SHALLOW AUTOENCODERS FOR LOSSY IMAGE COMPRESSION

Mario Sergio Fuentes Juarez
Department of Computer Science
The University of Texas at Austin
Austin, TX 78705, USA
mario.sergio@utexas.edu

ABSTRACT

We propose the use of shallow autoencoders as a self-contained lossy image compression format. Our method involves using an autoencoder network to learn the representation of a single image, and store the decoder submodule and the subimage-block codes as a compressed representation of the given image. Ideal conditions involve a small-sized decoder module and subimage-block codes of minimal length. This decoder/codes bundle can yield compression with respect to the storage of a raw image under proper architecture and code length choices. We explored two different autoencoder architectures (CNN and LSTM+CNN autoencoders) and measured their yielded compression ratio and image quality loss. We also measured the performance of the LSTM+CNN architecture under different code lengths. Although the results are not comparable to the ones produced by JPEG (both in terms of compression ratio and image quality loss), we believe that our work suggests the feasibility of near-lossless image compression with shallow LSTM+CNN autoencoders.

1 INTRODUCTION

Image compression through neural networks has been an active research topic since the 1980s and to this date [3, 5, 1]. Most of the recent work in the field has been devoted to developing novel architectures that are able to effectively compress images at different compression rates and image sizes [6]. Research has also focused on leveraging recent developments in deep autoencoder architectures, modular neural networks, CNNs, and RNNs [9, 7, 2].

Although many of these developments have reported compression ratios and image quality preservation comparable to JPEG [6, 5], to the best of our understanding, there has been little interest on developing an end-to-end image compression format that involves a neural network in its architecture. We believe that this has to do with the large dimensions of those well-performing compression architectures, whose large size makes it unfeasible to make them part of a self-contained compression format.

In this paper, we propose an image compression format that is the composition of the decoder module of an autoencoder network and the subimage-block codes of an image. Thus, the size of our compressed representation of an image is the addition of the size of its decoder network and the total size of the subimage-block codes that represent that image. It is worth noting that, in order to yield competitive compression ratios, the decoder network's size should be kept as small as possible, hence the shallowness requirement.

We present two different shallow autoencoder architectures: (1) a multilayer CNN autoencoder and (2) a multilayer LSTM+CNN autoencoder. We report their achieved compression ratio and image quality loss (measured using MS-SSIM [7, 8]). We compare these results to those of JPEG as our baseline reference. Finally, we also compare the performance of the multilayer LSTM+CNN autoencoder across different code length choices.

Although the compression ratio and quality preservation yielded by both architectures fall behind JPEG's, the LSTM+CNN autoencoder architecture did achieve close-to-complete image reconstruction results. We believe that this result should motivate more research on shallow autoencoder networks for lossless image compression.

2 Related Work

Autoencoder-based image compression was first discussed by Cottrell, Munro, and Zipser in the late 1980s [3]. Dony and Simon described this approach as modelling image compression as ”an encoder problem in which a network is forced to perform an identity mapping through a ’narrow’ channel” [3].

Toderici et al. [6] define the components of an autoencoder network as follows: (a) an encoder which consumes some image patch as input and transforms that through (b) a bottleneck layer that produces some compressed representation of the data, which in turn is used by (c) a decoder that attempts to reconstruct the original input. This architecture is usually trained through feedforward and backpropgation mechanisms. Reconstruction is achieved by having the expected output be equal to the given input. After training, the encoder and decoder sub-networks can be used separately for compression or reconstruction purposes [6].

Autoencoders have evolved from the basic input-bottleneck-output layer architecture to deeper and more involved architectures. Namphol et al. [4] were among the first to propose multi-hidden-layer autoencoders. Watanabe et al. [9] suggested a multi-autoencoder approach where different image patches could be compressed with different autoencoders depending on the compression complexity of the given image block.

In recent years, several researchers have leveraged developments in RNNs and CNNs to create novel autoencoder architectures [6, 7, 2]. For example, Toderici et al. [7] proposed different autoencoder architectures that integrated CNN, GRU and LSTM layers, achieving a compression performance similar to JPEG.

These developments in convolutional and recurrent autoencoder networks inspired the autoencoder architectures that we report in later sections.

3 Compression Mechanism and Architectures

In this section, we first present an outline of the compression method, then discuss the compression feasibility criterion for our method, and finally present the two shallow autoencoder architectures that we used in our experiments.

3.1 Compression method

The compression mechanism can be outlined as follows:

1. The input image is decomposed into square image patches (e.g. 32x32 pixels).
2. These subimage-blocks are individually fed into an autoencoder network, which is trained via basic feedforward and backpropagation.
3. After training ends, one final pass is made, presenting all the subimage-blocks to the autoencoder and extracting the outputs produced by the bottleneck later. These outputs are the compressed code representations generated for each subimage-block.
4. The encoder half of the autoencoder is discarded. The final size of the compressed representation of the image is the size of the decoder network plus the size of all subimage-block codes.

3.2 Compression feasibility criterion for a shallow autoencoder

In order for any compression to take place, the size of the compressed representation of an image must be smaller than the size of the original image. Hence, the following inequality must hold:

$$S_d + c_l * n_b < P_i * 3$$

In this inequality, S_d is the size in bytes of the decoder network, c_l is the code length (e.g. the number of nodes in the bottleneck layer multiplied by the size of a node in bytes), n_b is the number of subimage-blocks in which the input image is decomposed, and P_i is the total number of pixels in the input image.

This inequality expresses that the size in bytes of the decoder network and the subimage-block codes must be smaller than the size in bytes of the original input image. We assume that the input image is a raw image in RGB format, where each each color channel is represented with an 8-bit integer (1 byte). Hence, the bytes size of the original image is measured as the number of pixels in it, P_i, times the 3 color channels contained in each pixel.

This compression feasibility criterion was taken into account for the choice of autoencoder architectures, image size, and subimage-block size in our experiments.

3.3 Shallow multi-layer CNN autoencoder

The first autoencoder that we tested was a basic multi-layer CNN network. The encoder sub-network contains two convolutional layers, a max-pooling layer, a dropout layer, and a flattening layer that connects to a 64-nodes-long bottleneck layer. The decoder sub-network is densely connected with the bottleneck layer and performs a simple deconvolution to reconstruct the input image shape.

Figure 1 shows more details about the architecture.

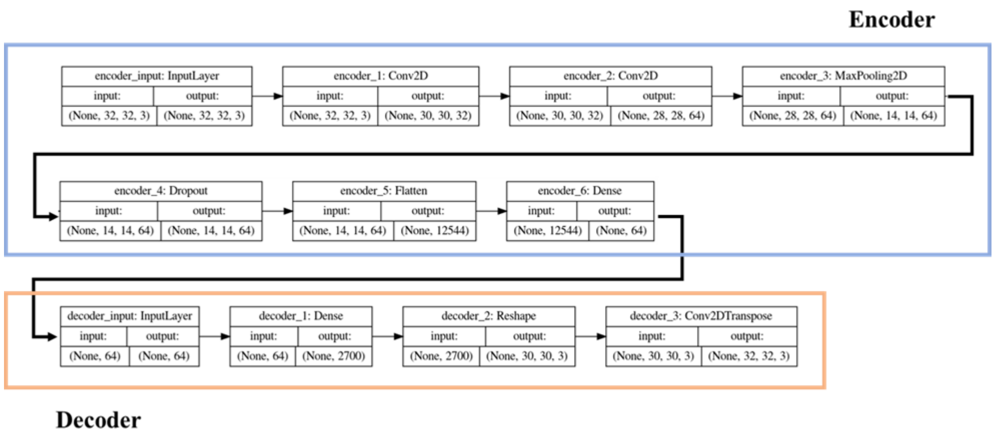


Figure 1: Architecture of shallow multi-layer CNN autoencoder

As seen in the previous figure, the autoencoder's input and output layers handle volumes of dimension 32x32x3. The reason for this shape is that, during our experiments, we chose to subdivide the input images into blocks of size 32x32 pixels. We believe that future work could explore the effect of varying the subimage-block size.

3.4 Shallow multi-layer LSTM+CNN autoencoder

Our second autoencoder was modelled after the architecture presented in [7]: we incorporated three convolutional LSTM layers in the encoder network, and connected to a 64-nodes-long bottleneck layer. The decoder sub-network is similar to the one in our simple CNN autoencoder.

Figure 2 shows the architecture in greater detail.

4 Experiments

We carried out two different sets of experiments. In our first experiments, we ran the compression method using our two different autoencoder architectures to compress two test images. We compared the performance of the these two autoencoders among themselves and against JPEG, in

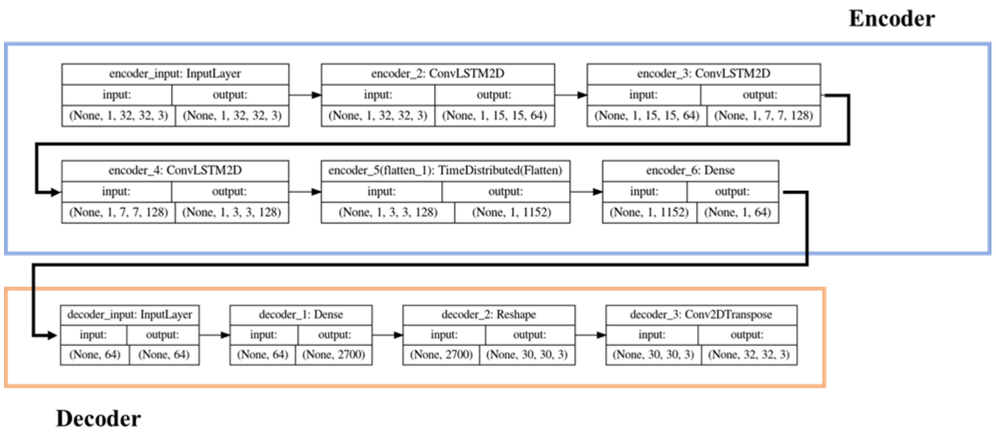

Encoder
encoder_input: InputLayer
input: (None, 1, 32, 32, 3)
output: (None, 1, 32, 32, 3)
encoder_2: ConvLSTM2D
input: (None, 1, 32, 32, 3)
output: (None, 1, 15, 15, 64)
encoder_3: ConvLSTM2D
input: (None, 1, 15, 15, 64)
output: (None, 1, 7, 7, 128)
encoder_4: ConvLSTM2D
input: (None, 1, 7, 7, 128)
output: (None, 1, 3, 3, 128)
encoder_5(flatten_1): TimeDistributed(Flatten)
input: (None, 1, 3, 3, 128)
output: (None, 1, 1152)
encoder_6: Dense
input: (None, 1, 1152)
output: (None, 1, 64)
decoder_input: InputLayer
input: (None, 64)
output: (None, 64)
decoder_1: Dense
input: (None, 64)
output: (None, 2700)
decoder_2: Reshape
input: (None, 2700)
output: (None, 30, 30, 3)
decoder_3: Conv2DTranspose
input: (None, 30, 30, 3)
output: (None, 32, 32, 3)
Decoder

Figure 2: Architecture of shallow multi-layer LSTM+CNN autoencoder

terms of quality loss and compression ratio. Our second set of experiments were carried out on the LSTM+CNN architecture, and consisted of compressing a test image with varying code lengths. We compared the compression ratio and quality loss results across different code lengths.

4.1 LSTM VS LSTM+CNN AUTOENCODERS

We ran our compression mechanism using the two previously described autoencoder architectures. We chose to test two images with different characteristics, one of them containing large ”flat” regions of color and the other one composed of many ridged areas. Both images were RGB-encoded in TIFF format, and 640x640 pixels in size (1.23 MB in disk space).

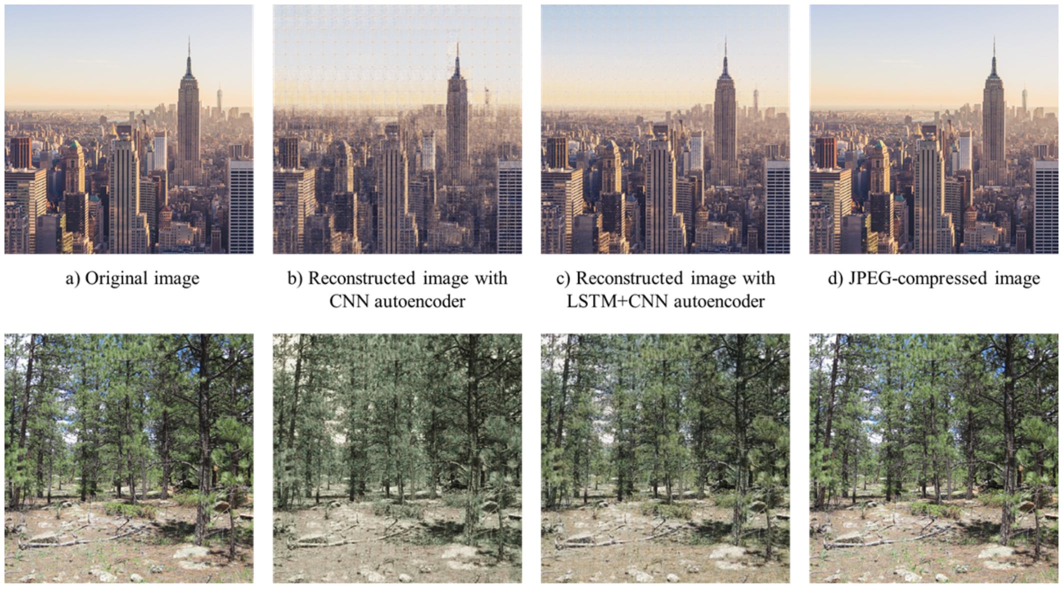
The compression method was carried out by training the autoencoders for 300 epochs. The input image was decomposed into subimage-blocks of size 32x32 pixels (for a total of 400 such blocks per image), and those blocks were presented to the autoencoders one by one, aiming for reconstruction using pure gradient descent (no batch or stochastic gradient descent). We used the RMS Prop algorithm for the optimizer (using TensorFlow/Keras’ default parameters), and our chosen loss function was L2, that is, the sum of the squared differences between the ”real” and the ”reconstructed” values of the RGB channels of all pixels. For more implementation details, please refer to our public repository in `https://github.com/msf1013/shallow-autoencoders-compression`.

Figure 3 shows how the two different test images look after being compressed and then reconstructed with with our CNN and LSTM+CNN autoencoders.

Naked eye, it is possible to identify multiple graphical artifacts in our reconstructed images. Such artifacts include imperfect reconstruction of sharp edges, washed out colors, and grid-looking reconstruction corresponding to the subimage-block decomposition of the image. Although both autoencoders yield these artifacts, visual imperfections are more evident when analyzing the output of the CNN autoencoder.

Tables 1 and 2 summarize the image quality loss and compression ratio achieved by both autoencoders for the test images, as well as the results obtained after compressing the input images using JPEG with a 90% quality parameter. Image quality loss was measured in terms of multi-scale structural similarity [8], comparing the similarity between the original input image and the reconstructed compressed image. This measure ranges between 0.0 and 1.0, with values closer to 1.0 indicating greater image similarity. Compression with JPEG was done using Linux’s built-in *convert* tool.

Our results prove that image compression with our two autoencoder architectures was largely outperformed by JPEG, both in terms of compression ratio and quality loss. Not only was JPEG compression ratio between 3 and 6 times higher, but also the image quality was better preserved by JPEG, as noted by the MS-SSIM metric in Tables 1 and 2.



a) Original image

b) Reconstructed image with CNN autoencoder

c) Reconstructed image with LSTM+CNN autoencoder

d) JPEG-compressed image

e) Original image

f) Reconstructed image with CNN autoencoder

g) Reconstructed image with LSTM+CNN autoencoder

h) JPEG-compressed image

Figure 3: Comparison of test images after compression and recontruction. Pictures obtained from Civitatis and US Forest Service websites without any commercial purposes.

Compression method	Compression ratio	MS-SSIM	Compressed image size
CNN autoencoder	1.53	0.8722	805 KB
LSTM+CNN autoencoder	1.53	0.9679	805 KB
JPEG (quality=90%)	8.53	0.9966	144 KB

Table 1: Compression results for test image no. 1

It is worth noting that the compressed image size yielded by our autoencoders was the same regardless of the architecture and the input image. This had to do with the fact that both architectures shared the same decoder sub-network (as shown in Figures 1 and 2), and the two test images had the same dimensions (640x640 pixels). Hence, the final compressed image size was equal to the decoder sub-network size (175,584 float32 weights, equivalent to $\sim$702 KB) plus the total size of the subimage-block codes (input images were decomposed into 400 subimage-blocks, yielding 400 subimage-block codes consisting of 64 float32's each, for a total of $\sim$102 KB).

In spite of the somewhat disappointing results of these experiments, we were positively surprised by the high image-similarity yielded by the LSTM+CNN autoencoder architecture. Although some artifacts were spotted after image reconstruction, the MS-SSIM metric yielded by the LSTM+CNN autoencoder closely approached JPEG's, and clearly outperformed the results of the basic CNN architecture. This made us hypothesize that such architecture might be a promising one in terms of image quality preservation, and more work should be devoted to further fine-tuning. For our second round of experiments (varying code length), we decided to stick to this LSTM+CNN autoencoder architecture.

4.2 LSTM+CNN AUTOENCODER WITH VARYING CODE LENGTHS

For our second set of experiments, we chose to keep the autoencoder architecture fixed (LSTM+CNN autoencoder) and vary the code length of the bottleneck layer. We measured the compression ratio and MS-SSIM yielded with bottleneck layers of size 32, 64 and 80 nodes (that is, code lengths of size 32, 64, and 80). We used a single test image, borrowed from our first set

Compression method	Compression ratio	MS-SSIM	Compressed image size
CNN autoencoder	1.53	0.9381	805 KB
LSTM+CNN autoencoder	1.53	0.9546	805 KB
JPEG (quality=90%)	5.13	0.9985	240 KB

Table 2: Compression results for test image no. 2

Code length	Compression ratio	MS-SSIM	Compressed image size
32	3.02	0.9386	408 KB
64	1.53	0.9679	805 KB
80	1.22	0.9720	1 MB

Table 3: Compression results for different code lengths in the LSTM+CNN autoencoder

of experiments (image no. 1, containing large flat regions of color). We trained these autoencoders following the same procedure and settings explained in our first experiments.

Figure 4 shows the reconstructed image after being compressed using the LSTM+CNN autoencoder with bottleneck layers of size 32, 64, and 80.

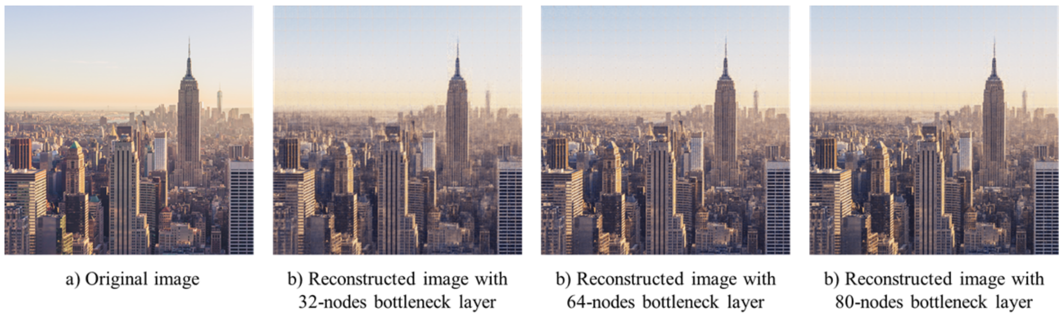

a) Original image
b) Reconstructed image with 32-nodes bottleneck layer
b) Reconstructed image with 64-nodes bottleneck layer
b) Reconstructed image with 80-nodes bottleneck layer

Figure 4: Comparison of test image after compression and recontruction.

As in our first round of experiments, the three different image reconstructions showed graphical artifacts, this time mostly related to a grid-looking reconstruction. Naked eye, the autoencoder with 80-nodes-long bottleneck layer is the one in which these artifacts were less noticeable.

Table 3 summarizes the image quality loss and compression ratio achieved by the three autoencoder variations. As expected, a smaller code length yielded higher compression ratios: the compression ratio achieved with the 32-nodes-long bottleneck layer was almost 3 times higher than the 80-nodes-long bottleneck layer. This has to do with the reduced code length representing each subimage block, and also with a less dense decoder sub-network: a lower number of bottleneck layer nodes required less weighted connections in the decoder half of the autoencoder, reducing the size of the decoder network and thus the overall size of the compressed image.

Nonetheless, a higher compression ratio comes at the expense of lesser image quality. The 32-nodes-long bottleneck layer autoencoder achieved the worst image similarity with respect to the other two autoencoder variations. The 80-nodes-long bottleneck layer autoencoder achieved an MS-SSIM close to the one obtained by JPEG in our first round of experiments. This result suggests that a more accurate image reconstruction might need a higher code length.

These initial investigations suggest that there is a trade-off between compression ratio and image quality, in which the code length plays an important role. Higher compression ratios could be achieved with reduced code lengths, albeit yielding lower image quality. Conversely, higher image quality could be achieved with increased code lengths, with the side effect of worse compression ratios.

5 Discussion and Future Work

Our work aims to motivate further discussion on autoencoders as a standalone image compression format. To the best of our understanding, most work up to this point had focused on deep, powerful architectures, which have also proved to compete with JPEG in terms of compression ratio and image quality [6, 7, 2]. Nonetheless, the depth and size of such networks make it unfeasible to embed them as portable compressed image representations. We explored shallow autoencoders with manageable size that could be used as a standalone image compression format.

Our experiments suggest that RNN-based autoencoders could be effective architectures for yielding competitive compression: this had been already suggested by Toderici et al. in [7], but here we extend such assertions by claiming that even shallower autoencoder networks could benefit from LSTM+CNN layers. Similarly, we found that the code length in the autoencoder's bottleneck layer is a relevant parameter for modulating compression ratio and image quality. Hence, further investigations should be carried out for determining ideal code lengths depending on the desired compromise between compression ratio and image quality.

It is true that none of the results that we reported comes close to matching JPEG's performance. Nonetheless, the competitive MS-SSIM achieved by the LSTM+CNN autoencoder makes us believe that near-lossless image reconstruction could be feasible with a more fine-tuned architecture and increased training time.

We acknowledge some areas for improvement in our work. Firstly, we believe that more diverse images should be explored, as the reconstruction capabilities of our method might be dependent on the complexity/structure of the image's content. We also believe that our compression procedure should be given more training time, as 300 epochs was probably not a long-enough training period. Due to the time constraints, we could not do better in this regard, but an increased training time could help the autoencoders learn better representations of an image, possibly yielding better MS-SSIM values. We also think that the effects of padding should be an important area of focus. For the purposes of our experiments, the dimensions of our test images could be exactly spanned with the dimension of our chosen subimage-block size. Nonetheless, in a more realistic setting, some images might require padding for subimage-block decomposition to be possible, thus introducing some potential noise in the image reconstruction. Finally, we believe that the compression effects of meta-parameters should be explored in greater depth. In our work, we considered the effects of code length, but it would also be relevant to explore others such as the subimage-block size or the batch size during training time.

6 Conclusions

Shallow autoencoders are not yet a competitive image compression mechanism, compared with deeper neural network architectures [5, 1, 7] or more traditional algorithms like JPEG. Nonetheless, our work suggests that the same tricks that improved deeper autoencoder architectures (in particular, CNN+RNN layers) could be beneficial even for shallower architectures. We hypothesize this after evaluating the surprisingly positive results obtained with our shallow LSTM+CNN autoencoder, specially in terms of image quality preservation. These preliminary efforts could eventually lead to the development of a self-contained image compression format that utilizes neural networks, potentially for both lossy and near-lossless compression. We suggest that further work on shallow-autoencoder architecture discovery could benefit from neuroevolution approaches.

Acknowledgments

We would like to acknowledge Prof. Risto Miikkulainen for his invaluable guidance and discussions that shaped the path of this project. Special thanks to Angelica Fuentes for her assistance while debugging the graphics output of our models and curating the styling of our graphs. We would also like to thank Citatis and the US Forest Service for uploading helpful images to the web, which allowed us carry out our experiments.

References

[1] Zhengxue Cheng, Heming Sun, Masaru Takeuchi, and Jiro Katto. Deep convolutional autoencoder-based lossy image compression. *Picture Coding Symposium*, 2018.

[2] Michele Covell, Nick Johnston, David Minnen, Sung Jin Hwang, Joel Shor, Saurabh Singh, Damien Vincent, and George Toderici. Target-quality image compression with recurrent, convolutional neural networks. 2017.

[3] R. D. Dony and S. Haykin. Neural network approaches to image compression. *Proceedings of the IEEE*, 83:288–303, 1995.

[4] A. Namphol, S.H. Chin, and M. Arozullah. Image compression with a hierarchical neural network. *IEEE Transactions on Aerospace and Electronic Systems*, 32:326–338, 1996.

[5] Lucas Theis, Wenzhe Shi, Andrew Cunningham, and Ferenc Huszar. Lossy image compression with compressive autoencoders. *ICLR*, 2017.

[6] George Toderici, Sean M. O'Malley, Sung Jin Hwang, Damien Vincent, David Minnen, Shumeet Baluja, Michele Covell, and Rahul Sukthankar. Variable rate image compression with recurrent neural networks. *ICLR*, 2016.

[7] George Toderici, Damien Vincent, Nick Johnston, Sung Jin Hwang, David Minnen, Joel Shor, and Michele Covell. Full resolution image compression with recurrent neural networks. 2017.

[8] Z. Wang, E.P. Simoncelli, and A.C. Bovik. Multiscale structural similarity for image quality assessment. *The Thrity-Seventh Asilomar Conference on Signals, Systems Computers*, 2003.

[9] E. Watanabe and K. Mori. Lossy image compression using a modular structured neural network. *Neural Networks for Signal Processing XI*, 2001.

7 Appendix

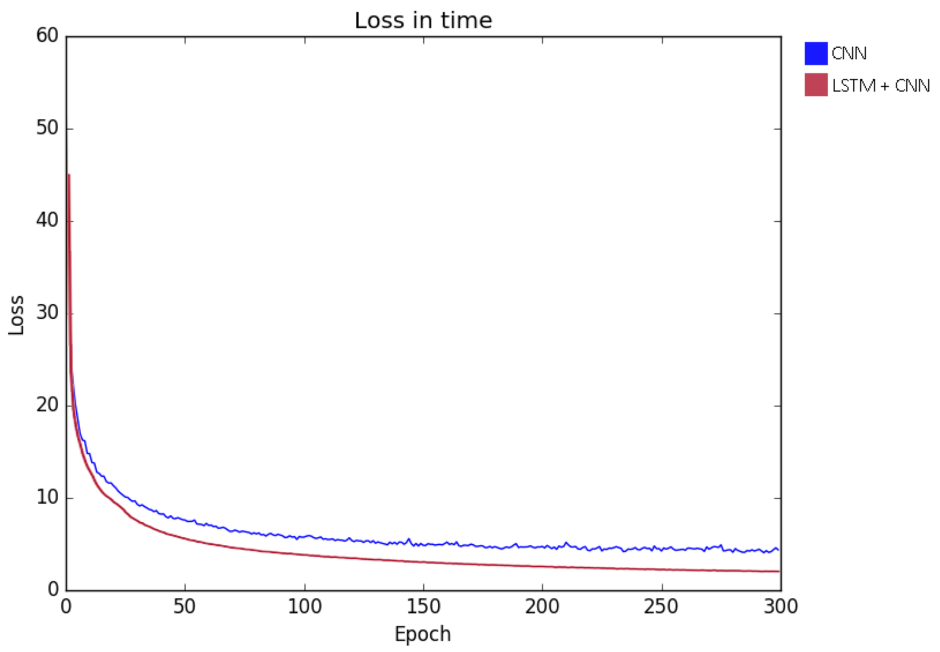


Figure 5: First round of experiments: loss in time during training with test image no. 1.

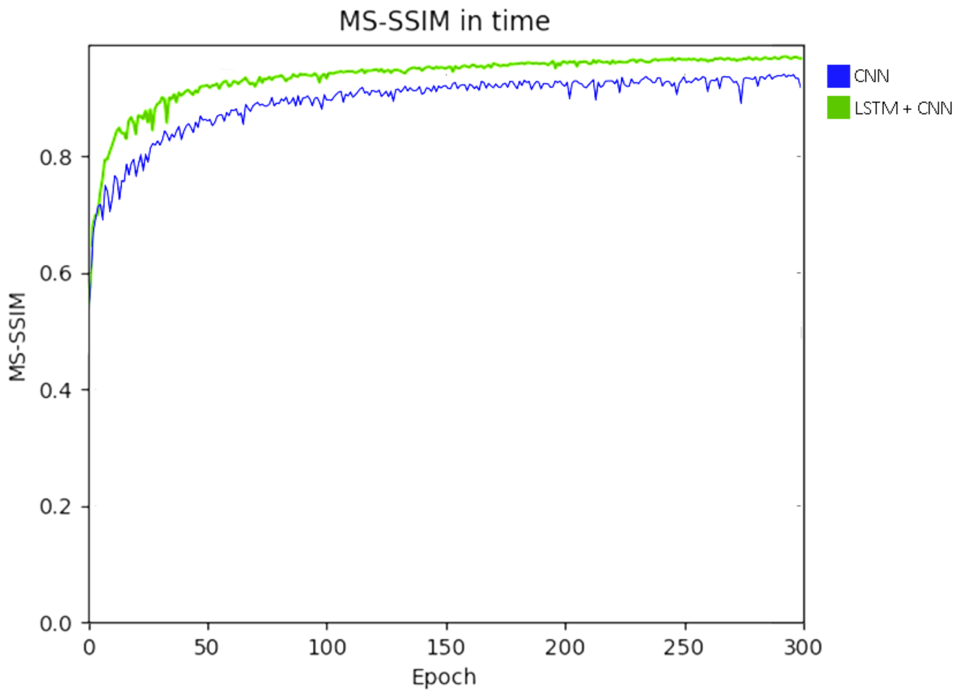


Figure 6: First round of experiments: MS-SSIM in time during training with test image no. 1.

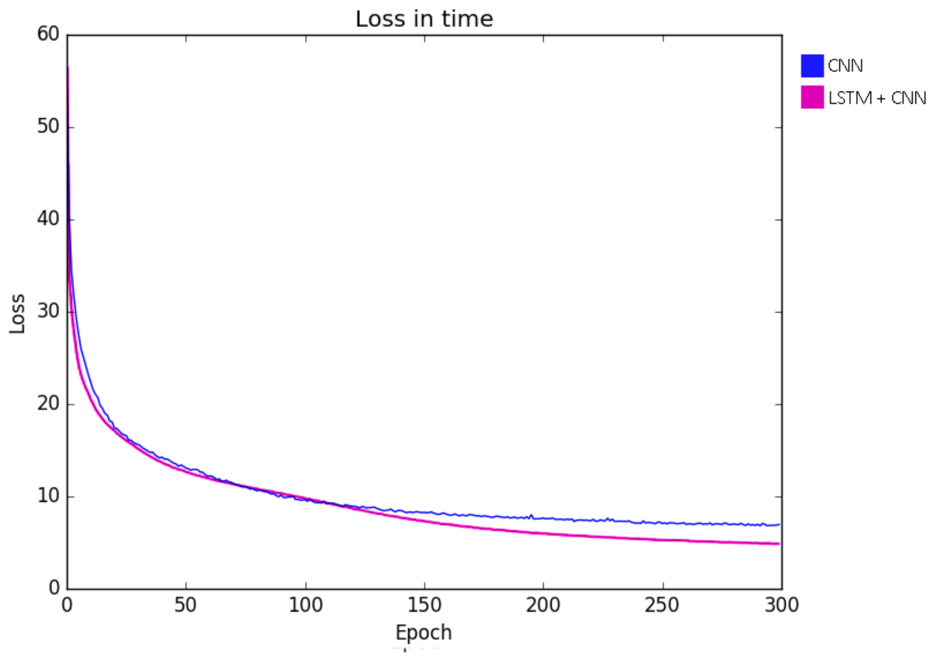


Figure 7: First round of experiments: loss in time during training with test image no. 2.

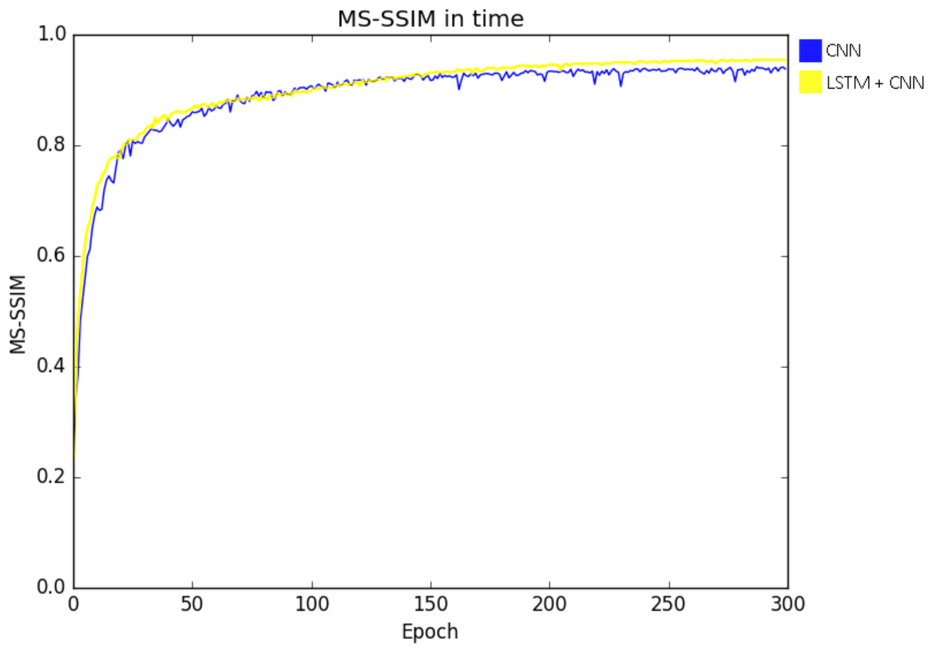


Figure 8: First round of experiments: MS-SSIM in time during training with test image no. 2.

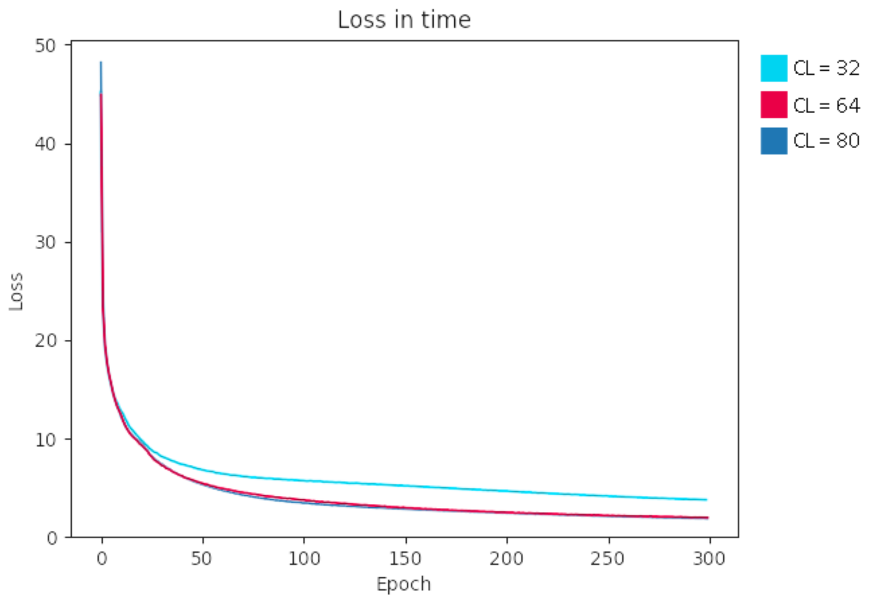


Figure 9: Second round of experiments: loss in time during training across different code lengths.

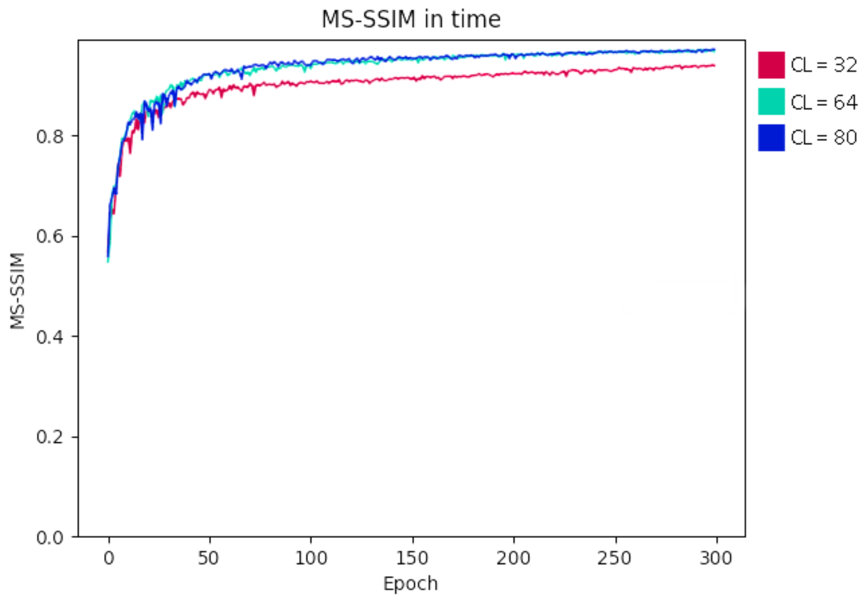


Figure 10: Second round of experiments: MS-SSIM in time during training across different code lengths.