Image Compression Using Deep Learning

H. Kubra Cilingir, Sivaramakrishnan Sankarapandian, M. Ozan Tezcan
kubra@bu.edu, sivark@bu.edu,mtezcan@bu.edu

Figure 1. (left) our best performing algorithm (CNN-AE-FT), (right) original image

1. Task

Image compression has an important role in data transfer and storage, especially due to the data explosion that is increasing significantly faster than Moore's Law.[1] It is a challenging task since there are highly complex unknown correlations between the pixels, as a result, it is hard to find and recover them. We want to find a well-compressed representation for images and, design and test networks that are able to recover it successfully in a lossless or lossy way.

2. Related Work

Several neural networks and deep learning methods have been utilized for the purpose of image compression. [2] summarizes most of the neural network approaches for image compression that are proposed before 1999. However, in those years there were no efficient deep learning algorithms. Toderici et al. proposed one of the most successful deep learning approaches for this aim [3]. They used a recurrent neural network (RNN) architecture and achieved slightly better performance than JPEG. They addressed the problem of JPEG compression for small images where the amount of redundant information is small. This network follows the classical three stages process of compression: encoding, quantization and decoding. Encoding and decoding are done iteratively using two different architecture of RNNs, Long Short Term Memory (LSTM) and convolutional/ deconvolutional LSTMs. The advantage of this method is that the compression ratio can be increased progressively while the disadvantage is that the method is time consuming (as encoding and decoding are done iteratively). Very recently, a group implemented a compression network by using GANs in which they achieved visually more pleasing pictures at high compression rates [4]. However, their reconstructed images are different than the original ones in the sense that they are commonly perceived as a different person or object.

Theis et al. proposed compressive autoencoders which uses a technique of entropy coding involving assigning bit representation to values in images based on how frequent a value appear in an image [5]. Quantization (rounding to the nearest integer) step before encoding makes the derivative of rounding function undefined,

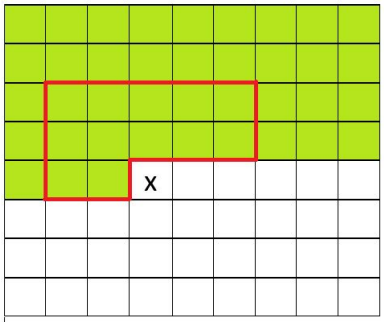

Figure 2. Predictive coding for lossless image compression. Green pixels shows the ones that we already know, and the red box is prediction region. Our aim is to predict the value of x using the pixels in the red box

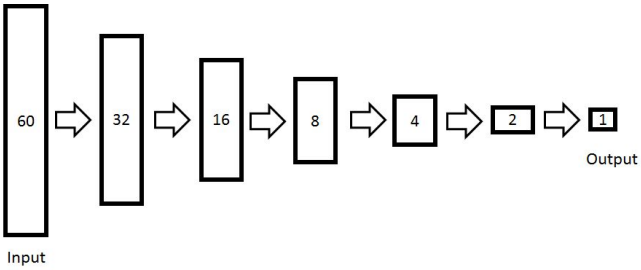

Figure 3. Layers of MLP. We used ReLU functions after every layer.

which is addressed by a smooth approximation only in the backward pass. Convolutional autoencoders are used for the implementation with performance metrics peak signal to noise ratio (pSNR) and structural similarity index metric (SSIM).

We applied several deep learning methods on the image compression problem. For the lossless image compression we used predictive coding via multilayer perceptron (MLP) and for the lossy compression we used autoencoders and GANs. Our results are better than JPEG and very close to JPEG-2000.

3.Lossless Image Compression

Lossless image compression is important for the areas which require precise imaging such as medicine and astronomy. In this project, we investigated MLP networks to apply predictive coding.

3.1. Predictive Coding with MLP

Most of the existing lossless image compression algorithms like JPEG-LS are based on predictive coding (PC). PC tries to estimate the new pixel value in the image assuming that some neighboring pixels are known. In this project we are reading the image in raster scan (first row, second row ...).

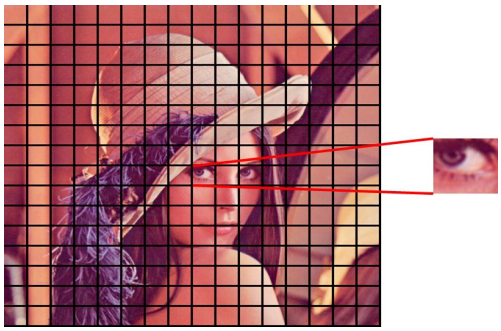

Figure 4. Illustration for block based methods

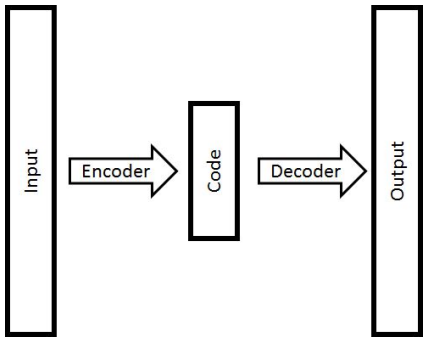

Figure 5. Autoencoder illustration

So we can assume that green colored pixels in Figure 2 are known and we are estimating the value of ‘x’. After the estimation step, we need to store the estimation error in that pixel for guaranteeing lossless compression. As a result, PC codes the whole image pixel by pixel and we only need to store some initial pixel values to start algorithm and the error image. Error image will have the same dimensions as the real image, however it will have a lower entropy in terms of information theory. So, using variable length coding (VLC) methods we can decrease the size of the error image. For VLC, we used Huffman coding.

MLP based networks have been used for PC [2], but after the publication of that paper, different extensions are proven be to successful in MLP networks (e.g. rectified linear units (ReLU), deeper networks). So, we tried to implement a MLP network for PC using state-of-the-art techniques. Figure 3 shows our final network. After every layer we used ReLUs to introduce nonlinearity. Note that, this is a regression network. At the end, we are quantizing the prediction to the nearest integer value. With this algorithm, we are creating image specific weights but since our layer size are small, we can easily store the network weights without creating a significant increase in the file size.

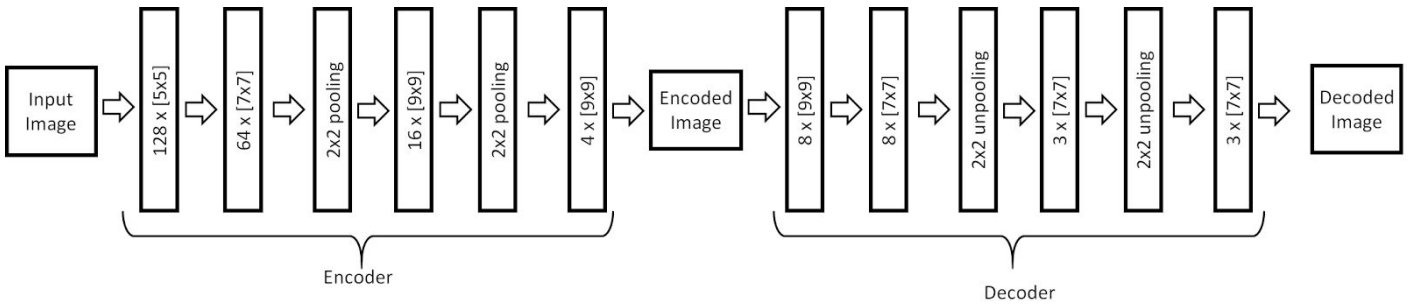

Figure 6. Best performed convolutional autoencoder. Pooling layers represent max pooling operations and unpooling layers represents bilinear interpolation.Here we quantized the encoded image using 6 bits instead of 8 to balance the increase in the images color dimension from 3 to 4

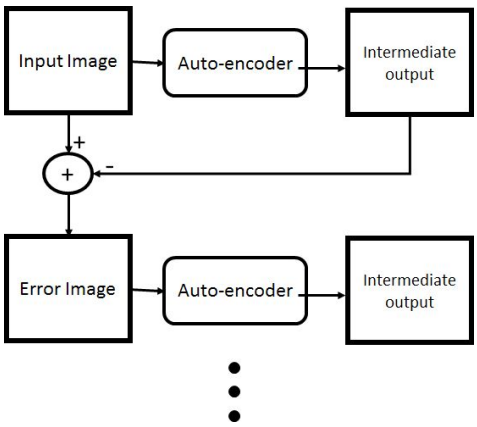

Figure 7. Recurrent CNN-AE for introducing a trade-off between image quality and compression ratio

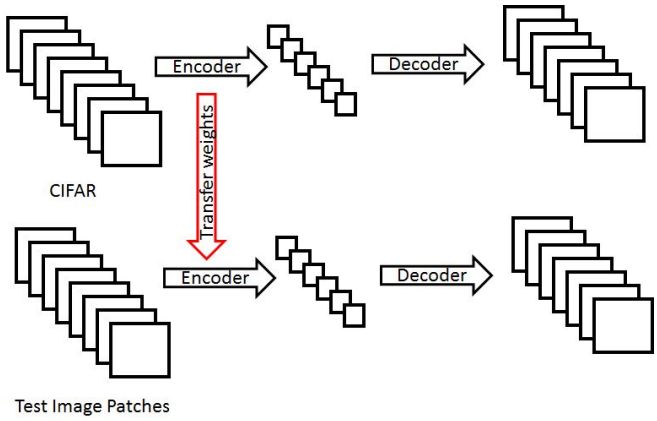

Figure 8. Fine-Tuning approach used for images specific compression

4.Lossy Image Compression

Human eye is not capable of sensing the small differences in pixel values, so people introduced several image compression algorithms which try to compress the images further with minimal information loss. Since human eye is less sensitive to local information in images, most state-of-the-art compression algorithms use block based methods. A simple illustration about image patching can be found in Figure 4. In this project, we also tried block based compression (using 32x32 blocks) methods using several autoencoder and GAN architecture for lossy compression.

4.1. Compression using autoencoders

Autoencoders (AE) are the networks which tries to represent the image using less information than the original signal. This process can be achieved using a dimensionality reduction, but we want to make sure that we can recover the original signals back from the reduced dimension with a small error. Thus, AE includes two networks; one is for encoding and the other is for decoding. The encoded signal is also called latent representation of the original signal The general scheme of the autoencoders can be found in Figure 5. We started by investigating MLP based autoencoders for the lossy compression by using the vectorized images as the inputs of MLP based autoencoders, but could not get any good results. This was mainly because of the missing spatial information in the vector representation of images.

Then, we tried fully convolutional autoencoders (CNN-AE). Here, the main aim is to decrease the dimensions of the image using max pooling operations. The tricky part is the decoder network. We need to use deconvolution operator for that part, but convolution operator is not injective so the inverse operator does not exist. However, Long et al. introduced an approximation for deconvolutional neural networks [6]. We followed their idea and used interpolation function followed by convolution operator for deconvolution. Figure 6 shows

the general layout of our convolutional autoencoder with the best parameters according our experiments.

For the autoencoder architectures, we used CNN-AE which defined above as the baseline method and tried two more improvements. First one is based on a very simple idea mentioned in [7] which is adding a recurrence relation to the network. Since our aim is lossy compression we are introducing some error in the output. For decreasing this error, we tried to use a recurrent convolutional autoencoder (CNN-RNN-AE) which treads the error after each step as the new image and applies CNN-AE again to compress the error. This idea is illustrated in Figure 7. This approach introduces a trade-off between image quality and the compression ratio.

We trained all of networks on a database consisting of 32x32 images and used block based compression in the test images. We realized that our problem has a difference from the classical machine learning problems which aims to find the unknown output given some input. Here both the input and output is the same image which is known from the beginning. So, we may be able to use some image specific information while compressing it. One simple idea is to fine tune the whole network using the patches of the current image. However that will give image specific network weights which we have to store with the image. Since deep neural networks include too many weights, this approach does not seem feasible. Instead, we decided to use a very narrow decoder design and fine tune only the decoder for the given image. This way, we can still use the image specific information without increasing the file size too much. This idea is illustrated in Figure 8.

4.2. Compression using GAN:

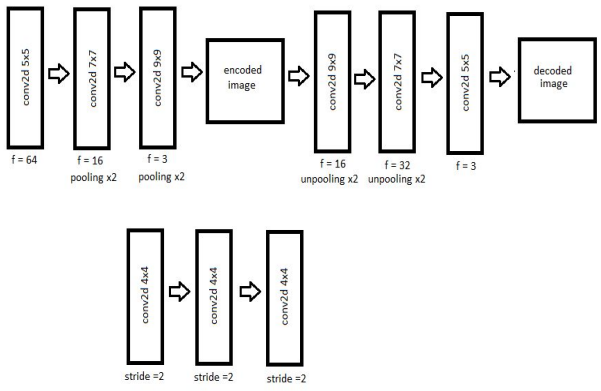

Figure 9. Generator of GANs being using as a decoder

Having considered compressed image as a different representation of the original image in the latent space, we came up with another question. How can we better construct the latent representations to capture more valuable information? In our case, this problem corresponds to designing a better loss function to map and inverse map the images. GANs [8] are appealing for this task since they are known to produce realistic looking images. However, at the same time, we need to keep the sufficient information in order not to end up with a realistic but a different image.

The generator part of the GANs can serve as a decoder for the compression. Therefore, we first picked our latent representations as the downsampled version of the image. We combined the mean squared error together with the adversarial loss to do reconstruction. The formulation of the loss function chosen for the first GAN compression network is as follows:

$$L_{total}(x, x') = \lambda_1 L_{error}(x, x') + \lambda_2 L_{adversarial}(x, x', w, \theta)$$

where w and θ are the weights of the discriminator and generator respectively. We picked Wasserstein loss (WGAN loss) function as our adversarial loss [9]. The classical GAN loss suffers from instability since it minimizes Jensen-Shannon distance which does not give a continuous loss function when the probability densities are in low dimensional manifold, which is our case. Wasserstein loss gives a more stable and converging error function which makes the interpretation easier, and have a potential to provide better results. In terms of the application, we removed sigmoid layer, eliminated the logarithm from the loss function, and clipped the weights. For more detail, the reader is encouraged to read [9] . The first term for the loss function is chosen as L_2 loss to achieve similarity between the original and reconstructed images. For a comparison, we also tried Goodfellow's suggested GAN loss as the first term [10].

The second approach is to replace the downsampled images with an encoder output to get the compressed representation. Adding an encoder - decoder scheme gives the system the flexibility to decide on its own latent representations. The training can be done simultaneously or successively between the auto-encoder and the GAN structure. We implemented the simultaneous training. We evaluated the network with classical GAN or WGAN loss as the first terms and L1 or L2 norm as the second terms. We validated the network to find a good weighted balance between these two terms.

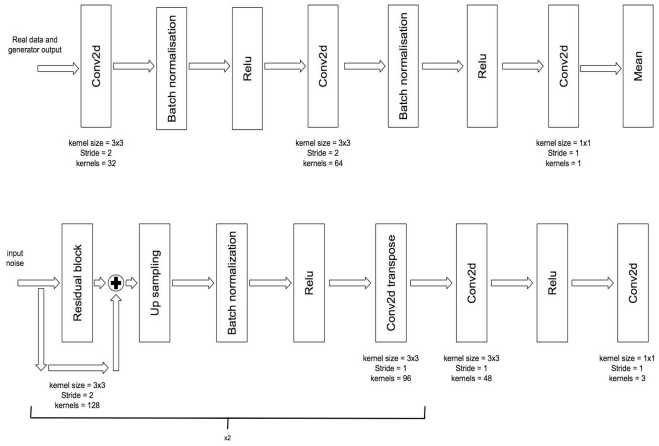

Figure 10. Compression achieved using combination of both auto encoder and GAN

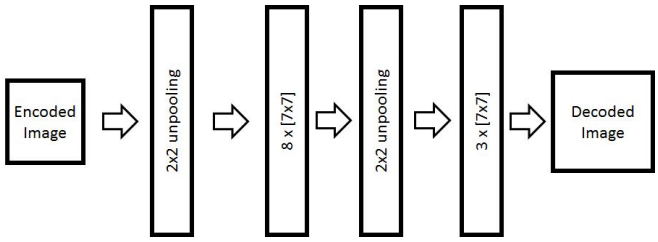

Figure 11. Decoder network for CNN-AE-FT. Encoder part is same with CNN-AE

5. Dataset and Metric

Since we developed an image specific algorithm for lossless compression, we used three test images for both training and testing.Those images can be found in our Github repository.For the comparison metric of lossless compression, we used bits per pixel (bpp).

For all training phase s of lossy compression we used CIFAR dataset which consists of 50,000 32x32 RGB images [11]. We chose this dataset for the following three reasons: (i) it has many samples, (ii) since it consist of small images, it does not require days to train a network on CIFAR, and (iii) it is reasonable use patch based compression on high resolution (HR) images to protect local relationships in the pixel values. We also used some HR images to test our methods real performances. For the HR images we trained our algorithms on CIFAR dataset and applied the resulting weights to HR image patches.

We used two different metrics for comparisons. The first one is peak signal to noise ratio (pSNR) which can be formulated as

$$PSNR = 20\,log10\left[\frac{255}{\sqrt{MSE}}\right]$$

Where,

$$MSE = \frac{1}{mn}\sum_{i=0}^{m-1}\sum_{j=0}^{n-1}[I(i,j) - I'(i,j)]^2$$

I is the input image and I' is the reconstructed image after applying encoder and decoder. Note that, we want bigger pSNR values to show that our algorithm performs better. However Wang et al. showed that pSNR is not a good comparison metric for images since this not take HVS into account [12]. They proposed a new metric called structural similarity index (SSIM). It can be formulated as

$$SSIM(I,I') = \frac{(2\mu_I\mu_{I'} + (k_1L)^2)(2\sigma_{I'} + (k_2L)^2)}{(\mu_I{}^2 + \mu_{I'}{}^2 + (k_1L)^2)(\sigma_I^2 + \sigma_{I'}^2 + (k_2L)^2)}$$

Where μ_X is the mean value of X, σ_X is the standard deviation of X, σ_{XY} is the covariance of X and Y, L is the dynamic range of pixel values (255 for 8 bit unsigned integers), k_1=0.01, and k_2=0.03 by default. SSIM is calculated on small image windows and then averaged over all image. It results with a decimal value between -1 and 1. SSIM=1 means the images are exactly the same, so we want it to be as close as possible to 1.

6. Experiments

6.1 MLP

For the lossless compression we used the model shown in Figure 3 and implemented Huffman coding to code the error images. One of the main parameters in predictive coding is the size of the prediction block. When we increase it too much the pixels became uncorrelated with the next pixel and for small box sizes it will become harder to estimate the new pixel value and use a deep network. After some tests we decided to use 60 pixels as the prediction size. For the loss function, we used a simple MSE loss.

6.2 Autoencoders:

Our successful autoencoders are all fully convolutional models. For consistency and easy comparison with JPEG, we always used 16:1 compression ratio in our experiments. The most important part about the autoencoders is that we cannot use too many number of filters in the last layer before the latent space since we will store this representation as the compressed image. This makes it hard to develop symmetric autoencoders so we used asymmetric ones. Since our input images are 32x32, very deep networks are not performing so well. After doing some experiments, we

found the network depicted in Figure 6 to be the best one for CNN-AE and CNN-RNN-AE. In CNN-RNN-AE we used 3 recursive steps. We can use the same architecture for also CNN-AE-FT, but it has ~1300 weights of 32 bit floating numbers in the decoder and it will be inefficient to store them with the compressed image. Thus we decided to use a shallower decoder for fine tuning approach. The network used for fine tuning is shown in Figure 11. In all AE variations we used Adam optimizer, with a learning rate of 10^{-5} and MSE ass the loss function.

6.3 GANs

The first GAN network has two parts: a decoder and a discriminator network. We inspired the decoder network from Resnet [13], however we chose to make it shallower since our images have sizes 32x32 and we desired a shorter training time due to the time constraints and to control the parameters more easily. The decoder (generator) consists of residual, upsampling, batch normalization layer, relu, conv2d and conv2d transpose blocks. Fig 9 shows the exact order of the layers and the details of the parameters of the layers. Convolutional layers in the decoder part have paddings and with stride size 1, so that they do not change the image size, we achieve the original size by upsampling twice. Discriminator only consists of convolutional layers Figure 9. We used xavier initialization [14] for convolutional layers to decrease the saturation of the activation units.

For Wasserstein loss function we took learning rate as 2.10^{-5}, used RMSProp, and clipped weights between [A-0.01, 0.01] interval as suggested in the paper [9]. $\lambda_1 = 1$ and $\lambda_2 = 10$ give the best SSIM results. When we increase λ_2, MSE error decreased as expected since that term directly minimizes MSE. For DCGAN loss, we used 2.10^{-4} as learning rate and ADAM optimizer with beta = 0.5.

The second GAN network consists of an auto-encoder and a discriminator. We used the same learning rate and coefficients for all of the loss functions. Encoder and decoder are convolutional networks with upscaling and max pooling layers to control the size of the image. The details of the layers and parameters are given in Figure 10.

7. Results and Comparison:

Even if there are variety of different compression techniques which claim they are better than JPEG, most of them does not give implementation details and also JPEG is still to most widely used compression algorithm for both lossy and lossless compression. Thus, in this paper, we will compare our results with JPEG and JPEG-2000.

Table 1 shows the comparison of bpp between our network, JPEG and JPEG-2000. So our network achieves far better results than JPEG and it is very close to JPEG-2000. Also note that this MLP network can run on real time on the current personal computers and mobile phones.

Table 1. Comparison of average bpp rates for different lossless compression algorithms on the test images

Algorithm	**Bits per pixel**
JPEG	5.2
JPEG-2000	4.3
MLP	4.5

For the lossy compression, we compared our results on both CIFAR test data and HR test images using patch based compression. For our best performing CNN-AE and GAN networks, we showed the results in Tables 2 and 3. All results are recorded for 16:1 compression ratio. We want to inform the reader that this comparisons are not so fair since we did not use any entropy coding in our algorithms.

On the CIFAR test data, all of our algorithms except GAN-AE with L1+DC loss beat JPEG and our best performing network is GAN-AE with L2+Wloss. One can observe that WGAN performs better than DCGAN and also adding L2 loss boosted the performance. We could not achieve a good performance with L1 loss. Moreover, we observed that as we give more weight to the similarity loss (L2), the pictures started to resemble more but images become more blurry. As we increase the weight of the adversarial loss, the outputs became more sharp and pleasing, but we encountered some problems related to the color. The network required many iterations to get the same color with the original image, and sometimes it degraded. But after finding good parameters for the networks, these problems are minimized and we achieved a sharper and a good reconstructed image. CNN-AE and CNN-RNN-AE also achieved very good performances.

In the CIFAR data, none of our algorithms are in a comparable level with JPEG-2000. However, when using patch based comparison algorithms on HR image set, we can get comparable results with

JPEG-2000. Results on HR images can be found in Table 3. Again, GAN-AE showed a great success. Here we only reported the best performed GAN-AE network which uses L2 loss and wasserstein metric. According SSIM, CNN-AE-FT seems to be very successful with an index very close JPEG-2000 and beat GAN-AE network. It shows that image specific compression using fine tuning shows a great promise. Also, note the performance increase in JPEG from CIFAR images to HR images. It seems that JPEG is much more powerful for the HR images and it is more meaningful to make JPEG comparisons on HR images.

Table 2. Results of various architecture on CIFAR dataset

CIFAR	
Method	**pSNR**
JPEG-2000	43.6
GAN-AE(L2, W)	36.98
GAN-AE(L2, DC)	31.54
CNN-RNN-AE	31.4
GAN WGAN	30.96
CNN-AE	30.6
GAN - DC	30.45
GAN-AE(L1, W)	30.04
JPEG	28.24
GAN-AE(L1,DC)	28.23

8. Future Work:

We have presented different networks to compress low and high resolution images. We have our results based on the network structures and parameters we tried. Although most of our results are promising, they raises many directions and modifications that can be applied. In our project, we focused on the lossy compression problem and tried very simple architecture for the lossless compression. We trained MLP network using MSE loss. Using an entropy loss instead of MLP loss can leverage the performance, but this is not a simple task since calculating the entropy requires the entropy of the error image. For estimating the entropy we can use density estimation methods such as Parzen windowing as defined in [15].

Table 3. Results of various architecture on HR dataset

HR Images		
Method	**pSNR**	**SSIM**
JPEG-2000	36.1	0.993
CNN-AE-FT	33.9	0.99
GAN-AE(best)	32.53	0.987
JPEG	33.2	0.986
CNN-RNN-AE	30.2	0.972
CNN-AE	28.9	0.968

We used two elements as the loss function for GAN networks, the first one is for the similarity between the original and the reconstructed image and the other one is the adversarial loss to make images more sharp and realistic. We can explore other metrics instead of the Euclidean distance as a similarity measurement. In addition, a third component can be added to minimize the Euclidean error not in the original space, but in some feature space. The feature space can be constructed by a known network such as VGG [16] or Alexnet [17]. This method is used in superresolution tasks to get superior images [18], it can be easily applied to our approach. In addition, entropy of the compressed representations can be added to the loss function to decrease the amount of the redundant information we stored. We can use a smooth approximation of the entropy function since the derivative is zero except at the integers. This approximation will allow us to backpropagate the loss [19].

Another improvement can be made by taking the advantage of the knowledge of the original images. For instance, high frequency elements make the image sharp, they can be fed to the system as a condition or directly to decrease the loss of this information. Alternative sources can be used to keep more information, such as the representation in a feature space. Moreover, we can condition on the classes and compress based on the class information which may lead to better results.

Fine tuning approach can be used in video compression since we will be storing weights specific

to a single image. In a video, images can share the decoder weights for the consecutive collection of the frames, which makes the storage of the decoder weights negligible. We can also apply fine-tuning for GANs.

In order to avoid the square patterns at the output image for high-resolution inputs; we can either use o overlapping patches to compensate the error on the boundaries, or we can do some image processing to eliminate them. Another option is to make the network bigger and deeper and compress the whole HR image in a lump rather than dividing it into patches.

Lastly, we used deterministic autoencoder in all of our networks to encode and decode the images. Although we believe that deterministic autoencoders are more appropriate for our scenario, there are some works on variational autoencoders due to their generative power [20, 21]. They can be explored to integrate our framework.

9. Detailed Roles

Task	File Names	Who
Implemented and test MLP for lossless compression	MLP_lossless/*	Ozan
Implemented entropy coding for comparisons	MLP_lossless/*	Ozan
Implemented and test autoencoders	Autoencoders/*	Ozan-Siva
Implemented and test GANs	GAN/* GAN-AE/*	Kubra
Implemented SSIM for comparisons	SSIM.ipynb	Siva
Prepared report and presentation	Report and Presentation	all

10. Code repository

https://github.com/scelesticsiva/Neural-Networks-for-Image-Compression

11.References

1) Ranganathan, Parthasarathy. "The data explosion." (2011): 39-48.
2) Jiang, J. "Image compression with neural networks–a survey." Signal Processing: Image Communication 14.9 (1999): 737-760.
3) Toderici, George, et al. "Full Resolution Image Compression with Recurrent Neural Networks." arXiv preprint arXiv:1608.05148 (2016).
4) Santurkar, Shibani, David Budden, and Nir Shavit. "Generative compression." *arXiv preprint arXiv:1703.01467* (2017).
5) Lucas Theis, et al. “Lossy image compression wit compressive autoencoders” (2017) submitted to ICLR.
6) Shirsat, T. G., and V. K. Bairagi. "Lossless medical image compression by integer wavelet and predictive coding." *ISRN Biomedical Engineering* 2013 (2013).
7) Toderici, George, et al. "Full Resolution Image Compression with Recurrent Neural Networks." *arXiv preprint arXiv:1608.05148* (2016).
8) Goodfellow, Ian, et al. "Generative adversarial nets." *Advances in neural information processing systems*. 2014.http://ieeexplore.ieee.org/document/1284395/?arnumber=1284395&tag=1
9) Arjovsky, Martin, Soumith Chintala, and Léon Bottou. "Wasserstein gan." *arXiv preprint arXiv:1701.07875* (2017).
10) Goodfellow, Ian. "NIPS 2016 Tutorial: Generative Adversarial Networks." *arXiv preprint arXiv:1701.00160* (2016).
11) Krizhevsky, Alex, and Geoffrey Hinton. "Learning multiple layers of features from tiny images." (2009).
12) Wang, Zhou, et al. "Image quality assessment: from error visibility to structural similarity." *IEEE transactions on image processing* 13.4 (2004): 600-612.
13) He, Kaiming, et al. "Deep residual learning for image recognition." *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2016.
14) Glorot, Xavier, and Yoshua Bengio. "Understanding the difficulty of training deep feedforward neural networks." *Aistats*. Vol. 9. 2010.
15) Erdogmus, Deniz, and Jose C. Principe. "Entropy minimization algorithm for multilayer perceptrons." *Neural Networks, 2001. Proceedings. IJCNN'01. International Joint Conference on*. Vol. 4. IEEE, 2001.
16) Simonyan, Karen, and Andrew Zisserman. "Very deep convolutional networks for large-scale image recognition." *arXiv preprint arXiv:1409.1556* (2014).
17) Krizhevsky, Alex, Ilya Sutskever, and Geoffrey E. Hinton. "Imagenet classification with deep convolutional neural networks." *Advances in neural information processing systems*. 2012.
18) Ledig, Christian, et al. "Photo-realistic single image super-resolution using a generative adversarial network." *arXiv preprint arXiv:1609.04802* (2016).
19) Theis, Lucas, et al. "Lossy image compression with compressive autoencoders." *arXiv preprint arXiv:1703.00395* (2017).
20) Santurkar, Shibani, David Budden, and Nir Shavit. "Generative compression." *arXiv preprint arXiv:1703.01467* (2017).

21) Gregor, Karol, et al. "Towards conceptual compression." *Advances In Neural Information Processing Systems*. 2016.

12.Appendix

Figure 12. (left) our best performed algorithm on CIFAR (GAN_AE with WGAN and L2 loss), (right) original CIFAR images

Figure 13. Original image

Figure 14. Image compressed with JPEG

Figure 15. Image compressed by CNN-AE

Figure 16. Image compressed by RNN-CNN-AE

Figure 17. Image compressed by CNN-AE-FT

Figure 18. Image compressed by WGAN L2

Figure 19. Image compressed by DCGAN L2