[2021 Winter/Spring semester URP] KAIST URP Program

Individual Research Project Final Report

Research Participation Project

부가 정보를 활용한 과제인지 이미지 압축

Using Side Information for Task-Aware Image Compression

Suro Lee / Computer Science (20160830)

Heecheol Yun / Electrical Engineering (20170440)

For Submission

Research Subject (Korean) : 부가 정보를 활용한 과제인지 이미지 압축

Research Subject (English) : Using Side Information for Task-Aware Image Compression

Research Period : January 4, 2021 ~ June 18, 2021

Advisory Professor : Dongsu Han (Signature)

Teaching Assistant : Heonho Yeo (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.

June 18, 2021

Researcher : Suro Lee (Signature)

Researcher : Heecheol Yun (Signature)

Contents

Abstract

This paper introduces two novel methods of performing classification with deep compressed images. Deep image compression has already surpassed most traditional image compression methods, and there has recently been few attempts to perform task-aware image compression with the deep-compressed images. This is possible due to the flexibility of deep image compression models that allows joint optimization with respect to two or more different losses.

We first present results from classification-aware compression performance as background work. Then, we attempt to enhance classification performance from deep-compressed images by exploiting the fact that most channels of the compressed code do not contain information relevant to image classification. From the channels of our compressed code, we extract two types of side information. The first side information is the entropy of the channel, and the second side information is extracted by a convolutional neural network. The two types of side information are used to enhance classification performance through two different methods. First, we use the bottleneck attention module to enhance our compressed code with self-attention obtained from the side information. Second, we use feature-wise linear modulation to condition layers of the classification model with side information.

Our results, however, degrade classification performance slightly. For future work, we plan to try a variety of other methods that utilize channel-wise information in order to enhance classification performance. In addition, we plan to solve various problems related to the training of a multi-objective neural network such as stabilizing training and increasing convergence speed.

1. Introduction

Recently, deep image compression algorithms using CNNs [1,2,3,4] have achieved notable advances, far surpassing JPEG [9] and obtaining comparable performance to that of BPG [10]. Essentially, deep image compression compresses an image by sending the image through a deep encoder and obtaining a vector representation of the image. This code is then quantized and entropy coded to further compress the code. As in an autoencoder, this quantized code is then sent to the decoder where it attempts to reconstruct the original image from the compressed code.

Deep image compression is superior to traditional image compression methods not only because of its performance, but also because of its flexibility. While traditional image compression methods can only be optimized for a single perceptual metric (i.e., mean squared loss), deep image compression methods can be optimized jointly for various perceptual metrics and/or task metrics. Such flexibility of deep image compression was exploited in [11], where the authors used the compressed code to perform various tasks such as classification and segmentation. In our approach, we extend the multi-objective model of Torfason et al. in an attempt to improve classification performance. In Torfason et al.'s work, we noticed that the overall structure of the image was preserved in only some of the channels of the compressed code. That is, the channels of the compressed code vary in their entropy. We exploit this fact by using another deep network to gather the channel-wise information and extract side information. This side information is used for either 1. enhancing our compressed code through the bottleneck attention module (BAM) [12] or 2. conditioning layers of the classification network using feature-wise linear modulation (FiLM) [13].

We first show that the multi-objective model outperforms a widely-used traditional compression algorithm, JPEG, in terms of image reconstruction. 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) [7]. In addition, we show that the multi-objective model demonstrates comparable classification performance by comparing the top-1 accuracy with the accuracy of ResNet32 [6].

Then, we demonstrate the motivation for our work by visualizing the channels of the compressed codes and observing that most of the channels do not contain information helpful for classification. Using this fact, we perform classification by utilizing side information from the channels of the compressed code. Our results show slightly degraded classification accuracy but reveal areas for future research.

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 [9]. Thus, lossy compression algorithms consist of three main parts that target each redundancy (Figure 1).

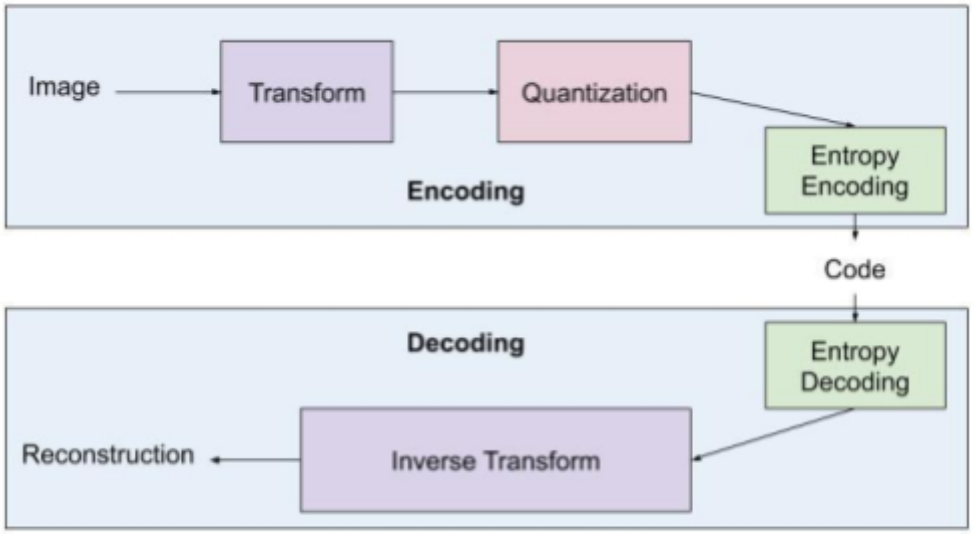

Figure 1: Lossy image compression

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 [9], 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 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) or a convolutional neural network (CNN) [1,2,3,4]. CNNs need a separate model for each target bitrate, while RNNs can be compressed into various bitrates with a single model.

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. This whole process can be learned in end-to-end fashion by minimizing Eq. (3), where y represents the output of the encoding, $\hat{y}$ represents the quantized feature vector, and $\hat{x}$ represents the output of the decoding. The entropy loss, L_e is defined as the entropy of the ŷ, which is an approximation of bitrates needed to represent ŷ, and the distortion, L_r is measured by a norm or perceptual metric. Total loss function is a linear combination of the entropy loss and the distortion. Since a separate model is trained for each bitrate when using a CNN in a compression network, β is required to decide a target bitrate by balancing the distortion and entropy loss.

$$y = E(x), \hat{y} = Q(y), \hat{x} = D(\hat{y}) \tag{1}$$

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

$$L = L_e + \beta L_r \tag{3}$$

Increasingly, more and more deep lossy image compression algorithms are outperforming traditional lossy image compression algorithms.

2.3 Compressive Autoencoder

Most recent compression networks are based on autoencoders. Autoencoders add flexibility to the compression network because it can be trained with various content types. However, it is hard to be trained due to the non-differentiability of the entropy loss. A compressive autoencoder handles this problem by introducing two main functions:

$$Q : \mathbb{R}^N \to \mathbb{Z}^N, P : \mathbb{Z}^N \to [0, 1] \tag{4}$$

where Q is the quantization function, and P is the discrete probability model. There have been many attempts to obtain a differentiable approximation of the quantization function, some of which include adding uniform noise, using binarization, and using the derivative of a smooth approximation. In compressive autoencoders, quantization function is smoothed to Qy=y≅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.

Since P is the discrete probability function, taking the derivative of P(y) with respect to y is undefined. Consequently, compressive autoencoders define a differentiable probability model, p, to approximate P(·) in the reconstruction loss as follows:

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

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

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

For the differentiable probability model, compressive autoencoders use gaussian scale mixtures (GSMs), which are known to accurately characterize the probability distribution of natural image wavelet coefficients. 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.

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

The model architecture of a compressive autoencoder is 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 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 was not used in any convolution layers because it typically results in suboptimal compression performance.

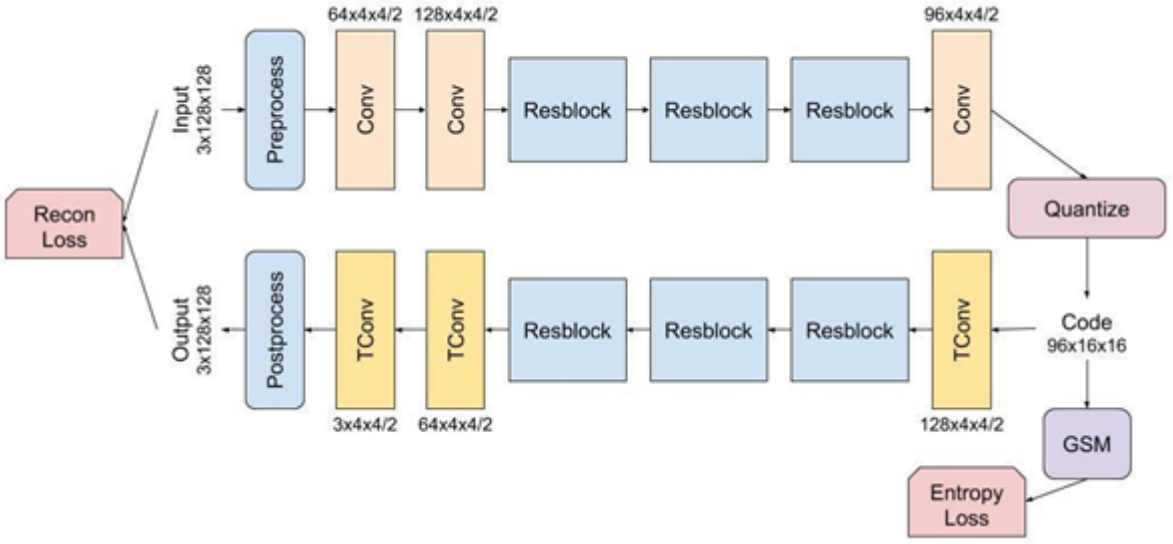

Figure 2: The compressive autoencoder. The C×K×K/S labels indicate convolutions and transposed convolutions with C output channels, K×K filters, and stride S.

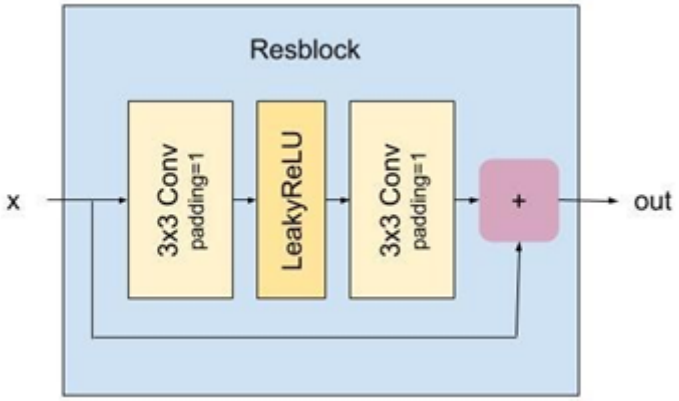

Figure 3: The Resblock layer used in the compressive autoencoder.

2.4 Task-Awareness

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 Multi-objective Deep Image Compression

Task-aware deep neural networks make it possible to learn multiple tasks in an end-to-end manner with compressed features. For example, we can use compressed features which are obtained by a pre-trained encoder in several tasks such as image classification and reconstruction. However, optimal compressed features for image classification and reconstruction will vary from one another because the features that are important in image reconstruction may not be important in image classification. With joint training, it is possible to learn a single compressed feature, which can be used in several different tasks. In [11], the authors jointly trained an encoder-decoder network and image classification model and showed comparable performance compared to when both models were trained separately. The block diagram of the model is shown in Figure 3. The authors used a compressive autoencoder [2] for the compression network and a ResNet [6] for the classification network. However, since compressed features are used as the inputs of the ResNet, they did not use the full architecture of ResNet. They cut off the front part of the ResNet which has a larger spatial dimension than the compressed features and called it 'cResnet'. Images are first compressed with an encoder and these feature vectors can be used for various tasks. As done in this work, we can solve various tasks with a simpler model and less storage.

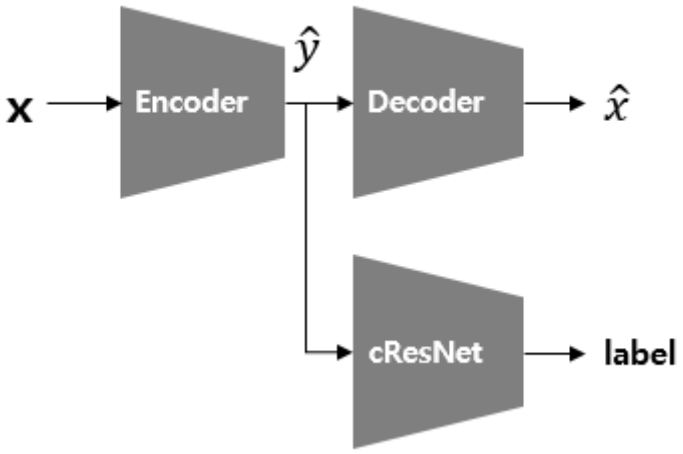

Figure 4 We can use compressed features $\hat{y}$ in various tasks.

We followed the similar architecture used in [11], but the compression network is a little different. Because how exactly we approximate a discrete marginal distribution as a continuous distribution has the greatest impact on performance(the role of the probability model), we modified the quantization function and discrete probability model of the compression network. For the quantization, we added uniform noise [1] to the feature vector instead of rounding(Eq. (5)). That is,

$$\tilde{y} = y + \Delta y \tag{9}$$

where Δy, is uniformly sampled from [-0.5,0.5].

For the probability model, many recent compression networks have shown modeling the prior by convolving the underlying density model p with a standard uniform(Eq. (10)) shows better performance than using the upper bound of the entropy loss(Eq. (7)). [15], [16]

$$
\begin{aligned}
p(\tilde{y}) &= \left(p * \mathcal{U}(-\frac{1}{2}, \frac{1}{2})\right)(\tilde{y}) \\
&= \int_{-\infty}^{\infty} p(y)\mathcal{U}(\tilde{y} - y| - \frac{1}{2}, \frac{1}{2})dy \\
&= \int_{\tilde{y}-\frac{1}{2}}^{\tilde{y}+\frac{1}{2}} p(y)dy
\end{aligned}
\tag{10}
$$

Eq. (10) holds because we used uniform additive noise in the quantization(Eq. (9)).

3.2 Side Information Extraction

In [11], the encoder-decoder is pre-trained with the Imagenet dataset and then the weights of the encoder are fixed while training the cResnet. Afterwards, the model is fine-tuned in joint training. However, since decoding(reconstruction) needs much more accurate information than classification, some features might be redundant in classification. Also, some features which are useful in classification might be discarded while training a compression network. As shown in Figure 5, some channels' information might not be helpful in classification. Therefore, to highlight the important features, we used bitrates which are required to represent each channel of the feature. For example, if the first channel of the feature needs 0.1 bpp(bits per pixel) while the second channel needs 0.02 bpp, then we can conclude the second channel is more important than the first channel because low entropy means high randomness. Therefore, we send the bitrate information of each channel to the classification network so that the model learns which part of the feature should be given more bits during compression. Since this concept is similar to an attention module, we modified existing attention modules [12], [13] to give side information to the classification network. These modules are based on a simple convolutional layer and fully connected layer, and will be further explained in the section 3.3 and 3.4. Although we used the side information only in image classification, the side information can be easily applied in various tasks.

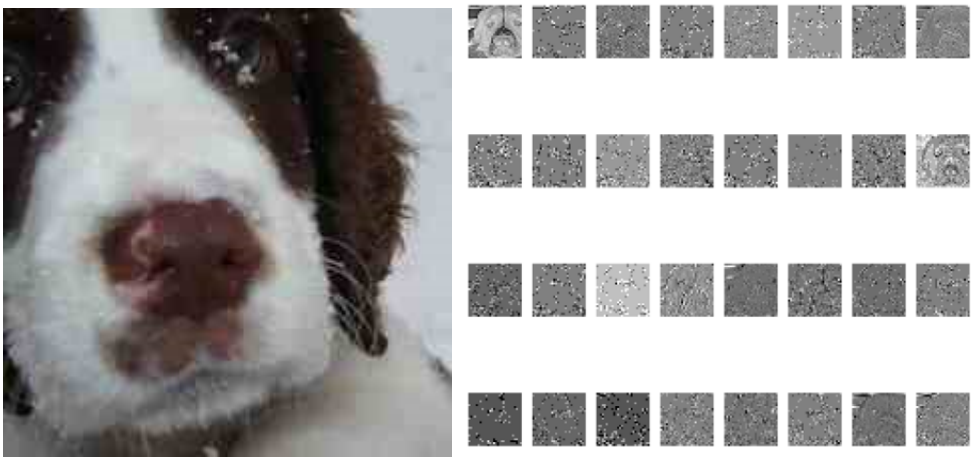

Figure 5: (Left) Original image (Right) Each image represents each channel of the compressed feature. In this figure, the encoder outputs features which have 32 channels.

To sum up, an input image is first compressed by the encoder and then we extract bitrate of each channel of the compressed code and use it as the side information of the classification

network. We used the modified compressive autoencoder explained in 3.1 for the compression network and used ResNet50 to obtain cResNet for the classification network.

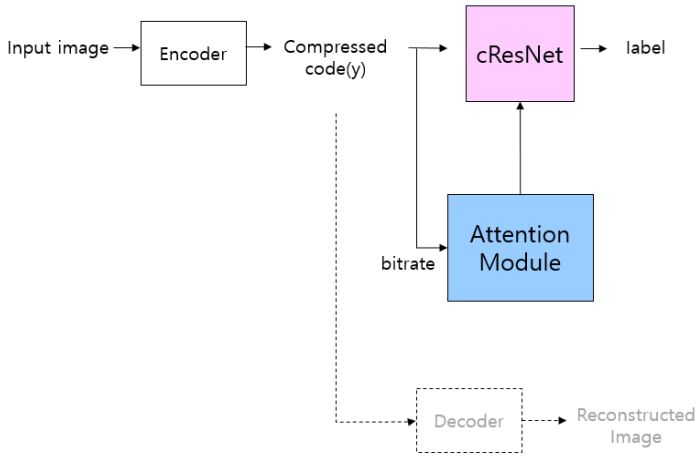

Figure 6: Block diagram of our model. Since we want to emphasize our attention module, the decoder is represented as a dashed line.

3.3 Bottleneck Attention Module

First, we modified CBAM(Convolutional Bottleneck Attention Module)[12] to give side information to the classification network. CBAM is used in the bottleneck of the model(usually before the pooling in a convolutional layer) to capture the important information before downsampling. It has a channel attention module and spatial attention module. A channel attention module calculates the attention in a channel axis and a spatial attention module calculates the attention in a spatial axis. Since we want to use bitrate of each channel, we modified the channel attention module in CBAM. Instead of applying a convolution layer to the whole feature to extract the channel attention, we just passed bitrate information into a channel attention module. Also, we attached the channel attention module only at the front of the classification network because bitrate information is likely to be useless as the feature vector is changed by the compression network. We tested 2 different attention modules: 1. Using only channel attention module 2. Using both channel attention module and spatial attention module. For the spatial attention module, we used the same architecture described in CBAM.

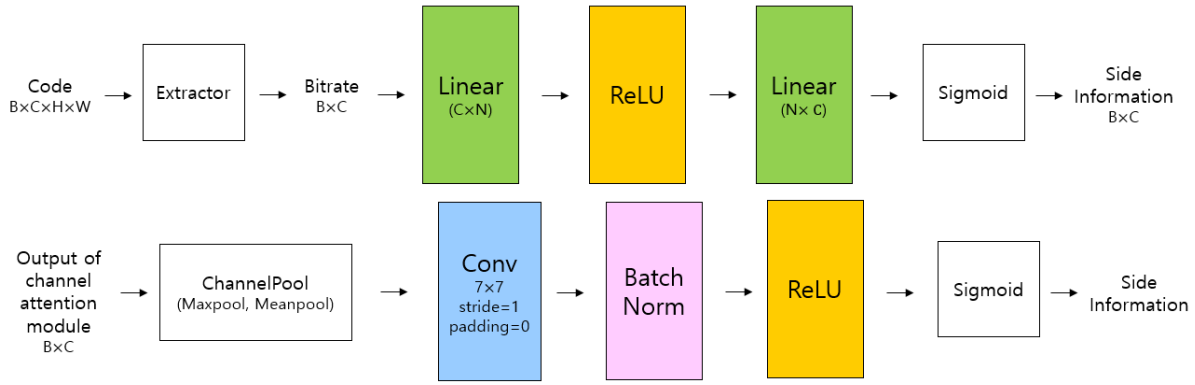

Figure 7: (Top) Channel attention module for extracting side information from compressed code. C is the channel of the compressed code and N depends on which compression network we used. (Bottom) Spatial attention module. ChannelPool is a concatenation of max pooling and mean pooling.

3.4 Feature-wise Linear Modulation

Feature-wise linear modulation (FiLM) is performed with two types of side information: information extracted from the entropy of each channel through a multilayer perceptron network and information extracted from the compressed code through a residual convolution network. The multilayer perceptron network for extracting side information from the entropy of each code channel consists of two linear layers with hidden dimension of 128. The residual convolutional network for extracting side information from the compressed code is shown below in Figure 8.

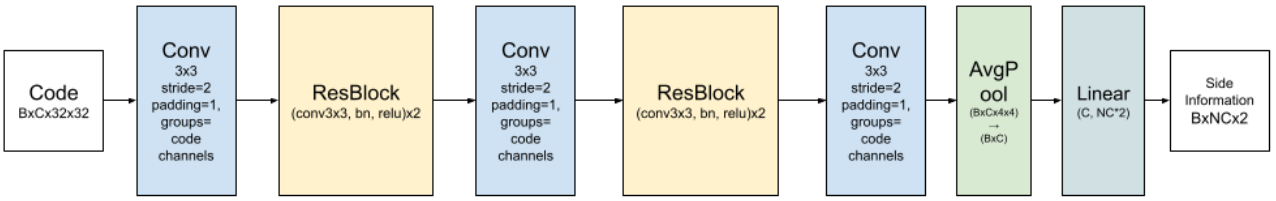

Figure 8: Residual convolutional network for extracting side information from compressed code. C is the channels of the compressed code and NC is the total number of channels in the last layer of each residual block.

As described in [13], FiLM conditions the last layer of each residual block of the classification model through $\gamma_{n,c}$ and $\beta_{n,c}$ extracted from side information, where n denotes n^{th} residual block and c is the channel number of the feature from the last layer of the residual block. The modulation operation is performed on entire channel as follows: $\gamma_{n,c} F_{n,c} + \beta_{n,c}$, where $F_{n,c}$ is the c^{th} channel of the last layer of the n^{th} residual block.

4. Results and Discussion

4.1 Experimental Setup

We evaluate two main aspects of our work in detail: the rate-distortion-classification performance of our compressive autoencoder and the classification performance improvement achieved by using side information. For each of the two evaluations, we also compare our performance with that of JPEG. All deep learning models and training/test scripts were implemented in Python using PyTorch. For training, all images are first cropped to 224x224. Initially, only the compressive autoencoder is trained for 100 epochs with the Adam optimizer with a learning rate of 1e-4. Then, the weights of the compressive autoencoder are fixed and the classification network is trained for 30 epochs with the SGD optimizer with a learning rate of 5e-3 that is decreased by a factor of 10 at fixed intervals. Lastly, all models are trained jointly for 10 epochs with the SGD optimizer with a learning rate of 2.5e-6 that is decreased by a factor of 10 every 3 epochs. For low bpp, we use 8 or 16 bottleneck channels, while for high bpp we use 32 bottleneck channels.

To evaluate the rate-distortion-classification performance of our compressive autoencoder, we train our model using the Imagenette dataset [14], which is a subset of the ImageNet dataset. To measure perceptual distortion, we use PSNR and MS-SSIM. 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 top-1 accuracy over the test set.

4.2 Multi-objective Compression Results

We first report rate-distortion performance of our compressive autoencoder before joint compression and after joint compression compared to that of JPEG in Figure 9 and 10. 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, joint training of the two models seems to achieve no noticeable improvement. The lack of improvement is most likely due to the learning rate, which had to be kept low to ensure stable training.

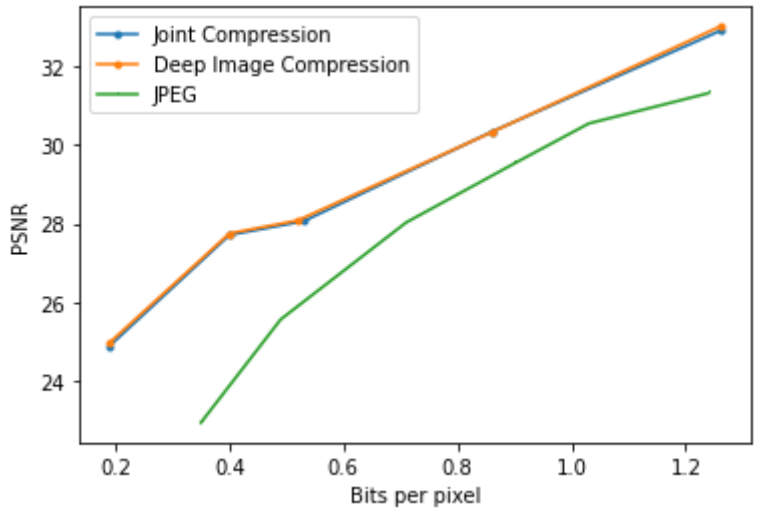

Figure 9: bits per pixel vs. PSNR performance for JPEG, deep image compression prior to joint compression and deep image compression after joint compression.

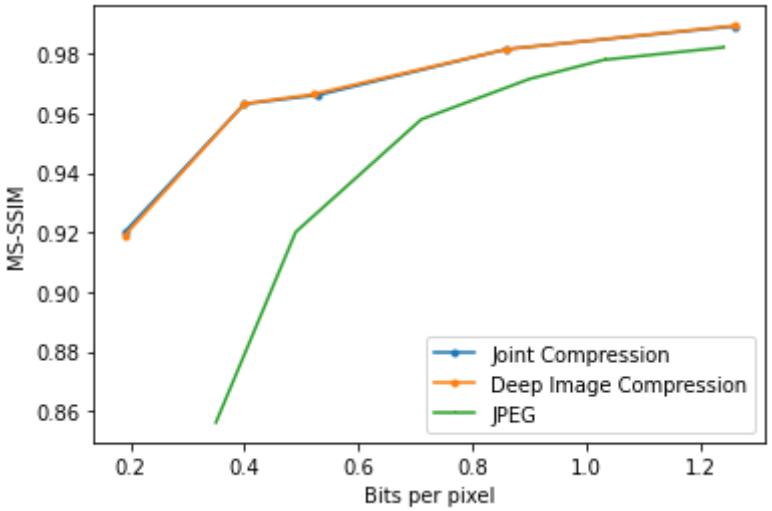

Figure 10: bits per pixel vs. MS-SSIM performance for JPEG, deep image compression

prior to joint compression and deep image compression after joint compression.

Also, we report rate-accuracy performance of our task-aware model compared to that of JPEG in Figure 11. We measure JPEG performance by compressing our original images to the designated bpp and then performing classification with the 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. This result shows that deep image compression effectively keeps information required for classification in the compressed code, which JPEG is not optimized to do.

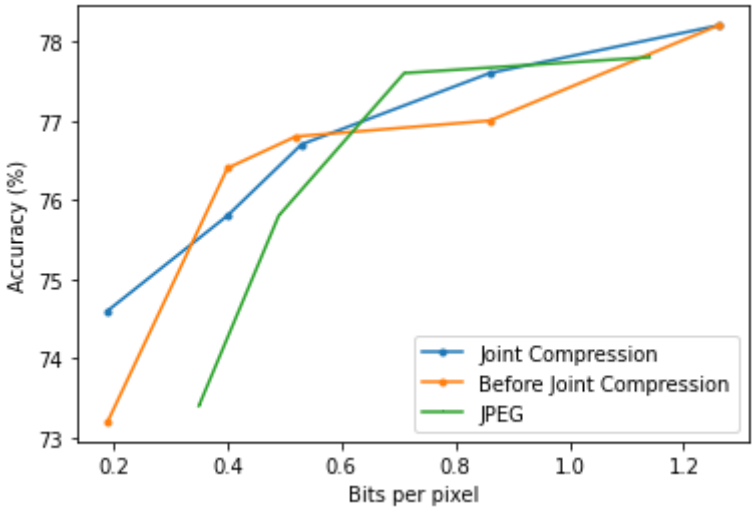

Figure 11: bits per pixel vs top-1 accuracy performance for JPEG, deep image compression prior to joint compression and deep image compression after joint compression.

4.3 Bottleneck Attention Module Results

The results after applying modified CBAM for different values of rate-classification tradeoff parameters are shown in below tables. 'Original' means the cResNet without the attention module, 'Channel' indicates the cResNet with only channel attention module, and 'Channel + spatial' indicates the cResNet with both channel attention module and spatial attention module.

	Accuracy(%)
Original	76.2
Channel	64.4
Channel + spatial	72.3

Table 1: Accuracy of the models with 0.38 bpp(bits per pixel)

	Accuracy(%)
Original	76.7
Channel	74.6
Channel + spatial	75.7

Table 2: Accuracy of the models with 0.74 bpp(bits per pixel)

From the results, there was no improvement after applying modified CBAM in the classification network. They showed even worse accuracy than the original model. However, we could see a clear improvement after using the spatial attention module. Also, the gap between performance of original model and model with attention module has decreased significantly as bpp increases. Shallow attention modules and few channels in the compressed code are the possible reasons for the bad results. Since CBAM is usually applied to features which have hundreds of channels, CBAM might not be an appropriate module in this case. Also, the channel attention module may not be effective when it is not combined with the spatial attention module.

4.4 Feature-wise Linear Modulation Results

The results after applying FiLM for different values of rate-classification tradeoff parameters are shown in Table 3. All measurements were made with a constant bpp of 0.52.

	PSNR	Accuracy (%)
Original	28.0420	75.6
FiLM_entropy	28.0431	59.6
FiLM_conv1	28.0621	68.9
FiLM_conv2	23.7585	75.0

Table 3: PSNR and accuracy of the original task-aware model compared to applying FiLM with two different types of side information.

From the results, it is clear that using FiLM with side information extracted from the entropy of each channel results in a significantly worse accuracy. For FiLM with side information extracted from the code using residual convolutional networks, either the PSNR or accuracy was degraded slightly compared to that of the original task-aware model. This phenomena may be because too few channels were used in the compressed code. For 0.52 bpp, we used only 8 channels, thus most channels likely had important information critical for classifying the images correctly.

5. Conclusion and Future Work

We propose two novel methods to utilize side information obtained from the compressed code in two different ways. Our methods resulted in slightly worse classification

performance than the original model, which may be due to low bottleneck channels, our choice of side information, and our choice of applying side information in classification.

For future work, we plan to experiment with different kinds of side information that may be obtained from the compression network or the compressed code, and also we plan to try other attention or modulation methods than the two methods proposed in the paper.

6. References

[1] 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.
[2] 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.
[3] 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.
[4] 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.
[5] 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.*
[6] 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.
[7] 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).
[8] Q. Huynh-Thu, & M. Ghanbari (2008). Scope of validity of PSNR in image/video quality assessment. *Electronics Letters, 44(13), 800-801.*
[9] Gregory K. Wallace, . "The JPEG Still Picture Compression Standard".*Commun. ACM* 34, no.4 (1991): 30–44.
[10] Sullivan, Gary J., Jens-Rainer, Ohm, Woo-Jin, Han, and Thomas, Wiegand. "Overview of the High Efficiency Video Coding (HEVC) Standard".*IEEE Transactions on Circuits and Systems for Video Technology* 22, no.12 (2012): 1649-1668.
[11] Robert Torfason and Fabian Mentzer and Eirikur Agustsson and Michael Tschannen and Radu Timofte and Luc Van Gool, . "Towards Image Understanding from Deep Compression Without Decoding." . In *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*. OpenReview.net, 2018.
[12] Sanghyun Woo and Jongchan Park and Joon-Young Lee and In So Kweon, . "CBAM: Convolutional Block Attention Module." . In *Computer Vision - ECCV 2018 - 15th European Conference, Munich, Germany, September 8-14, 2018, Proceedings, Part VII* (pp. 3–19). Springer, 2018.
[13] Ethan Perez and Florian Strub and Harm de Vries and Vincent Dumoulin and Aaron C. Courville, . "FiLM: Visual Reasoning with a General Conditioning Layer." . In *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence, (AAAI-18), the 30th innovative Applications of Artificial Intelligence (IAAI-18), and the 8th AAAI Symposium on Educational Advances in Artificial Intelligence (EAAI-18), New Orleans, Louisiana, USA, February 2-7, 2018* (pp. 3942–3951). AAAI Press, 2018.

[14] fastai. “Fastai/Imagenette.” GitHub. Accessed June 23, 2021. https://github.com/fastai/imagenette.
[15] Johannes Ballé and David Minnen and Saurabh Singh and Sung Jin Hwang and Nick Johnston, . "Variational image compression with a scale hyperprior." . In *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*. OpenReview.net, 2018.
[16] David Minnen and Johannes Ballé and George Toderici, . "Joint Autoregressive and Hierarchical Priors for Learned Image Compression." . In *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, December 3-8, 2018, Montréal, Canada* (pp. 10794–10803).2018.