Residual CNN Image Compression

Kunal Deshmukh(✉) and Chris Pollett

San Jose State University, San Jose, CA 95192, USA
kunaldeshmukh27@gmail.com, chris@pollett.org

Abstract. We present a neural network architecture inspired by the end-to-end compression framework [1]. Our model consists of an image compression module, an arithmetic encoder, an arithmetic decoder, and an image reconstruction module. We evaluate the compression rate and the closeness of the reconstructed image to the original image for this model. Structural similarity metrics and peak signal to noise ratio are used to evaluate the image quality. We have also measured the net reduction in file size after compression and compared it with other lossy image compression techniques. Our architecture achieves better results in terms of these metrics compared to traditional and newly proposed image compression algorithms. In particular, an average PSNR of 28.48 and SSIM value of 0.86 are obtained as compared to 28.45 PSNR and 0.81 SSIM value in the previously mentioned network.

Keywords: Convolutional Neural Networks · Generative adversarial networks · Structural similarity metrics · Peak signal to noise ratio

1 Introduction

Several recently published articles [1,2] on image processing and compression frameworks have used deep learning networks. Parts and ideas from several of these frameworks seemed like they might augment each other to give better results. The present paper explore this by developing a new hybrid architecture which we show improves upon existing architectures for both efficiency and image quality metrics such as SSIM, PSNR while still being as time efficient as possible.

The first architecture we leveraged was Jiang et al. [1]. They consider an end to end image compression network in which a fully convolutional neural network (CNN) based encoder is used for image compression. Their inspiration was that in JPEG to achieve higher compression one uses bigger quantization steps. Unfortunately, these bigger steps result in blocking artifacts during decoding. Zhai et al. [3] and Foi et al. [4] had previously considered image based hand-crafted deblocking filters to reduce this problem. Jiang, et al propose instead that one should try to use a CNN to learn filters which would do the same job as efficiently as possible.

Jiang et al.'s architecture serves as our baseline model to compare with our approach. In Jiang et al.'s architecture, a smaller image is first constructed by

© Springer Nature Switzerland AG 2019
G. Bebis et al. (Eds.): ISVC 2019, LNCS 11845, pp. 371–382, 2019.
https://doi.org/10.1007/978-3-030-33723-0_30

the encoder. This image is nothing but a downsampled version of the original image. The decoder, called the re-constructor, is designed to generate an original image back from this smaller image.

Cavigelli et al. [2] propose a different architecture that is efficient at suppressing artifacts that are introduced during the image compression process. To do this their model makes use of skip connections. Our proposed architecture also makes use of skip connections. Besides suppressing artifacts, this helped our model converge 19.65% faster as compared to a network without skip connections. Space separable CNN layers such as pointwise and depthwise CNN layers are useful to speedup the training and inference. We used such CNN layers for some of the levels in our neural network. This further reduced the training as well as inference time by about 7% and 3% as compared to the network using vanilla CNN layers. We discuss what these layer types are and these result in detail in Sect. 2.3.

Finally, we have also designed a loss function in such a way that the images generated by the compression module have a very low variance in pixel values. This limits the pixel values the resultant compressed image can have. An arithmetic encoder is then used to process this compressed image. Since the compressed image now has a very small number of distinct values, the arithmetic encoder can use a lossless data compression algorithms such as entropy coding or RLE to further compress the results obtained from compression module.

In summary, our architecture at a high level starts with the base model of Jiang et al. Compression and decompression neural networks are trained simultaneously. The decompression network is larger and makes use of skip connections and space separable CNN layers which together with the choice of our loss function speed up training. To improve image quality of our decompression network we make use of an SRGAN sub-network.

We now discuss the organization of the rest of the paper. In the next section, we fix the definitions of some of the neural network layers we are considering as well as discuss prior models that have been considered in more detail. We also discuss image quality metrics in brief. We then have a section describing our proposed architecture in detail. This is followed by training methodology and experiments and, finally, there is a conclusion section.

2 Background

We assume the reader is familiar with Convolutional Neural Networks (CNNs), Recurrent Neural Networks (RNNs), etc. as well as the fundamental concepts in deep neural network training such as training and loss functions. Information on these topics can be found in Goodfellow et al. [5].

2.1 The Use of Convolutional Neural Networks in Image Compression

Our architecture relies heavily on CNNs to capture image artifacts. CNNs have been used recently in many image compression architectures.

Jiang et al. [1] use a fully convolutional auto-encoder to obtain a compressed representation of an image. Their auto-encoder consists of a series of two kinds of convolutional layers stacked one after the other to capture features of the image. The authors claim that, because of the use of multi-layer CNNs, this architecture maintains the structural composition of the image. The two distinct convolutional layers networks they use are called the ComCNN and the RecCNN. The ComCNN network is responsible for compressing images in such a way that the resultant images can be effectively reconstructed by a reconstruction network. The ComCNN network consists of three convolutional layers with the second layer followed by a batch normalization layer. Since the first convolutional layer uses a stride of two, the image size is reduced by half. The RecCNN layer uses twenty CNN layers. Apart from the first and the last layer, each layer in this architecture carries out convolutional and batch normalization operations. The authors train this network using 400 grayscale images for 50 epochs. The SSIM and PSNR metrics for these images are better than what is obtained by using JPEG. PSNR and SSIM values for 28.17 and 0.8206 respectively.

In lossy image compression techniques, artifacts of the compression are often visible in the images. One example class of artifacts is caused when tiling is used for quantization. In such images, the tile boundaries are often visible in the images. The CNN based architecture [2] proposed by Cavigelli et al. is a twelve layer image compression architecture designed to suppress such artifacts. Cavigelli et al. also give a new way to train deep neural network models which is adaptable to other low-level computer vision tasks. They propose the use of hierarchical skip connections together with a multiscale loss functions. Two advantages of skip connection are: In the forward pass, this method provides information that allows the network to obtain higher resolution images. In the backward pass, these skip connections allow gradient flow to skip middle layers and help train earlier layers more quickly. Using skip connections though does not eliminate the possibility of very long paths in the network. Hence, they calculate the loss function on many intermediate low-resolution images. The authors observes that batch normalization does not reduce the accuracy of their network.

2.2 The Use of GANs for Image Interpolation

Ledig et al. [6] use GANs to recover finer detail from an image that are lost because of compression. Their framework uses a perceptual loss function that consists of a content loss term and an adversarial loss term. We have used this idea as an efficient way to do image up-scaling.

Their architecture up-scales the images by a factor of four. The quality of images generated using this method are close to the quality of original images. This GAN's architecture uses a VGG network [7] as a Discriminator.

2.3 Space Separable Convolutional Neural Betworks

Our architecture makes use of two special types of CNN Layers: Point-wise and Depth-wise CNN layers. A point-wise convolutional operation is a special type of operation where the size of the kernel is always 1×1. This operation returns a layer of the same dimensions as that of its inputs. Depthwise CNNs work on depth. Each kernel can have any height and width, however, its depth is always one. Separate kernels act on each depth level. Stacking all these layers together results in an image.

Point-wise and Depth-wise convolutions involve fewer multiplications, hence, they are computationally efficient. However, shallow neural networks with space separable convolutions may fail to learn the underlying function.

2.4 Image Quality Metrics

Image quality metrics are used to measure how well an image compression architecture performs. Image quality metrics are of two types: Reference image quality metrics and non-reference image quality metrics. Reference image quality metrics require a reference image to compute; on the other hand, non-reference image quality metrics, such as Mean Subtracted Contrast Normalized (MSCN), do not require such a reference image.

Since we are going to compare the uncompressed image with the original image anyway for our loss function, we have used only reference image quality metrics. We next review some of these reference image quality metrics:

Mean Square Error (MSE): MSE calculates the sum of the squared differences between pixel values of two images of the same dimensions: $MSE = \frac{1}{MN}\sum_{y=1}^{M}\sum_{x=1}^{N}[I(x,y) - I'(x,y)]^2$. Here M and N are the width and height respectively of the images in pixels and $I(x,y)$ is the pixel intensity value at the image position (x,y). This is perhaps the simplest image quality metric to understand, however, it is not always a good metric to access image compression quality as it does not take into account the range of variations in pixel values in an image or the high and low-frequency components in an image, two factors which affect human perception of image quality.

Peak Signal to Noise Ratio (PSNR): PSNR is a measure of the peak error between the two images. It is computed using the equation: $PSNR = 10\log_{10}\frac{R^2}{MSE}$ where R represents the maximum fluctuation in input image pixel values. Since PSNR uses Mean Square Error in its denominator, higher PSNR values represent a better quality compression.

Structural Similarity Index (SSIM): SSIM measures image quality degradation due to processing tasks such as compression. SSIM is considered to be a better metric for accessing degradation of images as it takes into account visible structures in an image. SSIM is calculated via the following formula:

$$SSIM(x,y) = \frac{(2\mu_x\mu_y + C_1) + (2\sigma_{xy} + C_2)}{(\mu_x^2 + \mu_y^2 + C_1)(\sigma_x^2 + \sigma_y^2 + C_2)} \tag{1}$$

Here x and y represent images, μ_x, μ_y are the average pixel value for each image, σ_{xy} is the co-variance between x and y, and σ_x^2, σ_y^2 are the variances of x and y.

3 Network Design

Our proposed architecture consists of four parts: an Image Compression Module (ICM), an Arithmetic Encoder, an Arithmetic Decoder, and an Image Reconstruction Module (IRM). The ICM and IRMs result in lossy image compression while the Arithmetic Encoder and Decoder are lossless. Both the ICM and the IRM are based on deep neural networks.

3.1 The Arithmetic Coder and Decoder

Our arithmetic coder and decoder make use of a Python implementation of Huffman coding [8]. Since the original data can be completely retrieved in the decoder part, it is a lossless compression algorithm and its introduction or removal does not affect the other neural network-based modules. Thus, for neural network training, we did not use the arithmetic coder or decoder so as to avoid unnecessary computational overhead.

3.2 The Image Compression Module

The CNN layers in the ICM are used to learn features and components of the image helpful for further image reconstruction tasks. These components include the overall structure of the image as well as some salient features such as edges and corners that cannot be regenerated by the reconstruction layer unless they are provided as inputs. Thus, this module acts as a filter through which only a few critical components are passed to an intermediate image. We have used a three layer CNN module as shown in Fig. 1. We specify this network in more detail below:

The first block consists of a CNN layer followed by a rectified linear unit (ReLU) non-linearity. It uses a 3×3 kernel of a depth 3 in the case of RGB compression and of depth 1 in the case of grayscale images. This layer has 3 feature maps. The second block has a CNN layer, a ReLU activation, and a layer for batch-normalization. This block's CNN uses a 2 padding, resulting in a tensor of half the size of the original image. This layer thus reduces the size and resolution of an image by half, when the stride is 2.

The resultant tensor is passed through a batch normalization layer which helps [9] avoid gradient overflow or underflow and makes the neural network less sensitive to the choice of random initializer and its variation. A single convolution layer is used for the third block of the network. The number of kernels used in this layer corresponds to the number of channels in the output image. The output of this layer is not subject to a non-linearity as the goal of this layer is to reproduce an image as close to the original image as possible.

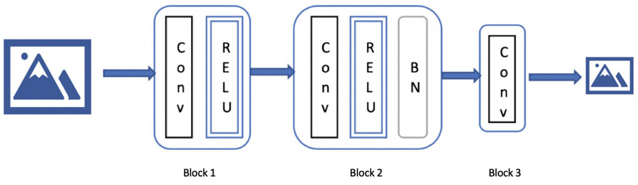


Fig. 1. Image compression module

The Image Compression Module was trained simultaneously with the Image Reconstruction Module. The goal of the compression network is to generate an intermediate representation of an original image that can be used by the reconstruction layers to generate an image as close to the original image as possible. The size of this intermediate result decides the compression factor of the compression algorithm. This compression factor can be changed by varying the stride and dilation values in the intermediate layer.

3.3 Image Reconstruction Module

The reconstruction module has two responsibilities: To resize the image to the original size and to improve the quality of the resized image.

Since this module is tasked with the reconstruction of a facsimile of the original image from the minimal information passed on by the compression module, this module requires more layers to try to detect residual information from the compressed image that might aid in reconstruction. These residuals are detected by the feature maps used in its convolutional layers. The first few of these layers are responsible for the reconstruction of basic shapes such as lines, points, corners etc. while further layers add more information about higher order combinations of these base features. The size of kernels used in this network is always 3×3 since all these features are local to a region and are less likely to have any impact on other parts of an image.

The framework in [1] uses a bicubic interpolation method to resize the compressed image to its original size. Our system uses an SRGAN (Super Resolution Generative Adverserial Network) [6] for this purpose. Our SRGAN network returns an image of size four times the size of the original image. This can be scaled down to the desired size using an interpolation technique before it is fed to the reconstruction module. The SRGAN paper shows these networks generate better quality images as compared to simple interpolation techniques, giving our network better input data for image enhancement.

Using an SRGAN allows us to use bad quality images to train the network for image enhancement. Slightly bad quality data is often generated as part of a data augmentation task for robust training. Using a bicubic interpolation instead of SRGAN during training alleviated the amount of data augmentation needed when training the reconstruction network. The SRGAN network that we used, was trained separately in its original proposed shape on the ImageNet dataset.

The IRM consists of five blocks each containing one convolutional layer and an optional batch normalization or ReLU layer.

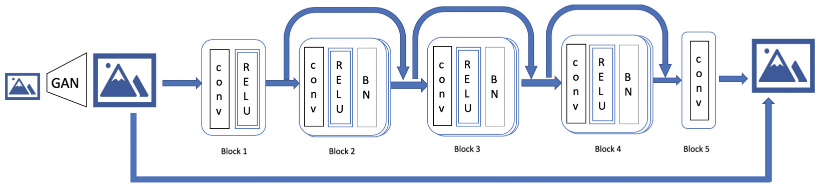


Fig. 2. Image reconstruction module.

The first block contains a CNN layer with ReLU activation function. For this layer, we used 64 features each of which had a 3×3 kernel with depth 3 for RGB images and 1 for grayscale. The second block consists of five concatenated sub-blocks each consisting of a CNN layer, a ReLU activation, and a batch normalization layer. Hence, overall, this block contains five CNN, ReLU, and batch normalization layers. The third and fourth blocks are identical to the second block. Pointwise and depthwise convolution functions are applied to the second block. The fifth and last block consists of a CNN layer which takes in 64 channels and outputs a tensor of size height $\times$ width$\times$ # of channels.

After the 2^{nd}, 3^{rd}, and 4^{th} blocks, a skip connection is added as shown in Fig. 2. Then a ReLU is applied after this.

An interpolated image from the start of the network is added to the output of the fifth block. The resultant tensor is saved as an image. As we have discussed in previous section, skip connections are useful in backpropagation. The use of ReLU activations also ensures that the tensors passed in these layers are subjected to enough non-linearity so that the network weights in successive layers are not updated in a uniform manner during training. This activation function also ensures that the resultant values in tensors are restricted to the desired domain. Since we have used batch normalization layers in the network, the use of ReLU does not contribute to an exploding gradient problem. For skip connections, two tensors of same dimension are added and each new tensor is passed through at least one convolutional layer, this ensures that the convolutional layer weights are adjusted in the training phase so as to accommodate the values obtained from skip connections.

3.4 Loss Function

A loss function computes an error between the desired values and those output by a neural network. The training algorithm tries to adjust weights so as to minimize this function. For our training procedure, the loss function was calculated using the following formula:

$$L = \frac{1}{M}\sum_{y=1}^{M}[(\sum_{i,j} P_{ij}(y) - P_{ij}(\widehat{y}))^2 + |(P_{ij}(I_{in}) - P_{ij}(I_{out}))|] \qquad (2)$$

In Eq. 2, M is the number of images and the function $P_{i,j}(I)$ outputs the i,jth pixel of image I. The first term in Eq. 2 is the mean square error between the input image to the ICM and the output image from the IRM. The second term is the mean absolute error between the input image to the first CNN block of IRM and the output image after the fifth block of the IRM before they are added together. It can be viewed as a regularization term. I_{in} is after we have upscaled the compressed image. We do not want this image to be substantially different from the final output, so we add this as a penalty term.

4 Experiments

Experiments were conducted using the publicly available image datasets: STL10, CIFAR10, COCO and CLIC. We varied the dataset, the numbers of epochs and network sizes during our experiments, however, our best results were obtained for 50 epochs on the COCO dataset. Training required more than 110 h on a Google Cloud instance that had an Nvidia Tesla P 100 GPU. SRGAN and arithmetic encoder and decoders were not added to the network during training. We used an Adam optimizer [10] and Xavier initialization [11] for all experiments.

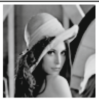
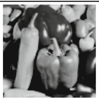
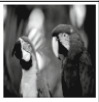
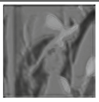
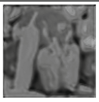
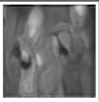
Our first experiment was a reimplementation of the baseline model of Jiang et al. [1]. We did not achieve exactly the same results as claimed by these authors as we could not obtain the original dataset for training used by authors. Instead, we trained our version on the STL10 and COCO datasets, the COCO dataset giving the better results. We implemented the image augmentation techniques as described in their paper. That is, we generated images from our starting images by random horizontal and vertical flips and by rotations of angles between 0 and 360°. Our results below are for the COCO dataset of 330 thousand images. All training images were 200×200 pixels. The COCO dataset contains a variety of image sizes 650×450, so we also performed image croppings by randomly choosing a base point from the image and then cropping out a 200×200 image at this base point. Each epoch during our training had 330 thousand images, where each image was generated from the COCO dataset image using the random processes just described. Twenty percent of the images were used in each epoch for validation. Finally, for testing we used the Kodak image dataset of 25 images where we re-scaled these images to 200×200.

For comparison, Jiang et al. used images of size 180×180. They created augmentations of 400 starting images and tested using 7 images from the Kodak dataset. We observe that our network performance increased when we increased the number of images in our augmented datasets.

Table 1. Baseline model results

Image name	PSNR	SSIM
Lenna	28.07	0.59
Peppers	27.9088	0.48
Parrots	28.036	0.61
Average	28.00	0.56

Table 2. Examples of baseline model results

Type	Lenna	Peppers	Parrots
Original Image			
Reconstructed Image			

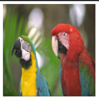
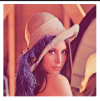
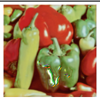
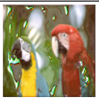
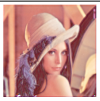
Our baseline model results are shown in Tables 1 and 2. Overall, we obtained a PSNR 28.45 and an SSIM 0.81 for their model; whereas, their paper obtains a PSNR of 30.17 and an SSIM 0.88. However, to achieve comparable results we needed to use more data. This might be due to the particularities of their data set, which we did not have, as well as to tuning and initialization factors in their training. Table 2 shows some examples of compressed and uncompressed images we obtained on the test data for this experiment. Table 2 shows results only for grayscale images as only grayscale images were used by Jiang et al.

We next describe the experiments we conducted using our proposed neural network architecture. These experiments were also conducted using the COCO dataset with the same image augmentations. For these experiments we used color rather than grayscale images. Our results are given in Tables 3 and 4.

Table 3. Color image results for the proposed model

Image name	PSNR	SSIM
Baboon	27.7869	0.89
Lenna	29.26	0.86
Peppers	28.46	0.79
Parrots	28.33	0.69
Average	28.45	0.81

Table 4. Example color images using the proposed model

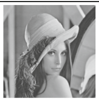
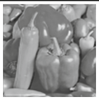
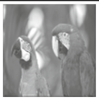
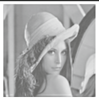
Type	Peppers	Parrots	Lenna
Original Image			
Reconstructed Image			

These results show some distortion in the color images as the IRM introduced errors during image regeneration. This distortion might be reduced if deeper neural networks were used in tandem with a larger dataset. It might also be reduced by using smoothing or removal of high-frequency areas. We conducted experiments for both color and gray scale images. Grayscale was used so we could compare our results to what was used in the baseline model. To handle grayscale we slightly modified the first and last layer of our networks so they dealt with only one channel. The grayscale results are shown in Tables 5 and 6.

Table 5. Grayscale results for the proposed model

Image name	PSNR	SSIM
Cameraman	27.34	0.78
Peppers	28.30	0.88
Lenna	29.88	0.87
Parrots	28.23	0.89
House	28.67	0.89
Average	28.48	0.86

Table 6. Example grayscale images using the proposed model

Type	Peppers	Parrots	Lenna
Original Image			
Reconstructed Image			

From our results, we see that the performance of our model on grayscale is better than on color images. For grayscale, each image pixel is represented by a single value, for RGB, three values are required to define a single pixel. This increases the possibility of error in color images. Grayscale images are generated using a two-dimensional tensor. Since the possibility of error is reduced by the factor of three, errors in grayscale are not easily perceived by a human eye. As the SSIM metric is modeled after human perception, understandably grayscale perform better for the SSIM metric as compared to PSNR. In addition to comparing the image quality of our proposed model with the baseline model, we also conducted experiments looking at the time to train our model, the time to carry out compression and decompression of images using our model, and the average compression sizes obtained via our model.

On the COCO dataset, the average time to train the baseline framework for one epoch was 173 min as compared to 139 min for our framework.

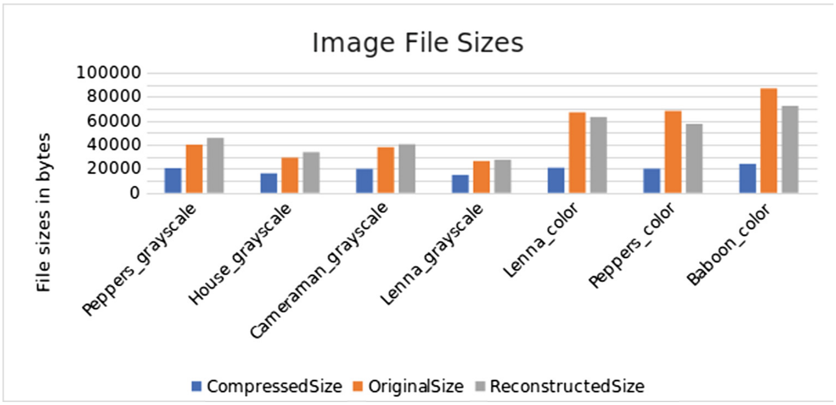


Fig. 3. Image file size before and after compression

Compression and decompression speeds were measured in images/second. We tested 200 images from COCO dataset. The average compression time for the baseline model was 3.2 and for the proposed architecture was 3.3. These were comparable as these networks were essentially the same. On the other hand, the average decompression time for the baseline model was 4.1 and for the proposed architecture was 3.4. The inference time is slower than the baseline model probably due to the evaluation of the SRGAN portion of the reconstruction as it add more than twenty layers to the model. Replacing this portion withta simpler bicubic upscaling, gave a value of 4.3 while worsening the image quality to an average PSNR of 28.44 and SSIM of 0.85. In terms of compressed files sizes, Fig. 3 shows that the file size of an image was substantially reduced after going through the image compression module and the arithmetic encoder. In this figure, the original size is that of the JPEG (already compressed) image from the data set.

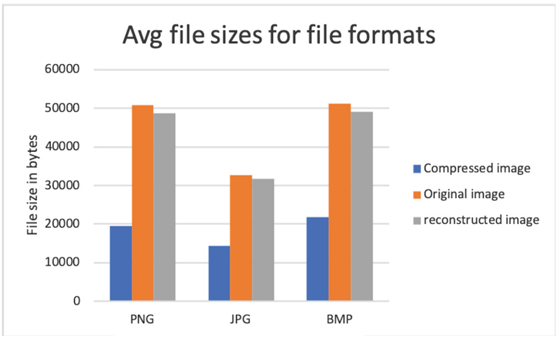


Fig. 4. Average file size for image file format

Figure 4 shows the compression achieved for images of various file types. The compressed image files were obtained by taking the original file, reading it into our compression neural network, computing its compressed image and then saving the result in the same file format as was input. So if the original image was PNG, we read it in, applied our neural network to obtain a compressed image, and then saved that compressed image as a PNG. Similarly, the decompressed file sizes were obtained by reading in the compressed image in some format X, applying our neural network, and writing out an image in format X. It can be concluded from this that our architecture is useful for a variety of file formats. The loss and overall quality for the image file formats do not vary substantially with a change in file formats. The same was true for implementation of a baseline model as well.

5 Conclusion

Autoencoders seem particularly useful for image compression and reconstruction. Using residual connections and space separable layers, reduced training by approximately 20% over the baseline model [1]'s training. Our model also achieves slightly better quality with a PSNR of 28.48 and SSIM of 0.86 are obtained as compared to 28.45 PSNR and 0.81 SSIM, with only a slight change in decompression time. A slightly lower quality version of our model where the SRGAN portion is replaced with bicubic interpolation actually gives a faster decompression time than the baseline with only a slightly reduction in quality of the output. Our method provides a way to achieve image compression after it is processed by traditional image compression algorithms such as JPEG. Hence, it can be used to supplement traditional image compression and encoding algorithms. We expect that in the future a variety of interesting experiments and improvements can be made in the field of neural network compression. For example, even without changing our model we might obtain faster training times for our network and potentially higher compression ratios by focusing on images coming from a single category such as just car images.

References

1. Jiang, F., Tao, W., Liu, S., Ren, J., Guo, X., Zhao, D.: An end-to-end compression framework based on convolutional neural networks. IEEE Trans. Circuits Syst. Video Technol. **28**(10), 3007–3018 (2018)
2. Cavigelli, L., Hager, P., Benini, L.: CAS-CNN: a deep convolutional neural network for image compression artifact suppression. In: International Joint Conference on Neural Networks (IJCNN), pp. 752–759 (2017)
3. Zhai, G., Zhang, W., Yang, X., Lin, W., Xu, Y.: Efficient image deblocking based on postfiltering in shifted windows. IEEE Trans. Circuits Syst. Video Technol. **18**, 122–126 (2008)
4. Foi, A., Katkovnik, V., Egiazarian, K.: Pointwise shape-adaptive DCT denoising with structure preservation in luminance-chrominance space. IEEE Trans. Image Process. **16**, 1395–1411 (2006)
5. Goodfellow, I., Bengio, Y., Courville, A.: Deep Learning. MIT Press, Cambridge (2016). http://www.deeplearningbook.org
6. Ledig, C., et al.: Photo-realistic single image super-resolution using a generative adversarial network. In: IEEE Conference on Computer Vision and Pattern Recognition, pp. 4681–4690 (2017)
7. Krizhevsky, A., Sutskever, I., Hinton, G.: Very deep convolutional networks for large-scale image recognition, arXiv preprint arXiv:1409.1556 (2014)
8. Knuth, D.: Dynamic huffman coding. J. Algorithms **6**(2), 163–80 (1985)
9. Ioffe, S., Szegedy, C.: Batch normalization: accelerating deep network training by reducing internal covariate shift, arXiv preprint arXiv:1502.03167 (2015)
10. Kingma, D., Ba, J.: Adam: A method for stochastic optimization, arXiv preprint arXiv:1412.6980, December 2014
11. Thimm, G., Fiesler, E.: Neural network initialization. In: International Workshop on Artificial Neural Networks, pp. 535–542 (1995)