Generative Neural Network Based Image Compression

Aum Khatlawala
CND Student
IIIT-Hyderabad
2020113008
aum.k@research.iiit.ac.in

Aaron Monis
CSE Student
IIIT-Hyderabad
2020101129
aaron.monis@students.iiit.ac.in

Khush Patel
CSE Student
IIIT-Hyderabad
2020101119
khush.patel@students.iiit.ac.in

Rohan Modepalle
CSE Student
IIIT-Hyderabad
2020101130
rohan.modepalle@students.iiit.ac.in

***Abstract*—The paper aims at finding an alternate compression technique that can serve as an improvement over traditional compression techniques like JPG and WebP with respect to different testing metrics. The paper proposes a neural network based image compression approach for doing so. The benefits of using such a technique are as follows:**

1) **Much better compression percentages.**
2) **Improved fidelity of the construction.**

I. Introduction

The neural network based solution proposed in the paper is that of GANs (Generative Adversarial Networks).

Let's understand what GANs are before moving to the approach of the paper and the methods used.
GANs are a class of Machine learning frameworks that consist of two parts – a generator and a discriminator.
The generator is responsible for generating fake samples and the discriminator tries to tell if it is real or fake. This is the adversarial nature of GANs.
The result is revealed to both the generator and the discriminator and is a zero-sum game where the loser has to update its weights and the winner can remain the same.
The ultimate goal is to be able to generate samples that can fool the discriminator as well as the human eye.
GANs are often used in video frame prediction, image generation, enhancing or compressing images and in surveillance.

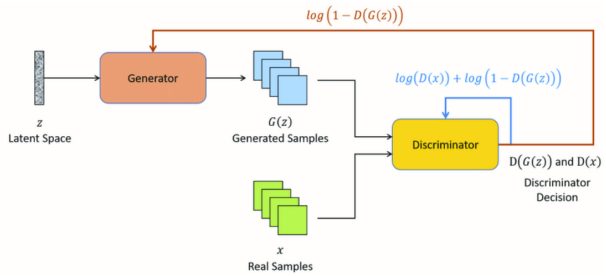

Fig. 1: GAN

A. How GANs help in Image Compression

GANs and VAEs are used in conjunction to compress images. The decoder of the model is the generator for the GAN. VAEs are good for reconstructing the images whereas GANs are used in producing high-quality images which don't seem to be very distinguishable from the images of the training set. The problem is that GANs give us only the decoder but not the encoder. The authors of the paper propose a remedy to this problem by using the concept of GAN Reversal.

B. GAN Reversal

A random latent space vector is used and then recovering is done using their approach of GAN Reversal. A vector is trained in the latent space for this compression. Other research typically trains the model with standard L2 loss functions but the authors of this paper try to experiment with other loss functions like structural similarity index and this is shown to improve the results obtained.

II. Methods and Approach

The adversarial nature of GANs is due to the two sub-models (Generator and Discriminator) it consists of that work together to form the GAN model. The job of the generator is to create images that cannot be distinguished from the images in the test dataset by the discriminator model. The job of the discriminator is to try to correctly classify the image from either the domain (test dataset) or the generator as either real or fake. The trained generator will be able to map a low-dimensional latent vector to an image.

For the purpose of this project, we used a pre-trained GAN model - ProGAN128 that was trained on the CelebFaces dataset. The GAN model is able to generate 128×128 images that look like human faces (albeit with inaccuracies) if we provide a 512×1 latent vector that can be generated randomly.

Firstly, we randomly generate a uniformly distributed latent vector with dimensions 512×1 in the range [-1, 1] and call this z'. Let the mapping of this (image generated by the GAN) be a function of z' called f(z').
Let the value of the latent vector space of the original uncompressed image be z and the original uncompressed image a function of z called f(z).

Now, we define the similarity/closeness between the original image and the GAN-generated image as the mean squared error loss function given by the formula:

$$l_2(z') = ||f(z) - f(z')||_2^2 \tag{1}$$

However, the l_2 loss function only gives information about pixel-wise similarity and is not a measure of how well the overall image is recovered. This leads us to use perceptual similarity metrics for training. One such metric is SSIM (Structural Similarity Index). This index takes two aligned windows of the two images and computes the similarity between them using the following formula:

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

Since this value becomes close to 1 as similarity increases, the loss function to be minimised can be formulated as follows:

$$L_{SSIM} = \frac{1}{N} \sum_{(x,y)} 1 - SSIM(x, y) \tag{3}$$

We can also use the Peak Signal Noise Ratio which is given by:

$$L_{PSNR} = 20 \log_{10}(\frac{MAX}{\sqrt{MSE}}) \tag{4}$$

where MAX is the maximum possible value of a pixel in the image.

We create a combined loss function given by:

$$10 * MSE + (1 - SSIM) - 0.01 * PSNR \tag{5}$$

The larger value for the MSE is important since it ensures that the image generated by the GAN is as close as possible to the original. The values of SSIM and PSNR help improve the metrics for SSIM and PSNR as compared to just using MSE.

In order to test our approach we first created a CelebFaces Dataloader that contains images similar to but not the same as images from the CelebFaces Dataset. This allows us to effectively test the approach described by the authors since they claim that their approach is effective across different images in the same domain. We prepared over 11000 images to use but were only able to use a small subset (about 100 to 200) due to computational limits. We then created the pipeline for testing the various compression methods utilised by the authors. We first set up the ProGAN 128 that would be used.

In order to utilise the ProGAN, we needed to use TensorFlow 1. We first download the GAN from **tfhub** which contains a large number of GANs for various uses. For every image that we run, it is important that we reset the graph of the GAN to its original state to prevent errors or prevent the GAN from picking up information from images. We then run the GAN for 1000 iterations on the image. This is done by first setting up a random latent vector zp and then iteratively modifying the latent vector by comparing the result of the GAN when the latent vector was passed to it versus the original image. This allows us to generate a low-dimensional latent vector that can be used to generate the image. We also passed the original image to PIL and opened it as a *JPEG1* and *JPEG10* image. We also used the *scikit-learn.clusters* KMeans package to compress the image using the KMeans $= 4$ compression method suggested by the author. Each of the images was then saved and passed to the function containing the code to find the results of each of the compressions.
We repeat this process for each of the images and then generate the final metrics as the average over all images.

We also tested the above approach using the Adam Optimiser with Exponential Gradient Descent with two different loss functions. We first used the Mean Squared Error Loss directly as suggested by the authors and also used a loss function combining MSE, PSNR and SSIM which helps to improve the performance across each of the metrics. The loss function has been described above.

III. Test Set

We used the images from this github repository Link.

The images were originally in 256*256 dimensions, which we scaled down to 128*128 dimension.

We randomly picked 500 of such images for testing purposes.

IV. Metrics Used

The objective of the paper is to obtain higher compression rates compared to standard compression protocols while retaining the perceptual quality after image reconstruction of the input image and make it as similar to the original image. BPP is the number of bits used to represent one pixel. The lower the BPP, the higher the compression.
Metrics such as MSE, PSNR, CR and SSIM have been used to measure the quality of reconstruction.
MSE stands for Mean Squared Error, PSNR stands for Peak Signal to Noise Ratio, SSIM stands for Structural Similarity Index and CR stands for Compression Ratio.

A. MSE

While MSE seems like the easiest metric to use, it is not too indicative of perceptual or textural quality as the other metrics mentioned.

B. SSIM

SSIM was first proposed by Wang and others in a research paper in 2004. It has been shown that it is more indicative of the perceptual quality and thus has gained traction as the metric to use in applications where texture and perception are the most important.

C. Peak Signal Noise Ratio

The PSNR computes the peak signal-to-noise ratio, in decibels, between two images. This ratio is used as a quality measurement between the original and a compressed image. The higher the PSNR, the better the quality of the compressed, or reconstructed image.

D. Bits Per Pixel (BPP)

This metric is used to measure the compression achieved by a scheme. It measures the magnitude of compression achieved by indicating the average number of bits needed to represent one pixel. For an uncompressed image, the BPP is 24 (each colour channel uses one byte ($3\times$ 8)). However, upon losslessly compressing the image, the BPP will be lesser than 24.
BPP is given by:

$$BPP = \frac{S_{compressed}}{128 \times 128} \tag{6}$$

Here, $S_{compressed}$ is the size of the compressed file when it is losslessly compressed.

E. Compression Ratio

Compression Ratio (CR) is defined as:

$$CR = \frac{BPP_{uncompressed}}{BPP_{compressed}} \tag{7}$$

It is important to note that the product of BPP and CR remains constant.

F. Results Table

Since the aim of the paper is extreme compression, the final outputs were compared with the metrics applied on the JPEG 1 percent compression. The aim is to beat this particular compression scheme since GANs achieve a similar percentage of compression and the main metric is SSIM as it is the best indicator of how similar two images are.

Scheme	MSE	SSIM	PSNR	BPP	CR
Uncompressed	0.0	1.0	inf	32.87	32.87
K-Means=4	171.61	0.74	26.02	4.15	3.34
JPEG 10%	63.35	0.85	30.29	1.07	12.70
JPEG 1%	280.30	0.60	23.74	0.80	16.91
GAN Reversal	208.41	0.72	25.33	28.49	0.48

V. Results and Discussion

We aim to reduce the value of Mean Squared Error as low as possible while simultaneously decreasing PSNR and increasing SSIM. This allows us to improve the structure of the image.

We observe that the generated images from the GAN are better than JPEG1 and close enough to KMeans=4 but not as accurate as JPEG10. As visible in the examples, the eyes and mouth are often places that are visibly different from the original and can be improved by using a better pretrained GAN.

We ran the simulations using the combined loss function defined earlier (using a linear combination of SSIM, PSNR and MSE). The results we obtained are as follows:

We ran the simulations using the MSE loss function. The results we obtained are as follows:

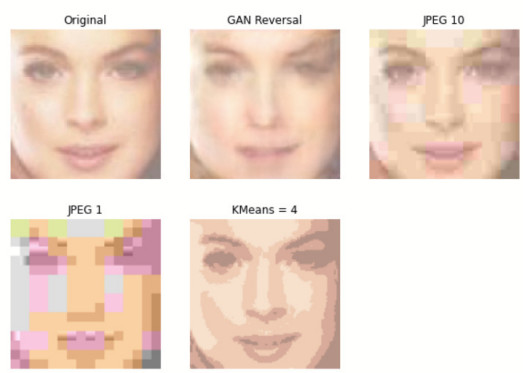

Fig. 2: Combined Loss Function Output 1

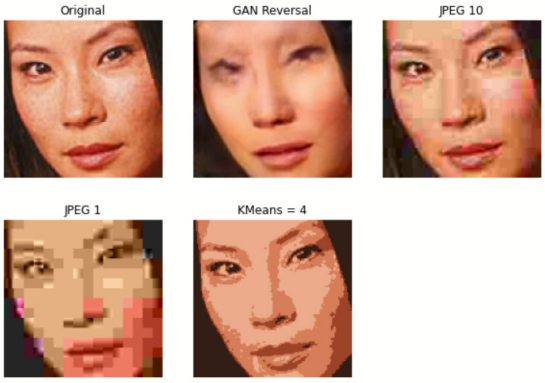

Fig. 3: Combined Loss Function Output 2

Fig. 4: MSE Output 1

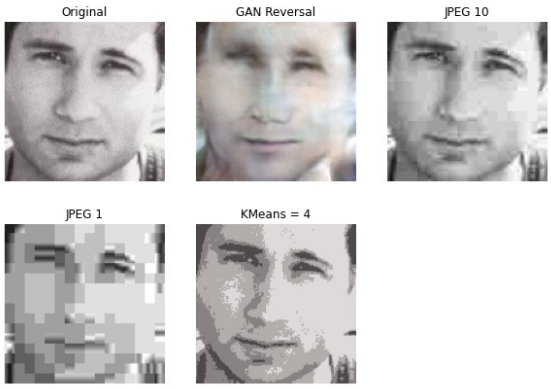

Fig. 5: MSE Output 2

VI. Conclusion

We observe that the generated images from the GAN are better than JPEG1 and close enough to KMeans=4 but not as accurate as JPEG10. As visible in the examples, the eyes and mouth are often places that are visibly different from the original and can be improved by using a better pretrained GAN.

Better results can be obtained by running the model over more iterations but this is impractical for real-life implementations. We aim to improve this by using a better GAN and searching for metrics that can improve the accuracy of the generated image over a few hundred iterations.

VII. Limitations

We faced a number of limitations while implementing the authors' novel approach to compression. The first of which is the GAN. The pretrained ProGAN128 was the only publicly GAN model we were able to find. The GAN model works only on TensorFlow 1 making its application to future works difficult.

Moreover, the GAN is not very accurate. Although the GAN can generate images that look like humans, it lacks the accuracy and quality of images that usually accompany well-trained GANs. Additionally, GANs are heavily reliant on GPUs to improve computational speed making this implementation difficult for users or systems aiming to use GANs without the use of GPUs.

This particular implementation performs reasonably better than JPEG 1 across MSE, PSNR and SSIM and is close to the KMeans compression strategy but cannot match the accuracy of JPEG 10. This is because the model was run only for 1000 iterations while the authors suggest close to 10000 iterations to match the results they obtained. Doing so takes a lot of time and we simply do not possess the computational capacity to test and check the results on it which also makes this impractical for professional use.

VIII. Future Work

We believe that the GAN is a major bottleneck in this implementation which is why we plan to improve upon our work by using better GAN models. These GAN models must be able to generate high-quality human faces quickly and accurately.

We also plan to search for metrics and/or methods that could be used to improve the compute time required by the GAN to iteratively refine the result obtained from the latent vector and also boost its accuracy by using it in the loss function.

Contributions

Aaron - Model Training (Combined Loss function) and Integration

Khush - Model Training(MSE) and Test Set Creation

Aum - Metrics Testing

Rohan - Metrics Integration