij}, β_i, γ_{ij}, ε_i represent trainable parameters, and i, j represent channel indexes. It has been demonstrated that, compared to common pointwise activation functions such as relu or tanh, GDN yields substantially better performance at the same number of hidden units and comes with a negligible increase in number of model parameters when used inside an autoencoder-style image compression
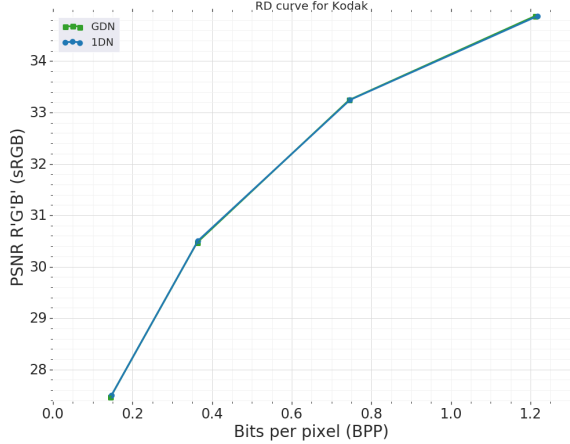
Figure 2. Rate-distortion plots on Kodak in PSNR for both GDN (with square/square root components) and 1DN (those without square/square root components). The lines overlap for the entirety of the rate-distortion curve, showing no compression performance degradation despite a significant computational simplification.

model [3]. It is also used in our Mean-Scale baseline. In their work, some parameters were fixed for simplicity ($\alpha_{ij} \equiv 2$ and $\varepsilon_i \equiv 0.5$). Given that computing a square root can take substantially more cycles than basic arithmetic, we re-examined this choice. We found that fixing $\alpha_{ij} \equiv \varepsilon_i \equiv 1$ doesn't hurt performance, while removing the need to compute square roots, and simplifying the function to basic arithmetic:

$$z_i = \frac{x_i}{\beta_i + \sum_j \gamma_{ij} |x_j|}. \tag{2}$$

Figure 2 shows a direct comparison of the rate–distortion performance for both GDN with square/square root exponents and our basic arithmetic version on the Kodak image set [13]. To make this comparison, we followed [3] in the choices for hyperparameters such as optimization method and network architecture.

3. Training with Group Lasso Regularization

For a given architecture, the number of filters in each layer is often an easy to ignore hyper parameter. The majority of CNN architecture either use a constant number of filters per layer, or a simple doubling scheme each time the spatial resolution is reduced. It is intuitive that these naive approaches do not necessarily provide the optimal structure for trading off task performance with computational cost[21]. We use a simple adaptation of the MorphNet approach[11] which uses a convex sparsity inducing regularizer in order to learn the number of filters in each layer. Some techniques use the batch-norm scaling factors to determine the relative activity of each channel. Since the Mean-Scale model doesn't use batch-norm, we use a

Hyper Decoder 1	Decoder 1	Hyper Decoder 2	Decoder 2
5x5deconv,2,76	5x5deconv,2,150	5x5deconv,2,10	5x5deconv,2,25
5x5deconv,2,107	5x5deconv,2,89	5x5deconv,2,10	5x5deconv,2,21
3x3deconv,1,320	5x5deconv,2,81	3x3deconv,1,320	5x5deconv,2,19
	5x5deconv,2,3		5x5deconv,2,3

Table 3. Two example decoder structures learned using MorphNet. Hyper Decoder 1 and Decoder 1 are paired examples for a MSE optimized network at 16% of the computation of the Larger Mean-Scale models while Hyper Decoder 2 and Decoder 2 are paired for a MS-SSIM optimized network at only 6.7% of the computation. We use the notation of CxC[conv or deconv],S,F to describe either a convolutional or deconvolutional layer with a kernel of CxC, a stride of S with F output channels.

weighted Group Lasso [28] regularization term. For a fully-connected layer with matrix W and n outputs this results in:

$$FLOP(W) = \sum_{j=1}^{n} \frac{1}{\sqrt{D}} \|W_j\|_2 \tag{3}$$

where W_j stands for all the weights associated with the j'th output. This applies analogously to convolutions.

In our experiments we use the FLOP-regularizer with a threshold of 0.001 to determine which output is active. We follow the MorphNet procedure, by first learning several structures by applying increasing regularization strength, and then after the structure convergence, we retrain the models given the learned structure.

Many of the architectures heavily constrained the mean prediction network in the hyper decoder. Manual tuning was required to ensure that the mean prediction network matched in tensor output shape to the scale prediction network, meaning that slightly more computation was manually added in to make valid architectures.

We do not store any checkpoints or reuse any weights between these two phases, the second phase of training is from scratch.

For the structure learning process, we use the code publicly available code from [18].

For training, we used the Larger Mean-Scale model architectures with a learning rate of 1e-4 using Adam as an optimizer with $\beta_1 = 0.9$, $\beta_2 = 0.999$ and $\epsilon = 1e-8$. In addition to training for the usual rate-distortion term, the first pass of the training was modified to include the FLOP-regularizer on weights in the decoder and hyper-decoder, represented by $FLOP(W)$. A sweep of weighting terms for alpha was done in parallel with the usual sweep on lambda terms.

$$L = D * \lambda + R + FLOP(W) * \alpha \tag{4}$$

The sweeps over λ depend on the distortion metric used: for MSE we used $\lambda \in \{0.1 \cdot 2^{t-6}\}_{t=0}^{7}$ and for MS-SSIM we used $\lambda \in \{2^{t-1}\}_{t=0}^{7}$. For both distortion functions, the α sweep was over $\alpha \in \{0.001 \cdot 0.2^t\}_{t=0}^{11}$.

As in [16] we do the λ sweeps over both MSE and MS-SSIM [25] as a distortion. MS-SSIM is reported in a logarithmic space to better visualize results in the ranges closer to loseless. These models were trained for roughly 100,000 steps in the first phase. The sweeps across λ and α result in 207 unique decoder architectures created in the first phase of training.

In the second phase of training, we reduce the convolutional filters in the decoder down to values determined by the FLOP-regularizer and train for over $1e6$ steps or to convergence. All of our models are trained using the TensorFlow [1] library.

4. Results

The training for FLOP optimization created 207 unique decode architectures. These optimized networks shared a few interesting traits that are worth noticing here.

None of the optimized networked shrank the scale prediction network in the hyper decoder. This could be explained by the fact that this part of the model only makes up for a fraction of the overall compute, hence little is gained by shrinking it.

The FLOP-regularized network tended to shrink most in the second and third decoder layers. This is certainly due to the fact that the image is close to the original size spatially and at this point is either one-half or one-quarter in both dimensions and a large amount of compute is needed due to the fully convolutional nature of the network and that all pixels must be processed.

A full listing of all architectures, along with the final loss function and λ values are available in the Supplementary Materials: https://storage.googleapis.com/compression-ml/cenic/Supplementary_Materials_Computationally_Efficient_Neural_Image_Compression.pdf.

4.1. Inference Performance

The run-time information was gathered on a single PC with a Intel Xeon W-2135 CPU @ 3.70GHz (no GPU or other accelerators are leveraged). A sweep of models described in Section 3 were ran on all images in the Kodak [13] dataset. We focus on decode times of the networks in our performance evaluation, this includes not only the neural network decoding but the range coder implementation to create an actual bitstream.

We compare the Mean-Scale and Larger Mean-Scale performances on Kodak in Figure 3. This is to show that that greatly increased capacity doesn't directly yield significantly different performance on the rate-distortion curve, but does significantly increase run-time and training time.

We also include a run-time–distortion graph for the Mean-Scale model in Figure 4. It's worth noticing that the variance for the MSE networks is lower than the MS-SSIM
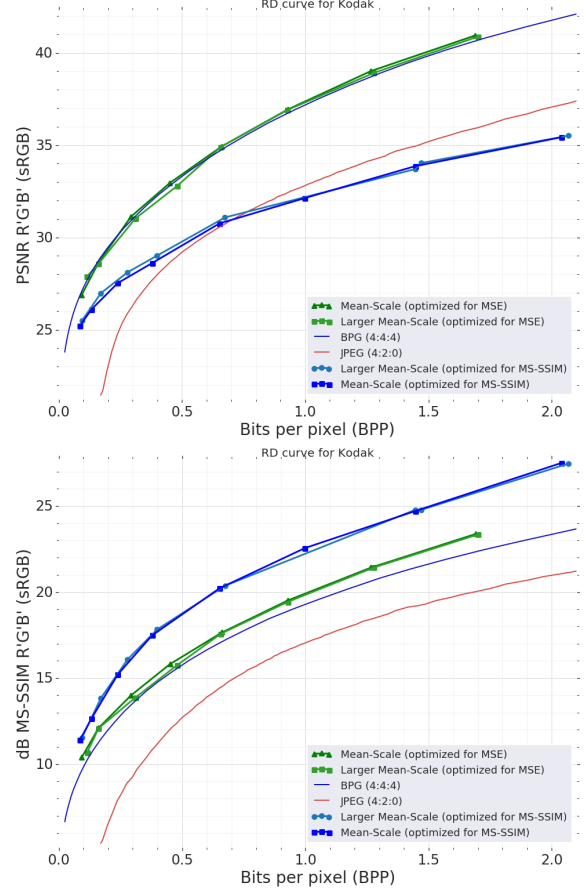

Figure 3. Rate-distortion plots on Kodak in PSNR (top) and dB MS-SSIM (bottom) for the Mean-Scale and Larger Mean-Scale networks. This is to show the Mean-Scale models are near that of what was presented in [16] and that blindly adding additional capacity to the Larger Mean-Scale models do not produce significantly improved results. dB MS-SSIM is calculated as $-10 * log_{10}(\text{MS-SSIM})$.

optimized networks. Another interesting point of analysis is that the run-time for the MS-SSIM network is almost 10% slower on the higher quality end. This is a concrete example of the loss function mattering, as it has a concrete effect on run-time despite all of the points in the plot having the exact same network structure.

The run-time difference is due to the complex interactions between reconstruction quality, how the images are encoded and the probability model, e.g. a model that produces highly probable codes and distributions with many unused symbols decode faster in a range coder implementation than one where the codes are evenly distributed. It is also worth noting that the majority of the time spent is within the neural network's hyper-decoder and decoder in reconstructing the intermediate and final images from the decoded bitstream.
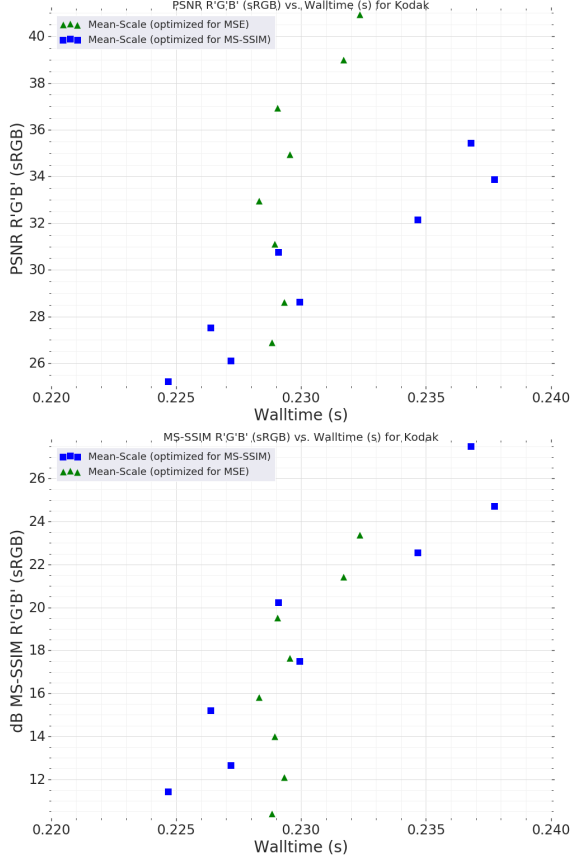
Figure 4. Run-time–distortion plots on Mean-Scale model for Kodak in PSNR (top) and dB MS-SSIM (bottom) on an Intel CPU. dB MS-SSIM is calculated as $-10 * log_{10}$(MS-SSIM).

	1st Layer	2nd Layer	3rd Layer
Square	0.327 ms (11.6%)	1.705 ms (12.9%)	6.843 ms (13.4%)
Sqrt	0.137 ms (4.9%)	0.998 ms (7.5%)	4.32 ms (8.5%)

Table 4. The absolute (and relative) time saved within the GDN activation after simplifying α_{ij} and ε_i in Equation 1. GDN is generally implemented with a the following TensorFlow ops: Square, Conv, BiasAdd, Sqrt, Mul. The Square and the Sqrt are simplified out in our faster formulation.

4.2. Performance on GDN activations

As shown in [3] and [4], GDN is a powerful activation function for neural image compression networks. Tensor-Flow operation level profiling shows the amount of time spent in average on each of the three GDN activations in the decoder layer. The change to the parameters α_{ij} and ε_i removes a square and a square root operation from the model. Table 4 shows the relative and absolute savings within GDN for this simplification. Overall, this results in a 14.33 ms reduction (21.4% savings) within the GDN operation and a 2.7% overall speed-up in the Mean-Scale models.
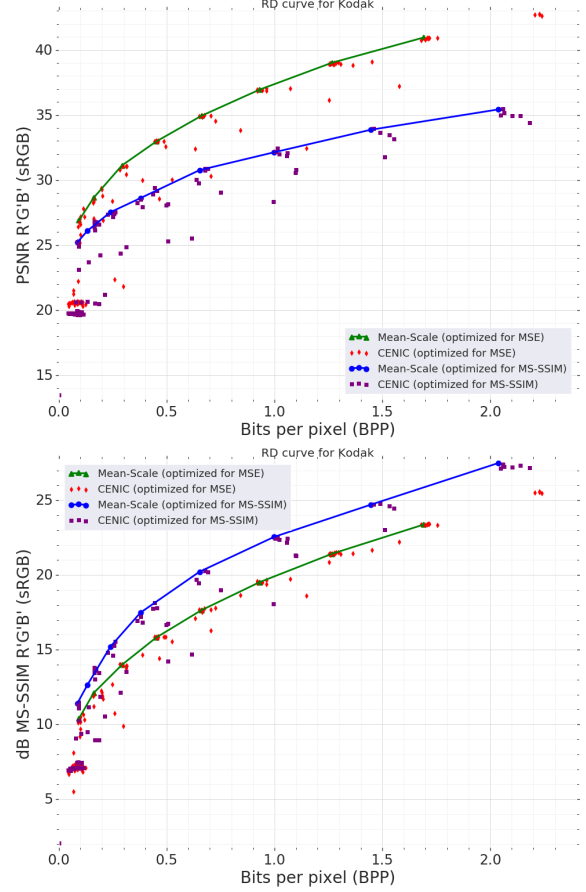

Figure 5. All learned architectures plotted with the Mean-Scale results with respect to MSE (top) and MS-SSIM (bottom). Due to the fact that we're doing an aggressive FLOP-regularized sweep over each rate point, we expect many low performance points to exist. dB MS-SSIM is calculated as $-10 * log_{10}$(MS-SSIM).

4.3. Performance on CENIC architectures

For the CENIC models, we graph the distortions compared to the Mean-Scale models with respect to average decoding time on Kodak. We draw a line connecting the various points for the Mean-Scale models so one can more easily see the frontier of faster models (on the left) vs. slower models (on the right).

A number of patterns appear in these run-time–distortion plots. The first is that for MSE optimized networks, many of them appear to the right of the vertical baseline (around 0.23 seconds). This is because we use the Larger Mean-Scale baseline (with more capacity) to begin the regularization process and many of these networks were not able to be decreased back to the baseline model. However, more of the MS-SSIM optimized models appear left of the vertical baseline, indicating that it is easier to optimize for FLOP performance when also training for this loss function.

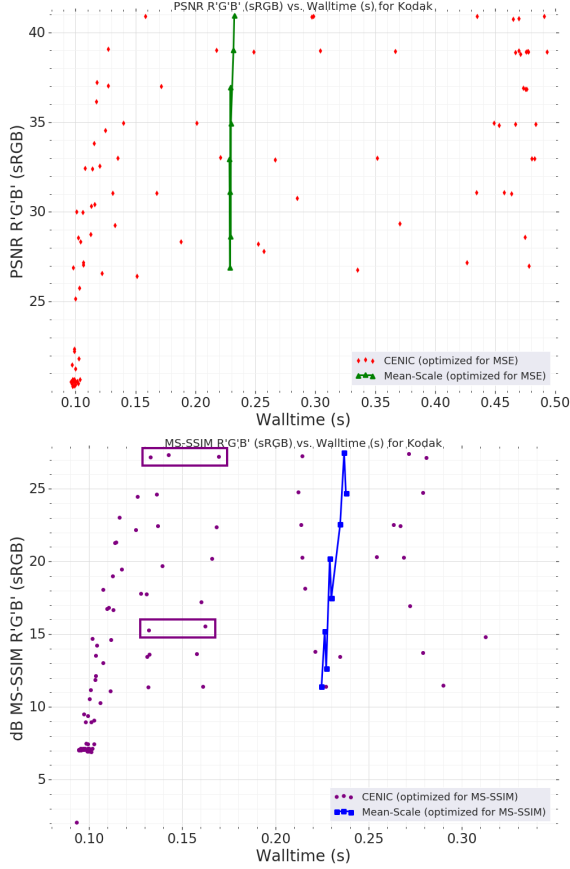The second is that there are more points clustered in the

Figure 6. All learned architectures plotted with the Mean-Scale results models optimized on MSE (top) and dB MS-SSIM (bottom) vs. average decode time. dB MS-SSIM is calculated as $-10 * log_{10}(\text{MS-SSIM})$.

lower-left corner of the graph for MS-SSIM vs MSE, meaning that low-quality (synonymous with low bitrate) models occurred for MS-SSIM vs MSE. These two graphs reinforce that taking into account the loss function and evaluation metric are key to developing fast, high-quality compression models.

Let's dive in on a few of the accelerated points we found in the MS-SSIM search. First the higher quality points (grouped in the top box of Figure 6b). These three models (architectures 5, 2 and 10 in the Supplementary Materials) have slightly decreased rate–distortion performance but is still much higher than the BPG baseline as shown in Figure 7 and their architectures can be compared in Table 5.

If we look at two models in the mid quality range of MS-SSIM (CENIC architectures 32 and 37), we find two models with substantial decoder speed-ups while performing very close to the Mean-Scale network shown in the bottom half of Figure 7 and their architectures are shown in Table 6. All architecture descriptions and runtimes are provided in the Supplementary Materials.

Hyper Decoder 5	Decoder 5	Hyper Decoder 2	Decoder 2
5x5deconv,2,76	5x5conv,2,79	5x5deconv,2,40	5x5deconv,2,149
5x5deconv,2,107	5x5conv,2,22	5x5deconv,2,67	5x5deconv,2,35
3x3deconv,1,320	5x5conv,2,43	3x3deconv,1,320	5x5deconv,2,39
	5x5conv,2,3		5x5deconv,2,3
Hyper Decoder 10	Decoder 10		
5x5deconv,2,66	5x5deconv,2,180		
5x5deconv,2,95	5x5deconv,2,58		
3x3deconv,1,320	5x5deconv2,73		
	5x5deconv2		

Table 5. The three high-quality MS-SSIM optimized CENIC models representing the scatter plot in Figure 7a. We use the notation of CxC[conv or deconv],S,F to describe either a convolutional or deconvolutional layer with a kernel of CxC, a stride of S with F output channels.
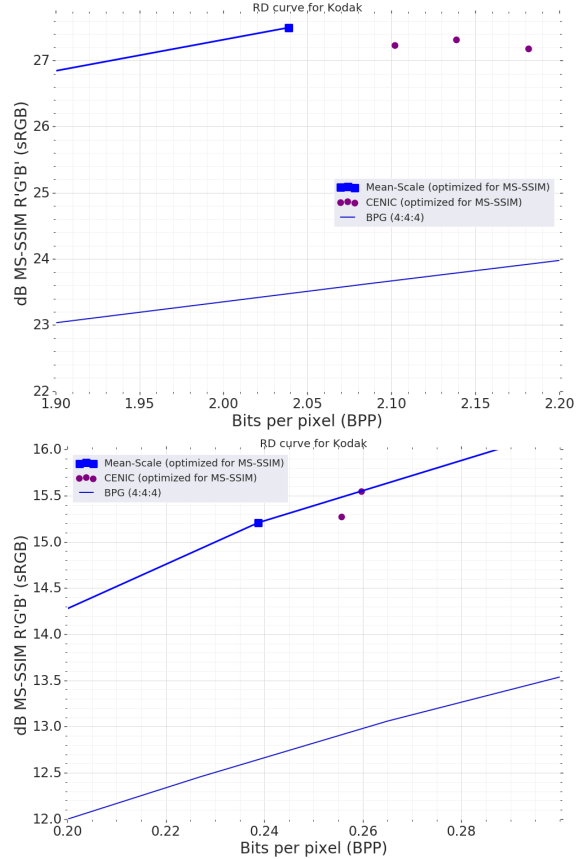

Figure 7. A closer look at CENIC models that operate at a similar distortion but much faster run-time than the Mean-Scale model. The higher quality models for comparison are in the top figure while the mid quality models are at the bottom. dB MS-SSIM is calculated as $-10 * log_{10}(\text{MS-SSIM})$.

4.4. Performance differences between MS-SSIM and PSNR

The clustering of models in Figure 5 shows that especially at the high bitrates of neural image compression, MS-SSIM models suffer less than MSE models. This can be seen by the much larger dips in performance above 1 bit-

Hyper Decoder 32	Decoder 32	Hyper Decoder 37	Decoder 37
5x5deconv,2,246	5x5conv,2,100	5x5deconv,2,110	5x5deconv,2,52
5x5deconv,2,170	5x5conv,2,126	5x5deconv,2,91	5x5deconv,2,99
3x3deconv,1,320	5x5conv,2,52	3x3deconv,1,320	5x5deconv,2,14
	5x5conv,2,3		5x5deconv,2,3

Table 6. The two mid-quality MS-SSIM optimized CENIC models representing the scatter plot in Figure 7b. We use the notation of CxC[conv or deconv],S,F to describe either a convolutional or deconvolutional layer with a kernel of CxC, a stride of S with F output channels.

per-pixel. Below 1 bit-per-pixel the MS-SSIM models lose substantially more performance and PSNR at similar capacities.

This observation is in line with anecdotal evidence of MS-SSIM being easier to train at higher bitrates and MSE being easier to train at lower bitrates–the loss functions and the capacity of the model play a key role of interacting together. It also explains the tendency of recent end-to-end learned compression research to report results in MS-SSIM, as it is easier to get high performance results on an unoptimized architecture.

4.5. Performance differences between CPU and GPU

Though we are looking at the decoding characteristics from a general purpose CPU perspective, it is worth understanding how Mean-Scale network's decoding time varies between CPU and GPU platforms. The GPU used for these numbers is a NVIDIA TitanXp on the a system with the same CPU as discussed above in Section 4.

From Figure 8, it is clear that, as expected, the Mean-Scale model runs much faster on GPU. With the average CPU run-time being around 230 milliseconds and the average GPU run-time being around 62 milliseconds (a 3.7x speed-up).

5. Conclusions

This work on analyzing the run-time and performance of neural image compression networks has led to a few of the following conclusions.

First, in the process of exploring FLOP-regularized architectures, we saw that simply adding more capacity to each layer led to no additional rate-distortion gains. This leads us to the conclusion that these four layer decoder and three layer hyper-decoder networks have already been well tuned for high rate-distortion performance, especially at the high rate/high quality end of the spectrum and this is done at significant costs to run-time and train time.

Second, we found that automating architecture search while constraining a network to a particular run-time and the use of methods like MorphNet that allows optimization directly on FLOPs is important because the hyperparameter
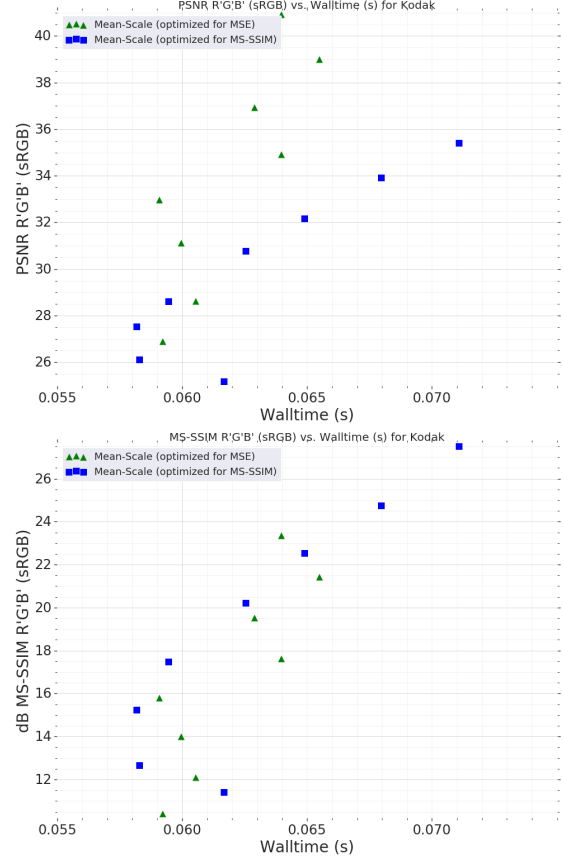

Figure 8. Run-time–distortion plots on Mean-Scale model for Kodak in PSNR (top) and dB MS-SSIM (bottom) on a NVIDIA TitanXp GPU. dB MS-SSIM is calculated as $-10 * log_{10}(\text{MS-SSIM})$.

search space is too large to effectively find good architectures manually.

The CPU and GPU comparisons of the models show that the compute platform and implementation matter. Running on a powerful GPU yielded almost a 3x improvement in run-time. Optimizing for MSE also produced lower variance decoding time across both platforms. The variance is due to the complex interactions between the neural network coding and probability distributions affecting encode and decode time of the range coder, since the amount of computation in the neural network components were fixed for these models.

After looking at the GDN activation functions, we also recommend the simplification to the α_{ij} and ε_i parameters to reduce decoding time while keeping rate–distortion performance nearly identical.

Lastly, MS-SSIM optimized models were able to be optimized for compute easier than their MSE optimized counterparts. This is encouraging because MS-SSIM generally correlates better with human perceptual quality than MSE [25]. Optimizing for decoder speed not only allows

for more practical, faster inference models, but also allows for faster neural network training due to having less parameters and fewer gradients.

6. Future Work

If working in real-time applications, it is clear that an arbitrarily slow encoder is not feasible. For example, when a photo is taken on mobile, it is generally expected to be viewable quickly, which means a reasonable encoding time is needed. Using MorphNet, both encoder and decoder networks can be optimized jointly to discover new architectures that balance real-time needs with compression performance.

Neural image compression networks that operate fully convolutionally also tend to produce large intermediate tensors, creating heavy stress on caches and internal memory systems. While we investigate the run-time components of these decoder networks, we did not look at the effects on the memory systems (though decreasing activations in these networks does strictly decrease the amount of memory required). These memory requirements may be more stringent in embedded applications or on special purpose accelerators where both memory and memory bandwidth are limited.

Finally, researching more perceptually-based image metrics that are differentiable (or approximately differentiable for the purposes of training neural networks) could find more optimal run-time–distortion trade offs.

References

[1] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org. 4

[2] Jose M Alvarez and Mathieu Salzmann. Learning the number of neurons in deep networks. In *Advances in Neural Information Processing Systems*, pages 2270–2278, 2016. 2

[3] Johannes Ballé. Efficient nonlinear transforms for lossy image compression. In *Picture Coding Symposium (PCS), 2018*, 2018. 3, 5

[4] Johannes Ballé, Nick Johnston, and David Minnen. Integer networks for data compression with latent-variable models. In *International Conference on Learning Representations*, 2019. 1, 2, 5

[5] Johannes Ballé, Valero Laparra, and Eero P. Simoncelli. Density modeling of images using a generalized normalization transformation. *arXiv e-prints*, 2016. Presented at the 4th Int. Conf. on Learning Representations. 1, 2

[6] Johannes Ballé, Valero Laparra, and Eero P. Simoncelli. End-to-end optimized image compression. *arXiv e-prints*, 2017. Presented at the 5th Int. Conf. on Learning Representations. 1, 2

[7] Johannes Ballé, David Minnen, Saurabh Singh, Sung Jin Hwang, and Nick Johnston. Variational image compression with a scale hyperprior. In *International Conference on Learning Representations*, 2018. 1, 2

[8] F. Bellard. BPG image format (http://bellard.org/bpg/). Accessed: 2017-01-30. 1, 2

[9] Matteo Carandini and David J. Heeger. Normalization as a canonical neural computation. *Nature Reviews Neuroscience*, 13, 2012. 2

[10] Google. WebP: Compression techniques (http://developers.google.com/speed/webp/docs/compression). Accessed: 2017-01-30. 1

[11] Ariel Gordon, Elad Eban, Ofir Nachum, Bo Chen, Hao Wu, Tien-Ju Yang, and Edward Choi. Morphnet: Fast & simple resource-constrained structure learning of deep networks. In *Computer Vision and Pattern Recognition*, 2017. 2, 3

[12] Information technology–JPEG 2000 image coding system. Standard, International Organization for Standardization, Geneva, CH, Dec. 2000. 1

[13] Eastman Kodak. Kodak lossless true color image suite (PhotoCD PCD0992). 3, 4

[14] Vadim Lebedev and Victor Lempitsky. Fast convnets using group-wise brain damage. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2554–2564, 2016. 2

[15] Haojie Liu, Tong Chen, Peiyao Guo, Qiu Shen, Xun Cao, Yao Wang, and Zhan Ma. Non-local attention optimized deep image compression, 2019. 1, 2

[16] David Minnen, Johannes Ballé, and George D Toderici. Joint autoregressive and hierarchical priors for learned image compression. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems 31*, pages 10771–10780. Curran Associates, Inc., 2018. 1, 2, 4

[17] Kenton Murray and David Chiang. Auto-sizing neural networks: With applications to n-gram language models. *arXiv preprint arXiv:1508.05051*, 2015. 2

[18] Andrew Poon, Max Moroz, Yair Movshovitz-Attias, and Elad Eban. MorphNet: Fast & Simple Resource-Constrained Structure Learning of Deep Networks. github.com/google-research/morph-net, 2019. 3

[19] Esteban Real, Alok Aggarwal, Yanping Huang, and Quoc V. Le. Regularized evolution for image classifier architecture search. In *The Thirty-Third AAAI Conference on Artificial Intelligence, AAAI 2019, The Thirty-First Innovative Applications of Artificial Intelligence Conference, IAAI 2019, The Ninth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2019, Honolulu, Hawaii, USA, January 27 - February 1, 2019*, pages 4780–4789, 2019. 2

[20] Oren Rippel and Lubomir Bourdev. Real-time adaptive image compression. In *International Conference on Machine Learning*, 2017. 1, 2

[21] Mingxing Tan and Quoc V. Le. Efficientnet: Rethinking model scaling for convolutional neural networks. 2019. 3

[22] Lucas Theis, Wenzhe Shi, Andrew Cunningham, and Ferenc Huszár. Lossy image compression with compressive autoencoders. In *Proc. of 5th Int. Conf. on Learning Representations*, 2017. 1, 2

[23] George Toderici, Sean M. O'Malley, Sung Jin Hwang, Damien Vincent, David Minnen, Shumeet Baluja, Michele Covell, and Rahul Sukthankar. Variable rate image compression with recurrent neural networks. In *International Conference on Learning Representations*, 2016. 1, 2

[24] Gregory K. Wallace. The jpeg still picture compression standard. *Communications of the ACM*, pages 30–44, 1991. 1

[25] Zhou Wang, Eero P Simoncelli, and Alan C Bovik. Multiscale structural similarity for image quality assessment. In *Signals, Systems and Computers, 2004. Conference Record of the Thirty-Seventh Asilomar Conference on*, volume 2, pages 1398–1402. IEEE, 2003. 1, 4, 7

[26] Wei Wen, Chunpeng Wu, Yandan Wang, Yiran Chen, and Hai Li. Learning structured sparsity in deep neural networks. In *Advances in Neural Information Processing Systems*, pages 2074–2082, 2016. 2

[27] Ming Yuan and Yi Lin. Model selection and estimation in regression with grouped variables. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 68(1):49–67, 2006. 2

[28] Ming Yuan and Yi Lin. Model selection and estimation in regression with grouped variables. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 68(1):49–67, 2006. 3

[29] Barret Zoph, Vijay Vasudevan, Jonathon Shlens, and Quoc V. Le. Learning transferable architectures for scalable image recognition. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2018. 2