Smaller is Better: Deep Neural Networks for Image Compression
(CS231n Final Report)

Amita Kamath
kamatha@stanford.edu

Aarti Bagul
aartib@stanford.edu

Suvadip Paul
suvadip@stanford.edu

Abstract

Image compression is a task of increasing importance, due to the pervasive nature of images in the applications we use today along with the need to transmit these images in an inexpensive way while maintaining reasonable quality. We investigate the problem of image compression, experimenting with several different models. Our baselines are a Convolutional Autoencoder, and a Convolutional Variational Bayes Autoencoder. We then experiment with a CNN encoder with a GAN decoder, an RNN model, an RNN model with modified loss, and GANs. Each of the models have been analyzed in depth. For the dataset we have picked, we find that RNNs model is the most flexible in variable compression rates but the best compression in terms of PSNR and MS-SSIM are from GANS and AEs. We also show why this problem needs to be explored more and is a non-trivial task with the need for dedicated modelling strategies to tackle them. Finally, we have benchmarked different categories of best performing models against each other, a first such comprehensive review of its kind in this domain.

1. Introduction

In this paper, we investigate the problem of image compression, which involves reducing the size (in bytes) of an image file without a significant loss in the quality of the image.

The amount of data generated and transmitted over the internet has increased dramatically over the past ten years and continues to rise every year - by the year 2020, it is hypothesized that about 1.7 megabytes of new information will be created every second for every human being in the planet [13]. In addition, with the rise of mobile computing, the need for fast and inexpensive image transmission has increased even further. As such, image compression has become an issue of increasing importance in order to decrease the costs of transmitting this data while keeping the quality as high as possible. Image compression is also ubiquitous - YouTube videos, social media images, even pictures on your computer have been compressed without you even realizing it, as a part of a seamless experience from data creation to data viewing.

In essence, developing image compression algorithms is crucial to provide end-users with high quality images at low costs (in terms of latency as well as bandwidth). Image compression will help reduce the file size to allow more images to be stored in a given amount of disk space. It will also reduce the time required to transfer the images over the internet and the time taken to render/download these images, which makes for a better user experience [14].

We see that hand-crafted image features used to training machine learning models in the past have now become obsolete: which is also going to become true for current state-of-the-art compression techniques. With the success of Deep Learning for vision, image compression is one of those tasks which seems like a natural progression from classification, as most models seem to learn hidden representations. Neural networks have shown promising results in this scope, beating conventional compression methods.

In this project, we aim to explore and build deep end-to-end neural network techniques for image compression which achieve comparable loss compression rates to the current state of the art (in terms of the family of pixel transforms used in JPEG). We also show that going from a typical classification task to a compression task is non-trivial (except for the use of convolutions) and a lot is left to be explored in this field.

To be concise, the input to our model is an image. We use a model (CNN/VAE/CNN-GAN/RNN/GAN) to output a compressed version of that image, i.e. occupying less space on disk for the same image size as the input. The type of compression is lossy image compression, like JPEG.

We have experimented with several models: our initial two baselines are a CNN autoencoder and a Convolutional Variational Bayes autoencoder. We then moved on to an architecture with a CNN encoder and a GAN decoder, with different types of loss functions for training the GAN. Finally, we explored an RNN-based architecture, along with our interpretation of an improvement to the model’s loss function.

The structure of this report is as follows. Section 2 dis-
discusses related work in the field of learned image compression. Section 3 describes the dataset. Section 4 details the evaluation metric, then each of the methods used, experiments run and results obtained. Section 5 concludes the paper and discusses scope for future work.

2. Related Work

Autoencoders were first thought of as the default image compression neural networks due to their ability to reduce dimensions [15], extract complex visual representations, convert images to compressed binary formats [16] and other applications [17]. More recently, variational autoencoders have been gaining popularity to capture hidden representations directly due to its reparameterization trick and generative nature [12].

Over recent years, the research for compression using deep learning has gone beyond autoencoders. Toderici et al [1] implement variable-rate image compression - prior to this work, neural networks required a fixed compression rate. They present a recurrent neural architecture based on convolutional LSTMS (convLSTM) that compresses 32x32 sized images, obtaining better SSIM values than JPEG.

However, the network was restricted to 32x32 sized images - the authors subsequently presented [2], in which images of varied sizes could be used - provided they were multiples of 32x32. This work also obtained results superior to JPEG, with respect to both PSNR and MS-SSIM. We experimented with this architecture, and it has been discussed in greater detail in section 4.4. Johnston et al [4] expanded on this work by incorporating hidden-state priming, spatially adaptive bit rates, and SSIM-weighted loss, obtaining improved results. We experimented with this architecture in Section 4.5. This architecture obtains the current state of the art, with an SSIM-AUC of 21.02 and a PSNR AUC of 61.40.

[3] trains deep autoencoders for image compression, outperforming the RNN-based approaches and made computationally efficient with the help of a sub-pixel architecture. [4] implements image compression using a recurrent autoencoder, with support for spatially adaptive bit rates and a loss function built on structural similarity (SSIM). [5] presents an image compression method based on nonlinear transform coding, optimized end-to-end for rate-distortion performance. [6] implements image compression using GANs, obtaining significantly lower bitrates than the previous state of the art. This approach thus achieves the current state-of-the-art “visually”, but only for extremely compressed images.

More detailed motivations behind the papers above along with their architectures have been explained below in the modelling section.

3. Data

We obtained our dataset from the Challenge on Learned Image Compression [7], released by the Computer Vision Lab of ETH Zurich. The data consists of “professional” and “mobile” datasets, each split into training, validation and test sets. The “professional” dataset has high quality photos taken by experiences photographers whereas the “mobile” dataset contains poor quality photos taken on cellphones.

We decided to work on the “professional” dataset for this project. It contains 585 images in the train set, 41 in the validation set and 118 in the test set. The images in the dataset are of varying sizes and have varied content. The total size of the “professional” dataset is about 3GB. This is because the images are very high in resolution often larger than 1000x1000 making this task definitely a non trivial one.

![Figure 1. Examples of images in the dataset. The images are of different sizes and have varied content.](image)

Each of our models had different requirements in terms of original size of the image. The CNN encoder required fixed-size inputs, because it contained a fully connected layer. The RNN model as defined in [1] required 32x32 sized images. We did not implement this model, and instead implemented its successor, [2], which requires image sizes that are multiples of 32. The other models work on all image sizes. In order to have a fair basis to compare all of these models, we reduced the image size of all of our images to 32x32.

For the RNN models, we also incorporated random rotation and flipping. This is known to artificially inflate the dataset and thus improve performance of models given the same dataset - it also forces the model to generalize better.

In terms of features, we used end-to-end models such as VAE, RNNs and GANs - as such, these models extracted features themselves and did not need to be given features. We discuss these models in more depth in the report.

4. Methods, Experiments and Results

Evaluation Metric

Following existing research in the field of image compression, as well as the image compression challenge from which we obtained our data, we plan to evaluate our results...
in terms of PSNR (Peak Signal to Noise Ratio) as well as Structural Similarity (SSIM).

PSNR is the ratio between the maximum possible power of a signal and the power of corrupting noise [9]. It’s most easily defined in terms of mean squared error (MSE), which is the cumulative squared error between the compressed and the original image [10].

Given an image \( I(x, y) \) of size \((M \times N)\) and the compressed version \( I'(x, y) \), MSE and PSNR are defined in the equations below [10]

\[
\text{MSE} = \frac{1}{MN} \sum_{y=1}^{M} \sum_{x=1}^{N} [I(x, y) - I'(x, y)]^2 \\
\text{PSNR} = 20 \times \log_{10} \frac{255}{\sqrt{\text{MSE}}}
\]

A lower value for MSE (and hence a higher value for PSNR) means less error between the original and compressed image [10].

SSIM is another metric for measuring the similarity between two images. MSE and PSNR give an estimate of the absolute error whereas SSIM considers the perceived change in structural information while incorporating perceptual phenomena such as luminance masking and contrast masking [11]. To calculate the SSIM between two images, we used the \text{compare}\text{\_ssim} function provided in the scikit-image library. SSIM takes values between 0.0 and 1.0 (where a higher value indicates higher similarity between images).

For the models that a produce fixed size encoding (the autoencoders), we also include the compression rate and the space saving (defined below) as metrics. There is a natural trade-off between reducing the size of the image and maintaining the quality of the image. To take this to an extreme, calling the input image as the “compressed” version will have a perfect SSIM and PSNR score. However, this is practically useless because the image occupies the same amount of space. As we progressively reduce the size of the image, the image quality will progressively drop. The two metrics defined below capture this aspect of image compression.

\[
\text{compression ratio} = \frac{\text{size of uncompressed image}}{\text{size of original image}}
\]

### 4.1. Model 1: CNN Autoencoder

The first model that we experimented with is a simple convolutional autoencoder architecture. A generic autoencoder architecture consists of an encoder, that takes in an input image and produces a code (an intermediate representation which represents the compressed image), and a decoder, which takes in the code and reconstructs a lossy version of the original input.

![Figure 2. CNN autoencoder architecture](image)

#### Table 1. Results of both the autoencoder models on test set

<table>
<thead>
<tr>
<th></th>
<th>ConvAE</th>
<th>VAE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Avg PSNR</td>
<td>71.05</td>
<td>58.17</td>
</tr>
<tr>
<td>Avg SSIM</td>
<td>0.93</td>
<td>0.21</td>
</tr>
<tr>
<td>Compression ratio</td>
<td>0.01</td>
<td>0.13</td>
</tr>
</tbody>
</table>

For our baseline architecture, the encoder is a convolutional layer, followed by a non-linearity (ReLU), followed by a dense layer. This produces the compressed representation of the image (the code). The decoder is a dense layer followed by a deconvolutional layer, followed by a non-linearity. The non-linearity for the decoder is chosen to be a sigmoid in order for the output to have pixel values in the same range as the original input. The architecture used can be seen in Figure 2.

#### Experiments

Since our CNN Autoencoder model has fully connected layers and the images in our dataset are of varying sizes, we re-sized all images to be of the same size (128x128).

We chose the convolutional layer in the encoder to have 32 filters, each of size 3x3 and stride 1. The fully connected layer transformed the output from the conv layer to a code of dimensions 1x128. The deconvolutional layer in the decoder had 3 channels and dimensions that made the output from that layer match the size of the original input (1x1 filters).

We trained our baseline model on the training set (585 images) using Adam optimizer and MSE as the loss function. We tuned the learning rate (using range 1e-1 to 1e-4) using the validation set provided (41 images). The model with the lowest loss on the validation set was evaluated on the test set.

#### Results

The model with the highest PSNR and SSIM scores was trained with learning rate 1e-3.

The results of running our best model on the test set can be seen in Table 1. The table lists the average PSNR and SSIM on the test set along with the compression rate.

In this case, compression ratio = \(\frac{128+4}{128^2*128^3}\). This is because the compressed image (code) has size 1x128 and con-
Figure 3. Top: Original images. Bottom: Compressed images

contains floats whereas the original image has size 128x128x3 and contains uint8 characters.

Figure 3 shows some examples of compressed images from our model.

The model performs well even though the image is compressed significantly. However, a disadvantage of the model is that we would need to re-train the autoencoder if we want to change the compression rate (i.e. variable-rate encoding is not possible)[1].

**Code:** The entire pipeline (including the autoencoder model) was written from scratch.

### 4.2. Model 2: Convolutional Variational Bayes Autoencoder

Introduced by Kingma and Welling [12], Variational Auto Encoders (VAEs) are the generative analogs to autoencoders. In simple terms, they learn a hidden or latent representation which can then be used to sample images from a given distribution.

Using common VAE terminology, the training objective $L(x)$ is a tractable lower bound to the log-likelihood:

$$
\log p_\theta(x) \geq \mathbb{E}_{q_\phi(z|x)}[\log \frac{p_\theta(x, z)}{q_\phi(z|x)}] = -L(x)
$$

$$
L(x) = D_{KL}(q_\phi(z|x)||p_\theta(z)) - \mathbb{E}_{q_\phi(z|x)}[\log p_\theta(x|z)]
$$

Where $D_{KL}$ is the Kullback-Leibler divergence. The reparametrization trick is also to be noted, however we have not here gone into the details of how we sample or make certain distributions tractable using the reparameterization trick.

We have modified the above idea to include convolutional neural networks as well. The encoder comprises of two Convolutional layers (with ReLU), a dense layer followed by the latent sampling layer of the VAE which generates the compressed embeddings. The decoder is the reverse of reshaping the dense inputs and feeding it through 2 Convolutional layers to get the original input again.

### Experiments

We chose the convolutional layer (encoder) to have 20 filters, each of size 3x3, stride 1 and padding 2. The convolutional layer had dimensions that made the output from that layer match the size of the original input. The dense layer is of size 256 and the latent representation is 128.

We trained our baseline model on the training set (585 images) using RMSProp optimizer and the loss function as defined above. All experiments used a batch-size of 1 since the input images were all of different sizes. Due to the large size of the model and it causing out-of-memory errors and taking time to train, we then changed the images to black and white and resized them to 64x64. The results reported are on these values. The images are hazy as we have forced 87.5% compression from 64x64 uint8 to 128 float32.

The model with the highest PSNR and SSIM scores had filter size 3x3 and was trained with learning rate 1e-3.

As expected, due to the extreme compression this model is not able to capture the statistics well enough and not able to generalize well.

Figure 4. VAE. Top row: Original images. Bottom row: Compressed images.

The code is modified from the Keras tutorial about GANs.

### 4.3. Model 3: CNN Encoder + GAN Decoder

Next, we decided to experiment with using a CNN as an autoencoder and a GAN as the decoder.

The architecture for the CNN Encoder + GAN Decoder model was taken from [23]. The encoder takes in a 32x32 colour image and passes it through a series of convolution and pooling layers. The CNN Encoder can be seen in Figure 6.

We experimented with three different loss functions for training the GANs, which are explained briefly below.

**Vanilla GAN** First, we experimented with the standard loss for GANs [21]. The generator loss is:

$$
\ell_G = -\mathbb{E}_{z \sim p(z)}[\log D(G(z))]
$$

and the discriminator loss is:

$$
\ell_D = -\mathbb{E}_{x \sim p_{data}}[\log D(x)] - \mathbb{E}_{z \sim p(z)}[\log (1 - D(G(z)))]
$$

The code is modified from the Keras tutorial about GANs.
Wassertein GAN
The Wasserstein GAN paper has gotten a lot of attention since it was published [19]. The paper introduces a new loss called that uses the Wasserstein distance as the loss, which helps stabilize the training of GANs. Practically, the learnings from the WGAN paper can be summarized as follows: Remove the log from the loss, remove sigmoid from the output of the discriminator, clip the weights of the discriminator, train the discriminator more than the generator, use RMSProp instead of Adam and use a lower learning rate [20].

SSIM GAN
Lastly, we experimented with using Multi Scale (MS) SSIM as a loss function. MS-SSIM is a metric that is an extension of the SSIM metric that is differentiable and has been shown to perform well on image generation tasks [22].

Experiments
Due to memory and time constraints, we conducted experiments on a subset of the data for 5000 iterations. The best model from these experiments was trained on the entire dataset for 100,000 iterations.

As described above, we experimented with three different GAN model that had different loss functions - the vanilla GAN, Wasserstein GAN and SSIM GAN. We experimented with different hyperparameters for each of these models including different learning rates (1e-4 to 1e-6) and different optimizers (RMSProp and Adam).

Figures 6 and 7 show the results from the best models trained with the three different loss functions on a sample of training set data and a sample of the validation set data respectively.

Figure 6. Results of GAN architectures with different losses on a sample of the training data

Figure 7. Results of GAN architectures with different losses on a sample of the validation data

In figure 6, we can see that the vanilla GAN does not perform well. The WGAN and SSIM GAN both seem to perform equally well. To further compare the performance of the models, we ran both models on the entire training and validation set and we looked at the PSNR and SSIM scores of both models on the training and validation set. As we can see from the plots in Figure 9, both models perform well and it is hard to compare their performance since we are using two metrics. Overall, the WGAN seems to be performing marginally better than SSIM GAN. The best GAN architectures for both WGAN and SSIM GAN were trained using a learning rate of 1e-5 using Adam optimizer.

The best GAN model had SSIM 0.79 and PSNR 63.9 on the test set. The result from the GAN models seems promising but the models need to be trained for longer and further analysis needs to be done to prove the validity of the models. Similar to the autoencoder models, this model does not support variable bit encoding. This is because we are still using the CNN Autoencoder to compress the image.

Code: The starter code for the GAN implementations was obtained from [23]. We modified the code for our task and implemented two different loss functions. The code was modified to support different error analysis methods.

4.4. Model 4: RNN (ConvLSTM)
One significant drawback of the CNN encoder architecture was its requirement for fixed size images. As our
dataset consisted of variable-sized images (much like any real application would), we were forced to resize the images, ignoring aspect ratio; naturally, this affected our results adversely. We thus decided to move on to an approach that would allow variable-sized images.

Toderici et al [1] proposed an RNN-based compression model that allowed variable compression rates. The results presented in this paper were appealing; however, the model only worked for 32x32 sized images, bringing us back to the same drawback as CNN encoders. However, a subsequent paper from Toderici et al [2] based on the same architecture allowed variable bitrates and variable-sized images which works by dividing the image into 32x32 segments and learning the dependencies between them.

The compression network proposed in [2] consists of an encoding network E, a binarizer B and a decoding network D. E and D contain recurrent components. E encodes the image, B transforms it into a binary code for transmission to D, and D estimates the original image based on the given code. This is one “iteration” of the model. In each iteration, the residual error is calculated as the difference between the original image and the decoder reconstruction. Network weights are shared between iterations, and states in recurrent components are propagated to the subsequent iteration - thus, the residual is encoded/decoded in a different context depending on the iteration.

The network is shown in Figure 10. The Encoder consists of a Conv layer, followed by three RNN layers (in our implementation, we use ConvLSTMs). The Binarizer converts its input to binary code using a Conv layer. The Decoder takes this as input and reconstructs the image by passing it through a Conv layer, four RNN layers (also ConvLSTMs), and a final Conv layer. The details regarding filters and stride are shown in the figure. The “Depth to Space” module represents a rearrangement of the elements of the matrix to upscale resolution, i.e. reduce depth and increase height and width. Tanh nonlinearities are used to calculate hidden state after the binarizer and decoder layers. The equations for LSTM are given below:

\[
[f, i, o, j] = [\sigma, \sigma, \sigma, \sigma, \tanh]^T \left( (W x_t + U h_{t-1}) + b \right),
\]

\[
c_t = f \odot c_{t-1} + i \odot j,
\]

\[
h_t = o \odot \tanh(c_t),
\]

Where \(x_t\), \(c_t\), and \(h_t\) denote input, cell, and hidden states at iteration \(t\). \(f, i, o, j\) are the LSTM gates, and \(\sigma\) is the sigmoid function \(1/(1 + e^{-x})\).

At each iteration, a decoded reconstruction of the image is constructed, and its residual calculated based on L1 loss with the original image. Each iteration represents 1/8 bit per pixel (bpp) of compression.

For the binarizer, an architecture similar to PixelRNN[18] titled BinaryRNN is used, as depicted in Figure 11. The end-to-end probability estimation consists of three stages: first, a convolution is used to increase the receptive field of the LSTM; second, the line LSTM accepts the re-

Figure 8. Top: Results on the train set, Bottom: Results on the validation set. Y axis - PSNR/SSIM score, X axis - number of iterations.

Figure 9. Results of the WGAN on the train set after 1000,2000,3000,4000 iterations, showing the improvement in performance with increasing number of iterations of the model.
Figure 11. Binarizer architecture in the RNN model

As a result of the initial convolution and the previous iteration and processes one scan line at a time; finally, two convolutions are added to increase capacity of the network. A sigmoid activation is used in the final convolution to estimate the Bernoulli distribution. The loss is cross-entropy.

Experiments

We experimented with this model, following the model described in the paper closely. The authors trained on 6 million 1280x720 images - as we could not mirror these training conditions, our results did not meet the reported results. However, we were able to see a definite improvement from the CNN models we had experimented with previously.

We performed hyperparameter tuning, experimenting with the learning rate in the range of (1e-4, 1e-2), different loss metrics (L1, L2), different optimizers (Adam, RMSProp) and different numbers of iterations - finally settling on a learning rate of 5e-4, L1 loss, an exponential decay lr scheduler with SGD (as mentioned in ), batch size 128 (the highest we could set without facing memory issues), and 16 iterations. We faced several issues with respect to memory - the amount of memory consumed per iteration began to blow up until we modified our approach and made some of the variables “no-grad”. Large models such as these along with a significant number of residuals are not trainable on a single GPU.

Results

First we have the results from training on just 32x32 images and then the results for modified RNN structure to deal with larger images.

Seeing that the performance is not good enough on training on 32x32 sized images, we also ran a model on full sized images. As 32x32 images are too small to be seen, this is the final model we have along with comparison to JPEG to see where we are. We also have images of the decompressed images in full resolution to see its performance.

To conclude this section, we see that the results did not beat JPEG compression in PSNR or MS-SIM - however, they are visibly better than the VAE results previously shown. The fact that variable sized images can be used is a significant advantage in real-world applications. With extensive training (as done in the paper), these models can be shown to beat JPEG as well as CNNs in terms of PSNR and MS-SIM.
4.5. Model 5: RNN (ConvLSTM) with a modified loss

Looking at the results from Model 4, we see that we are not able to capture SSIM well enough. Training a lossy image compression network introduces a dilemma of which is the right function to optimize. We see that using a perceptual metric is not good enough as the underlying loss behind these (like SSIM) have poorly conditioned gradients. Thus the authors of [4] have suggested a loss that are a bridge between the traditional L1 and L2 and a perceptual metric. The loss between 2 images \( x \) and \( y \) is given by

\[
L(x, y) = w(x, y)||x - y||_1, w(x, y) = \frac{S(x, y)}{S'}
\]

where \( S(x, y) \) is a perceptual measure of dissimilarity between images \( x \) and \( y \) and where \( S' \) is a dissimilarity baseline. During training, the baseline \( S' \) is set to the moving average of \( S(y, x) \). In practice, we use a local perceptual measure of dissimilarity. \( S(x, y) \) is measured in the following way. The image is first split into 8 blocks. Over each of these blocks, a local weight is computed using

\[
D(x, y) = 0.5(1SSIM(x, y))
\]

as the dissimilarity measure (DSSIM). The average of all the values for each block is the weight of the image.

Experiments and Results

Using the same model as Model 4 and similar hyperparameters, we modify the loss function and rerun the model and then see the results. We notice that for the image sized we were looking at i.e. 32x32, the variation of SSIM we have used is a poor approximate of the weight and the results our far worse. Notably the usual trend of optimizing for SSIM at the cost of PSNR is seen as we move towards higher bit rates.

This section has been completely implemented from the paper with SSIM definitions taken from https://github.com/Po-Hsun-Su/pytorch-ssim.

5. Conclusion

This project tackles image compression using deep learning, an increasingly important field with significant scope for growth. In this project, we experimented with several different models, analyzing each in detail and discussing their performance in comparison with other models (along with reasons why). We obtained quite good results, in spite of computational and memory limitations - on the whole, it was an exciting exploration of an upcoming field in deep learning.

We faced a variety of challenges when tackling image compression and tried several approaches to obtain performances promised in literature. We note that the more complex models we experimented with like GANs and RNNs, outperform our VAE SSIM baseline by a large margin - thus showing the need for using such architectures for image compression.

We also note that overall, without any restrictions on size etc, JPEG still does exceptionally well and runs in the least amount of time. Encoding and decoding for RNNs is a time-consuming process which is accelerated by using GPUs. For GANs, the training time is large and the model requires a lot of fine-tuning.

By far, the largest challenge is generalizing well on the dataset. We see that we have used high resolution pictures taken by professional photographers of different landscapes, without much overlap in the train, valid and test, (especially in the test and the train+valid). This can be seen from Figure 14, where we are able to generate good quality images for validation but not on train. This shows that the test is not representative of the valid. Differential scaling and random cropping did not make a significant difference, and only a marginal increase in performance occurred when we moved to full sized images.

Another challenge of this space is that capturing information from a large number of images and learning latent representations from them is not only a computational challenge, but also a challenge for deep learning, wherein we usually pick images for a specific task with similar latent properties, which is not the case here.

RNNs performed quite well for the large images, showing the biggest advantage of using RNNs, which can capture the dependencies not only between residuals, but also between adjacent images. The drawback is that it is time consuming for each pass as we are using stacked LSTMs. GANs have reasonable performance and show promise; they are likely the future of compression, because they have performance comparable to state of the art (RNNs), and thus serve as a natural next step to explore. For our dataset,
the RNN model is the most flexible in variable compression rates, but the best compression in terms of PSNR and MS-SSIM are from GANS and AEs.

Future work in this scope could include working with a more representative dataset, and fine tuning some of the general-purpose deep learning ideas from classification for image compression. Our report discusses the challenges faced by various models when tackling image compression: in this task, the whole framework needs to change as well as the computation process needs to be sped up. Increased computational power and a larger dataset would also improve the performance of these models. Future work in GANs could include larger image sizes and better loss functions catered towards compression.

References

[14] https://whatis.techtarget.com/definition/image-compression
[24] https://github.com/1zb/pytorch-image-comp-rnn