[2020 Winter/Spring semester URP] KAIST URP Program

Individual Research Project Final Report

Research Participation Project

압축 오토인코더를 이용한 내용인지 및 과제인지 가변 비트레이트 이미지 압축

Suro Lee, Computer Science (20160830)

For Submission

Research Subject (Korean) : 압축 오토인코더를 이용한 내용인지 및 과제인지 가변 비트레이트 이미지 압축

Research Subject (English) : Content-Aware and Task-Aware Variable Rate Image Compression using Compressive Autoencoders

Research Period : January 2, 2020 ~ December 18, 2020

Advisory Professor : Dongsu Han (Signature)

Teaching Assistant : Youngmok Jung (Signature)

As a participant(s) of the KAIST URP program, I (We) have completed the above research and hereby submit the final report on the research.

December 28, 2020

Researcher : Suro Lee (Signature)

Contents

Abstract

This paper proposes a variable-rate deep image compression algorithm based on compressive autoencoders. To gain more flexibility over traditional image compression algorithms that rely on linear transformations, we use non-linear transformations that are end-to-end trainable using gradient-based methods. However, since the quantization function used to compress the real-valued code and the discrete probability model used for the entropy loss are both non-differentiable, we provide differentiable approximations to enable end-to-end optimization. Our model also supports optimization for various rate-distortion performances by using a parameterized loss function consisting of a reconstruction loss and an entropy loss.

Using the proposed compressive autoencoder, we design two more compression methods: content-aware compression and task-aware compression. These methods exploit redundancies in images not effectively captured by traditional image compression algorithms, resulting in enhanced compression performance. Content-aware compression captures content-specific redundancies (e.g. facial structures in a face-only dataset) and task-aware compression optimizes for a task-specific loss instead of a perception-oriented loss, which allows the compression network to learn compact representations that minimally degrades task performance when decoded. These new compression methods are expected to benefit many real-life applications such as high-quality video streaming, video surveillance, and Internet of Things.

We demonstrate that our compressive autoencoder model surpasses one of the most frequently used image compression algorithms, JPEG, by evaluating our compression performance with perceptual metrics such as PSNR and MS-SSIM. We also provide promising evidence that content-aware compression and task-aware compression can be used to further enhance deep image compression. For future work, we plan to apply loss-conditional training so that a single compression model can be used for various bitrates. We also plan to attempt multi-objective learning through a combination of many different loss functions to better optimize our compressive autoencoder.

1. Introduction

Multimedia transmission, such as downloading high quality images and videos, currently take up 82% of all Internet traffic [26]. As users and content providers require greater quantities and higher qualities of visual data, effective image compression is essential to ensure a seamless delivery.

Although traditional image compression algorithms provide reasonable compression ratios in general settings, they have three critical downsides. First, traditional image compression algorithms rely on fixed linear and invertible transforms to exploit the inter-pixel redundancies. However, these fixed linear transforms can only exploit a certain type of redundancy defined by the type of transform used, resulting in inflexibility. In addition, these transforms are generally applied on small image patches (e.g. 8x8 patches), making it impossible to capture larger scale redundancies. Second, traditional image compression algorithms cannot be further optimized for a certain content-type. For example, face-detection cameras only need to send images of faces over the network and surveillance cameras only need to send video frames of a certain scenery over the network. In such cases, content-specific side information can be used to better compress the images, which are not utilized in traditional image compression. Lastly, traditional image compression algorithms cannot be optimized for a certain task. For instance, there has recently been a large surge in smart devices, which integrate artificial intelligence into Internet of Things (IoT) [27]. A lot of these smart devices require images to be sent to a server, where the image is fed to a neural network that performs a specific task such as classification or segmentation. However, the objective of traditional image compression algorithms are to find a compact representation that least degrades human perception. By optimizing for maximum task performance instead of minimum perceptual distortion, we can save bandwidth wasted on enhancing human perception and improve task performance.

Thus, our work addresses each of these three issues using a deep image compression algorithm. First, we propose a autoencoder based neural network that can be optimized for various bitrates. By using non-linear transforms and compressing 128x128 patches of the image, we learn a much more expressive and flexible transform that better captures a wide range of spatial dependencies. Next, we design a content-aware compression network by training our network with a single-content dataset, allowing our network to further exploit content-specific redundancies. Lastly, we design a task-aware compression network by optimizing for a task-specific loss instead of a perceptual loss.

We show that our compression networks outperform a widely-used traditional compression algorithm, JPEG [2], by evaluating compression performance on various metrics. To evaluate the reconstruction quality of variable-rate compression and content-aware compression, we use two perceptual metrics known as peak-to-noise ratio (PSNR) and multi-scale structural similarity (MS-SSIM) [28]. To evaluate the compression ratio of all compression methods, we measure the average bits-per-pixel (bpp) of a set of compressed images. To evaluate the task performance of our task-aware compression network, we measure the test accuracy of the task network.

2. Research Background

2.1 Traditional Lossy Image Compression

In general, lossy compressors aim to remove three types of redundancies within a digital image: inter-pixel redundancy, which is the redundancy caused by statistical dependencies amongst pixels; psycho-visual redundancy, which is the redundancy caused by humans' insensitiveness to some image signals; and coding redundancy, which is the redundancy caused by inefficient coding methods [1]. Thus, lossy compression algorithms consist of three main parts that target each redundancy (Figure 1).

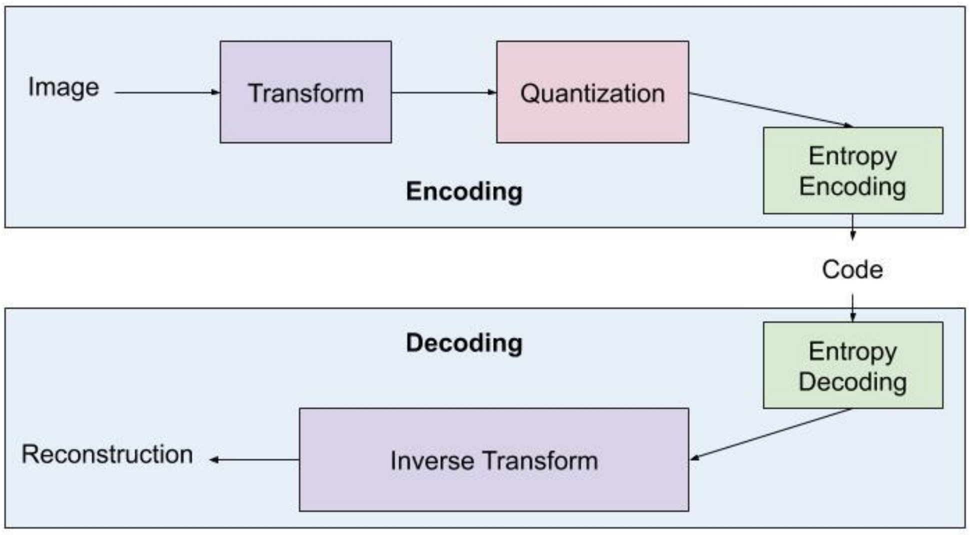

Figure 1: Lossy image compression

In a traditional image compression pipeline, a transform is first applied to the image to remove inter-pixel redundancy. This transform is generally a reversible and linear transform that enables better quantization by exploiting spatial dependencies. Then, quantization is performed on the transformed image to remove psycho-visual redundancies. Lastly, entropy coding is performed to reduce coding inefficiencies. Decoding is performed by first losslessly decoding the entropy-coded code and then applying the inverse transform of the linear transform used in the encoding. Due to quantization, some error is introduced in the encoding process, resulting in a lossy reconstruction.

In the case of JPEG [2], a discrete cosine transform is applied to 8x8 blocks of the image. Then, high frequency brightness variations are removed by dividing each component in the frequency domain by a constant and then rounding to the nearest integer. During entropy coding, a variant of Huffman coding [3] is used. For reconstruction, the entropy coding is undone and then the inverse discrete cosine transform is performed.

2.2 Deep Lossy Image Compression

Deep lossy image compression replaces the linear and invertible transform used in traditional image compression algorithms with a neural network. Both the forward transform and the inverse transform are usually replaced with either a recurrent neural network (RNN) [4,6] or a convolutional neural network (CNN) [5,7,8,9]. However, CNNs are recently gaining popularity over RNNs because RNNs tend to be slower than CNNs in both training and test time without significant gains in performance.

Similar to linear transforms used in traditional compression methods, CNNs also aim to reduce the redundancy caused by statistical dependencies amongst pixels. This is done by using convolution layers to continuously decrease the feature dimensions, eventually resulting in a feature vector that effectively represents the original image. This feature vector is expected to have learned a representation that has less statistical dependencies amongst its elements compared to the original image. The compressed code is then obtained by quantizing the feature vector and entropy coding the result. During decoding, the compressed code is first decoded using an entropy coder, and then the quantized feature vector goes through convolution layers and/or other spatial upsampling layers to reconstruct the original image. Increasingly, more and more deep lossy image compression algorithms are outperforming traditional lossy image compression algorithms.

2.3 Content-Awareness and Task-Awareness

Content-aware deep neural networks (DNNs) are different from traditional DNNs in that they are trained and tested on only a single content type. That is, the train and test sets are constrained to be very similar to one another. This method of training has been shown to be effective in video super-resolution [10], where a super-resolution DNN was trained for each type of video content (ex. a PC game, an animated film, etc.). Content-aware DNNs exploit the fact that DNNs tend to overfit the training data and result in low training loss. By constraining the train and test set to be similar, a low training loss in a content-aware DNN ensures a low testing loss as well. In other words, the content-aware DNN exploits content-specific characteristics in the training set that are also present in the test set, thus achieving superior performance to that of traditional DNNs.

Task-aware deep neural networks are typically encoder-decoder networks where the encoder network learns representations for a set of inputs. In terms of image compression, the task would be to learn a compact representation of the image that can be used to reconstruct the original image with minimal perceptual difference. However, we can train the encoder-decoder network to learn a representation optimized for other tasks as well. For instance, we can train the encoder-decoder network to learn a compact image representation to be sent to a classification network on the cloud, where it is decoded and used to classify the images. In this case, the task would be to classify the images using the classifier network with maximum accuracy. Task-aware representations can significantly save bandwidth when sending data over wireless networks for deep learning tasks without noticeable degradation in performance.

3. Approach

3.1 Compressive Autoencoder

Our compressive autoencoder is defined by four main functions:

$$E : \mathbb{R}^M \to \mathbb{R}^N,\ D : \mathbb{R}^N \to \mathbb{R}^M,\ Q : \mathbb{R}^N \to \mathbb{Z}^N,\ P : \mathbb{Z}^N \to [0, 1] \qquad (1)$$

where E is the encoder, D is the decoder, Q is the quantization function, and P is the discrete probability model. In this section, we will describe the deep learning model used for the encoder and the decoder, and other functions will be discussed in the following sections.

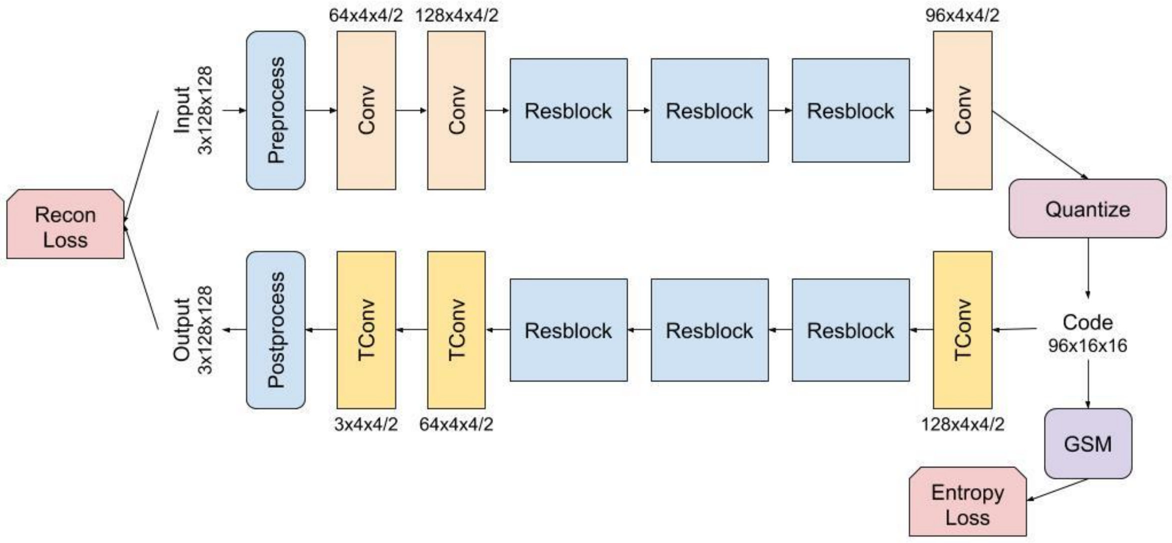

Figure 2: The compressive autoencoder. The CxKxK/S labels indicate convolutions and transposed convolutions with C output channels, KxK filters, and stride S.

As shown in Figure 2, the model begins by preprocessing the input image. In the preprocessing step, the original pixel values in the range [0, 255] are normalized to [-1,1]. Then, reflection padding of size 7 is added to all borders of the image so that the spatial dimensions of the resulting code is 16x16. After preprocessing, the spatial dimensions of the image are downsampled twice using 4x4 convolutions with a stride of 2. Then, the downsampled image goes through three residual convolution blocks that do not change the feature dimensions (Figure 3). The resulting tensor is downsampled once more using another strided convolution, giving us the feature vector. Each element in this vector is then quantized, resulting in the code vector. The reconstruction process mirrors the encoding process, except the downsampling convolutions are replaced with transposed convolutions [11] for upsampling. In the postprocessing step, the 142x142 reconstructed padded image is first clamped to the range $[-1, 1]$. Then, padding is removed from the reconstructed padded image and the pixel values are denormalized back to the range $[0, 255]$ to obtain the reconstructed image. For the clamping operation, we modify the gradient to be 1 outside the clamping range. This ensures that the error signal is properly propagated through the clamping operation. Note that we use a leaky ReLU after every convolution layer as a nonlinearity. Batch normalization [22] was not used in any convolution layers because it typically results in suboptimal compression performance [23].

Unlike Theis et al. [7], who also use a compressive autoencoder, we do not use sub-pixel convolution layers [12] to upsample the spatial dimensions during decoding. We found that sub-pixel convolution layers are not only relatively slower to train compared to transposed convolution layers, but also they frequently produce noisy artifacts in low frequency regions caused by the reshuffling of feature weights (Figure 4). Instead, we replace these upsampling layers with 4x4 transposed convolution layers with a stride of 2. We use a filter size that is a multiple of the stride to minimize checkerboard artifacts in the reconstructed image [25]. Naturally, we also use 4x4 convolution layers with a stride of 2 to downsample the images. This design choice results in faster convergence with significantly less compression artifacts and 20% less parameters compared to [7].

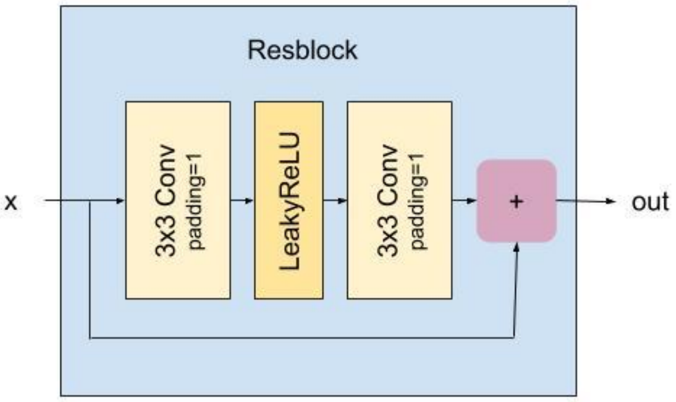

Figure 3: The Resblock layer used in the compressive autoencoder.

3.2 Variable Rate Image Compression

Variable bitrates in our compressive autoencoder are achieved by adjusting the rate-distortion tradeoff. The compressive autoencoder enables this by training jointly on a reconstruction loss and an entropy loss as shown in Equations 2-4. Here, entropy refers to the expected value of the information content as defined by [17].

$$\boldsymbol{y} = E(\boldsymbol{x}),\ \hat{\boldsymbol{y}} = Q(\boldsymbol{y}),\ \hat{\boldsymbol{x}} = D(\hat{\boldsymbol{y}}) \tag{2}$$

$$L_e = -\mathbb{E}[log_2 P(\hat{\boldsymbol{y}})],\ \ L_r = \mathbb{E}[(\boldsymbol{x} - \hat{\boldsymbol{x}})^2] \tag{3}$$

$$L = L_e + \beta L_r \tag{4}$$

During training, each compression model is trained on a different value of β to optimize for various rate-distortion tradeoffs. The entropy loss, L_e, is defined as the entropy of the quantized code vector, and the reconstruction loss, L_r, is defined as the mean-squared error between the original image and the reconstructed image.

However, the quantization function $Q(\cdot)$ and the discrete probability distribution $P(\cdot)$ are inherently non-differentiable, making back-propagation through these losses impossible. Therefore, we construct differentiable approximations for such non-differentiable functions as done other deep compression networks [4,5,6,7,13]. These approximations are described in the following sections.

3.3 Quantization

There have been many attempts to obtain a differentiable approximation of the quantization function, some of which include adding uniform noise [13], using binarization [4], and using the derivative of a smooth approximation [7]. After an empirical study, we use the derivative of a smooth approximation.

As a result, our quantization function is smoothed to $Q(y) = [y] \cong y$ giving the derivative:

$$\frac{d}{dy}Q(y) = \frac{d}{dy}[y] \ = \frac{d}{dy}y = 1 \tag{5}$$

where $[\cdot]$ is the rounding operator. Note that this approximation is only used for the derivative. That is, rounding is still performed during the forward pass.

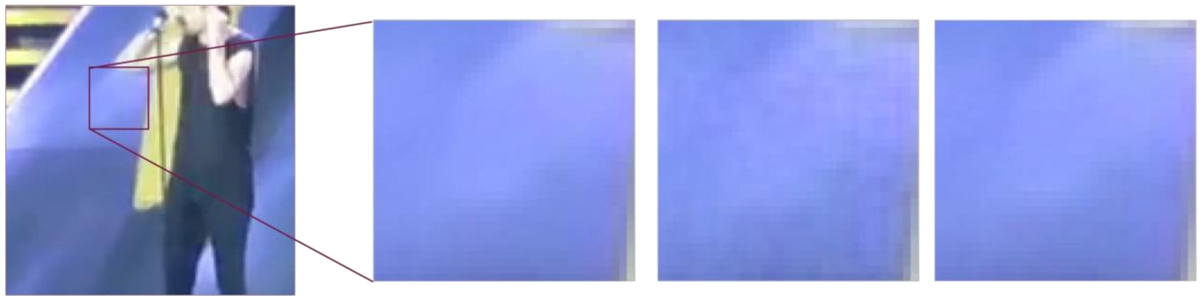

Figure 4: A close-up view of a low-frequency region of: the original image (left), the reconstructed image from Theis et. al's compressive autoencoder [7] (middle), and the reconstructed image from our compressive autoencoder (right).

3.4 Entropy Rate Estimation

As defined in Equation 1, $P(\cdot)$ is a discrete probability distribution. Therefore, taking the derivative of $P(\hat{\boldsymbol{y}})$ with respect to $\hat{\boldsymbol{y}}$ is undefined. Consequently, we define a differentiable probability model, $p(\cdot)$, to approximate $P(\cdot)$ in the reconstruction loss as follows:

$$-log\, P(\hat{\boldsymbol{y}}) = -log \int_{\hat{\boldsymbol{y}}-0.5}^{\hat{\boldsymbol{y}}+0.5} p(\hat{\boldsymbol{y}})d\hat{\boldsymbol{y}} \tag{6}$$

However, numerical instability often occurs during training in the tails of the differentiable probability model. To minimize such numerical instability, we minimize the upper bound of the reconstruction loss given by Jensen's inequality [14] as shown in Equation 7.

$$-log \int_{\hat{\boldsymbol{y}}-0.5}^{\hat{\boldsymbol{y}}+0.5} p(\hat{\boldsymbol{y}})d\hat{\boldsymbol{y}} \leq -\int_{\hat{\boldsymbol{y}}-0.5}^{\hat{\boldsymbol{y}}+0.5} log\, p(\hat{\boldsymbol{y}})d\hat{\boldsymbol{y}} \tag{7}$$

For the differentiable probability model, we use gaussian scale mixtures (GSMs), which are known to accurately characterize the probability distribution of natural image wavelet coefficients [15]. The approximation using GSMs is shown in Equation 8, where i and j are the spatial dimension indices and k is the channel index. Furthermore, s is the number of gaussian components in each GSM, which is set to 6. Note that we use a GSM for each channel of the code vector.

$$p(\hat{\boldsymbol{y}}) = \sum_{i,j,k}\sum_{s} \pi_s \mathcal{N}(\hat{y}_{ijk}; 0, \sigma_{ks}) \tag{8}$$

However, we found that numerical instability still occurs when integrating the log-likelihood of the GSM, thus we approximate the area underneath the distribution using a Riemann sum as follows:

$$-\int_{\hat{\boldsymbol{y}}-0.5}^{\hat{\boldsymbol{y}}+0.5} log\, p(\hat{\boldsymbol{y}})d\hat{\boldsymbol{y}} = -\int_{\hat{\boldsymbol{y}}-0.5}^{\hat{\boldsymbol{y}}+0.5} \sum_{i,j,k} log \sum_{s} \pi_s \mathcal{N}(\hat{y}_{ijk}; 0, \sigma_{ks})d\hat{\boldsymbol{y}} \tag{9}$$

$$= -\frac{[g(\hat{\boldsymbol{y}}-0.5) + 2g(\hat{\boldsymbol{y}}) + g(\hat{\boldsymbol{y}}+0.5)]}{4} \tag{10}$$

where

$$g(\hat{\boldsymbol{y}}) = \sum_{i,j,k} log \sum_{s} \pi_s \mathcal{N}(\hat{y}_{ijk}; 0, \sigma_{ks}) \tag{11}$$

The resulting entropy loss is then minimized using gradient-based methods. Minimizing the entropy loss minimizes the expected number of bits, which is also the absolute lower rate limit of a prefix-free code. The next section will describe the entropy coding method used to approach this entropy limit.

3.5 Entropy Coding

Once we obtain the code vector, this vector is further compressed using entropy coding. Entropy coding is class of lossless data compression algorithms that is independent of the specific characteristics of the medium. Generally, entropy coding algorithms use prefix-free codes to represent each fixed-length unique symbol, where the length of the prefix-free code is approximately proportional to the negative logarithm of the probability of the symbol. For our purposes, we use range coding [18] to further loselessly compress our code vector.

Following [7], we only use the GSM as the probability model during the training phase. During the testing phase, we use additively-smoothed histogram estimates of the coefficient distributions across the training set as the probability model. This is because the GSM often does not accurately match the probability distribution of the code vector coefficients, although it provides the error signal required to decrease the entropy.

3.6 Content-Aware and Task-Aware Image Compression

One of the downsides of traditional image compression algorithms is that they are neither content-aware or task-aware. By using content-aware and task-aware image compression algorithms, we can exploit other types of redundancies in an image that cannot be exploited using generic image compression algorithms, thus resulting in better rate-distortion performance.

By training our compressive autoencoder on a single-content dataset, we can expect our content-aware compression algorithm to better learn and remove commonly occurring inter-pixel redundancies that are specific to the content. Such content-awareness will result in superior performance over traditional algorithms for compressing images of the same type.

On the other hand, task-aware image compression algorithms do not need to optimize for human perception. Therefore, we replace the reconstruction loss in Equation 4 with a task-specific loss. By optimizing for a task-specific loss instead of a perceptual loss, the compressive autoencoder will learn a transformation that removes task-specific redundancies and extracts only the relevant information needed to successfully perform the given task. Since we also do not need to force our output to be in the range $[0,255]$, we skip the clamping and denormalization during postprocessing. Variable rates can also be supported in task-aware compression by optimizing for various values of the tradeoff coefficient, β. To maximize performance, the task-aware compression network is jointly optimized with the task network.

4. Experiments

4.1 Experimental Setup

We evaluate three main aspects of our work in detail: the rate-distortion performance of our compressive autoencoder, the rate-distortion performance improvement achieved by using content-aware image compression, and the rate-accuracy tradeoff achieved by jointly training our task-aware compressive autoencoder with a classification network. For each of the three evaluations, we also compare our performance with that of JPEG. All deep learning models and training/test scripts were implemented in Python using PyTorch [20]. For training, all images are first cropped or resized to 128x128. All models are optimized using the Adam algorithm [16] with a starting learning rate of 1e-4 that is decreased by a factor of 10 whenever there is no noticeable improvement in the validation loss. Training is done for a maximum of 100 epochs, with early-stopping when the validation loss stalls for too long.

To evaluate the rate-distortion performance of our compressive autoencoder, we train our model using a subset of the Youtube-8M Dataset [21]. To show that content-aware image compression can achieve superior performance over traditional algorithms, we train our compressive autoencoder on the aligned CelebA Dataset [19]. To measure the rate-accuracy performance for task-aware image compression, we use the Flowers Recognition Dataset[1] to jointly train the compressive autoencoder and the classifier. For the classifier, we use ResNet-18 [24] to classify images of flowers into five distinct classes.

To measure perceptual distortion, we use PSNR and MS-SSIM with a window size of 7 pixels. Average bits-per-pixel (bpp) over compressed test images are used to measure the degree of compression. To evaluate the classification performance of our task-aware compression network, we measure the accuracy of the ResNet over the test set.

4.2 Results

We first report rate-distortion performance of our variable-rate compressive autoencoder and content-aware compressive autoencoder compared to that of JPEG in Figure 5. From these graphs, it is evident that our compressive autoencoder significantly outperforms JPEG, obtaining a better PSNR and MS-SSIM values over all bitrates. However, content-aware compression achieves no noticeable improvement from content-agnostic compression.

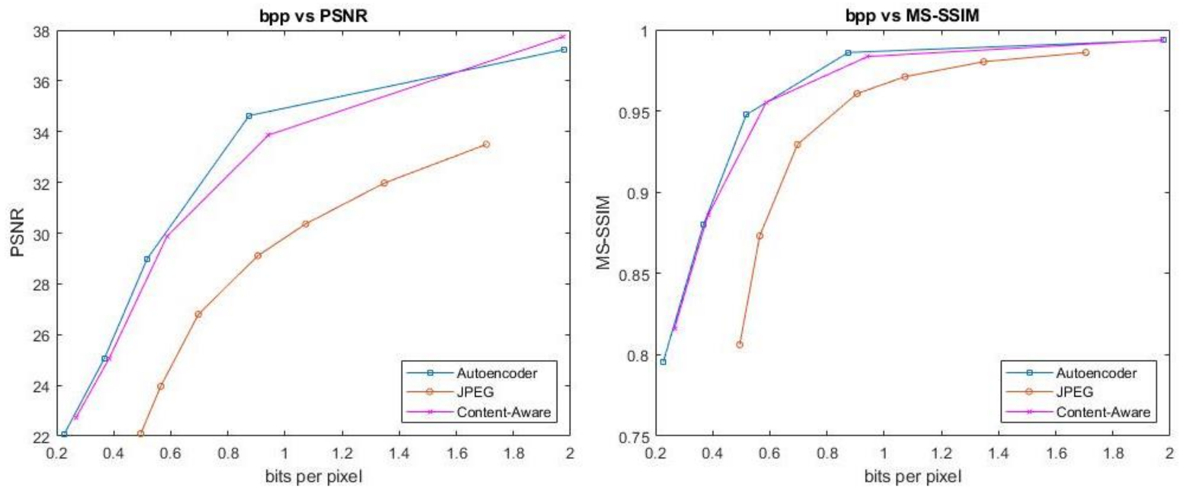

Figure 5: Rate-distortion curves for bpp vs. PSNR (left) and bpp vs. MS-SSIM (right)

Also, we report rate-accuracy performance of our task-aware compressive autoencoder compared to that of JPEG in Figure 6. We measure JPEG performance in two different ways for a fair comparison with our method. First, we measure the accuracy achieved by training with JPEG compressed images and testing with JPEG compressed images. Then, we also measure the accuracy achieved by training with the original images and testing with JPEG compressed images. Overall, our results show that task-aware compression is extremely effective compared to JPEG for low bpp compression, but becomes less efficient for higher bpp compression.

[1] https://www.kaggle.com/alxmamaev/flowers-recognition

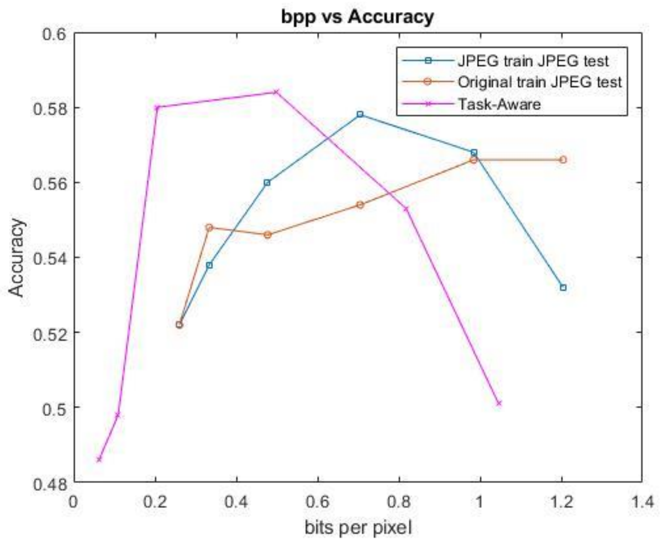

Figure 6: Rate-accuracy curves for bpp vs. flower classification accuracy.

5. Discussion

From our results, our compressive autoencoder clearly better exploits inter-pixel redundancies caused by spatial dependencies between pixels. We accredit this superior performance to the flexible non-linearities in the autoencoder, a larger field of view for the transformation, and our choice of 4x4 strided transpose convolution layers to suppress checkerboard artifacts.

The lack of performance improvement in the content-aware design is most likely due to the diversity of the 200,000 faces in the CelebA dataset. Unlike frames from surveillance cameras which have a stationary component, the faces from the CelebA dataset vary greatly in color, background, facial structure, illumination, and rotation. We believe using a less diverse dataset with some stationary components will show increased performance and leave it as future work.

For the task-aware design, we observed during training that providing a sufficient entropy error signal surprisingly helps the training of the classification network, resulting in faster convergence and higher accuracies. This phenomena can also be observed in Figure 6, where task-aware compression performance peaks around 0.4 bpp and decreases for higher bpp. We believe this phenomenon occurs due to the compression-classification trade-off inherent in the joint loss function. At higher bitrates, the joint loss function is parameterized to favor the rate loss and the reconstruction loss, often resulting in degraded classification performance. Nevertheless, our task-aware compression model shows promising results, achieving about 10% classification performance improvement from JPEG at the peak of its performance.

6. Conclusion and Future Work

We propose a variable rate image compression algorithm using compressive autoencoders, and attempt to maximize compression performance by incorporating content-aware and task-awareness into our model. Our compressive autoencoder achieves better performance than JPEG over two widely used perception metrics, and show that content-awareness and task-awareness can potentially improve compression performance under the right circumstances.

For future work, we plan to incorporate loss-conditional training [29] so that instead of training a model for each bitrate, only one model can be trained to support multiple bitrates. Furthermore, we plan to train our model with better perceptual metrics such as MS-SSIM, as mean-squared loss does not best reflect human perception [30]. We also plan to extend the expressiveness of our model by training our model with other multi-objective loss functions. In terms of evaluation, we plan to make more detailed comparisons between our work and the current state-of-the art deep compression networks.

7. References

[1] Kaur, R., & , P. (2016). A Review of Image Compression Techniques. *International Journal of Computer Applications*, 142, 8-11.
[2] G. K. Wallace (1992). The JPEG still picture compression standard. *IEEE Transactions on Consumer Electronics*, 38(1), xviii-xxxiv.
[3] D. Huffman (1952). A method for the construction of minimum-redundancy codes. *Resonance*, 11, 91-99.
[4] G. Toderici, S. O'Malley, S. J. Hwang, D. Vincent, David Minnen, S. Baluja, M. Covell, & R. Sukthankar (2016). Variable Rate Image Compression with Recurrent Neural Networks. *CoRR*, abs/1511.06085.
[5] Johannes Ballé and Valero Laparra and Eero P. Simoncelli (2017). End-to-end Optimized Image Compression. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017*, Conference Track Proceedings.
[6] George Toderici and Damien Vincent and Nick Johnston and Sung Jin Hwang and David Minnen and Joel Shor and Michele Covell (2017). Full Resolution Image Compression with Recurrent Neural Networks. In *2017 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017, Honolulu, HI, USA, July 21-26, 2017* (pp. 5435–5443). IEEE Computer Society.
[7] Lucas Theis and Wenzhe Shi and Andrew Cunningham and Ferenc Huszár (2017). Lossy Image Compression with Compressive Autoencoders. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017,* Conference Track Proceedings.
[8] Oren Rippel and Lubomir D. Bourdev (2017). Real-Time Adaptive Image Compression. In *Proceedings of the 34th International Conference on Machine Learning, ICML 2017, Sydney, NSW, Australia, 6-11 August 2017* (pp. 2922–2930). PMLR.
[9] Mu Li and Wangmeng Zuo and Shuhang Gu and Debin Zhao and David Zhang (2018). Learning Convolutional Networks for Content-Weighted Image Compression. In *2018 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2018, Salt Lake City, UT, USA, June 18-22, 2018* (pp. 3214–3223). IEEE Computer Society.
[10] Hyunho Yeo and Youngmok Jung and Jaehong Kim and Jinwoo Shin and Dongsu Han (2018). Neural Adaptive Content-aware Internet Video Delivery. In *13th USENIX Symposium on Operating Systems Design and Implementation, OSDI 2018, Carlsbad, CA, USA, October 8-10, 2018* (pp. 645–661). USENIX Association.
[11] Vincent Dumoulin and Francesco Visin (2016). A guide to convolution arithmetic for deep learning. *CoRR*, abs/1603.07285.
[12] W. Shi, J. Caballero, F. Huszár, J. Totz, A. P. Aitken, R. Bishop, D. Rueckert, & Z. Wang (2016). Real-Time Single Image and Video Super-Resolution Using an Efficient Sub-Pixel Convolutional Neural Network. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (pp. 1874-1883).
[13] Johannes Ballé and Valero Laparra and Eero P. Simoncelli (2016). End-to-end optimization of nonlinear transform codes for perceptual quality. In *2016 Picture Coding Symposium, PCS 2016, Nuremberg, Germany, December 4-7, 2016* (pp. 1–5). IEEE.
[14] J. V. Jensen (1906). Sur les fonctions convexes et les inégalités entre les valeurs moyennes. *Acta Mathematica*, 30, 175-193.
[15] Martin J. Wainwright and Eero P. Simoncelli (1999). Scale Mixtures of Gaussians and the Statistics of Natural Images. In *Advances in Neural Information Processing Systems 12, [NIPS Conference, Denver, Colorado, USA, November 29 - December 4, 1999]* (pp. 855–861). The MIT Press.

[16] Diederik P. Kingma and Jimmy Ba (2015). Adam: A Method for Stochastic Optimization. In *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015*, Conference Track Proceedings.
[17] C. E. Shannon (1948). A mathematical theory of communication. *The Bell System Technical Journal*, *27*(3), 379-423.
[18] G. Nigel Martin (1979). Range encoding: An algorithm for removing redundancy from a digitized message. In *Video & Data Recording Conference, Southampton, UK, July 24–27, 1979.*
[19] Liu, Z., Luo, P., Wang, X., & Tang, X. (2015). Deep Learning Face Attributes in the Wild. In *Proceedings of International Conference on Computer Vision (ICCV).*
[20] Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Kopf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., Bai, J., & Chintala, S. (2019). PyTorch: An Imperative Style, High-Performance Deep Learning Library. Curran Associates, Inc..
[21] Sami Abu-El-Haija, Nisarg Kothari, Joonseok Lee, Paul Natsev, George Toderici, Balakrishnan Varadarajan, & Sudheendra Vijayanarasimhan. (2016). YouTube-8M: A Large-Scale Video Classification Benchmark.
[22] Sergey Ioffe and Christian Szegedy (2015). Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. In *Proceedings of the 32nd International Conference on Machine Learning, ICML 2015, Lille, France, 6-11 July 2015* (pp. 448–456). JMLR.org.
[23] Johannes Ballé (2018). Efficient Nonlinear Transforms for Lossy Image Compression. In *2018 Picture Coding Symposium, PCS 2018, San Francisco, CA, USA, June 24-27, 2018* (pp. 248–252). IEEE.
[24] Kaiming He and Xiangyu Zhang and Shaoqing Ren and Jian Sun (2016). Deep Residual Learning for Image Recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016, Las Vegas, NV, USA, June 27-30, 2016* (pp. 770–778). IEEE Computer Society.
[25] Odena, A., Dumoulin, V., & Olah, C. (2016). Deconvolution and Checkerboard Artifacts. *Distill.*
[26] Cisco. "Global 2020 Forecast Highlights." Cisco, www.cisco.com/c/dam/m/en_us/solutions/service-provider/vni-forecast-highlights/pdf/Global_2020_Forecast_Highlights.pdf.
[27] A. Ghosh, D. Chakraborty, & A. Law (2018). Artificial intelligence in Internet of things *CAAI Transactions on Intelligence Technology, 3(4), 208-218.*
[28] Z. Wang, E. P. Simoncelli, & A. C. Bovik (2003). Multiscale structural similarity for image quality assessment. In *The Thrity-Seventh Asilomar Conference on Signals, Systems Computers, 2003* (pp. 1398-1402 Vol.2).
[29] Alexey Dosovitskiy and Josip Djolonga (2020). You Only Train Once: Loss-Conditional Training of Deep Networks. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net.
[30] Q. Huynh-Thu, & M. Ghanbari (2008). Scope of validity of PSNR in image/video quality assessment. *Electronics Letters, 44(13), 800-801.*