IMAGE COMPRESSION USING DEEP LEARNING

By
Nishit Anand (9918103133)
Shreya Agarwal (9918103146)
Aditi Dixit (9918103155)

Abstract

The ever increasing availability of cameras produces an endless stream of images. To store them efficiently, lossy image compression algorithms are used in many applications. Instead of storing the raw RGB data, a lossy version of the image is stored, with hopefully minimal visual changes to the original.

Various algorithms have been proposed over the years, including using state of-the-art video compression algorithms for single image compression.

At the same time,deep learning-based lossy compression has seen great interest, where a neural network is directly optimized for the rate-distortion trade-off, which led to new state-of-the-art methods.However, all of these approaches degrade images significantly as the compression factor increases.

We propose GAN based image compression which does not degrade the image quality and gives results which are visually identical to the original image.

Problem Statement

In today's world, most of the data is in the form of images. We take so many photos with our smartphones. Different social media platforms like Instagram are driven by multimedia data. Images take a lot more storage than text files. All this data is stored either on our devices or on the servers of companies like Meta, Google, Twitter. Thus, reducing the space taken by them and compressing them would not only benefit MNCs by saving them storage space on their servers, but also us as our mobiles and laptops would be able to store more content.

Thus, in today's day and age, image compression has become a topic which needs to be put light upon.

Image compression deals with minimizing the number of bits required to represent an image.

Significance

1. Image Compression benefits from better reconstructions at lower bitrates, reducing the amount of storage needed to save pictures and the bandwidth required to transmit pictures.

2. The latter is important as wireless technology typically lags behind user trends which have been continuously requiring higher bandwidths over the past decades, and there is no end in sight with emerging applications such as virtual reality.

3. Furthermore, better compression technology improves accessibility in developing areas of the world where the wireless infrastructure is less performant and robust than in developed countries.

Background study

We extensively studied how to combine Convolutional Neural Networks and Generative Adversarial Networks to obtain a state-of-the-art generative lossy compression system.

We studied the literature pertaining to learned image compression and read many research papers.

In the past image compression has been done using Convolutional Neural Networks (CNNs), Recurrent Neural Networks (RNNs) etc.

In contrast to previous work-

1. We obtain visually pleasing compressed images that are perceptually similar to the input
2. Our approach can be applied to high-resolution images well

Requirement Analysis

We require the following for the project:-

● A laptop or computer running Windows 10 or macOS Sierra or above or Linux, or Ubuntu 16.04 LTS

● A dual core processor 2 GHz or more

● 8GB of RAM or more

● NVidia GPU 1000 series or above

● Jupyter Notebook

● Google Colab

We also require these things:-

- For training our model we need training data. For this we need a large dataset of a wide variety of images. We need this, so that the model can learn from it and is able to generalize well and compress the images efficiently.
- We also need testing data to check how our model performs and to fine tune it.

Data Sets used

Our training set consists of a large set of high-resolution images collected from the Internet. We evaluated on four diverse benchmark datasets collected to demonstrate that our method generalizes beyond the training as well: the widely used Kodak dataset, as well as the CLIC2020 dataset and DIV2K dataset along with a subset of the Imagenet Dataset.

The Kodak dataset and the DIV2K Dataset mostly contain high-resolution images. We did not adapt our models in any way to evaluate on these datasets, and we evaluated on the full resolution images.

Our Model : In Depth

- Our proposed solution is to use GANs (Generative Adversarial Networks) and CNNs (Convolutional Neural Networks) for learned image compression.
- First Convolutional Neural Networks are fed the images from which they learn the important features in the image.
- Then they pass on these extracted features to the Generative Adversarial Network. In Generative Adversarial Network, the Generator and Discriminator, work on these extracted features and create a new image from scratch which has only the important details and gets rid of less useful information.
- Thus, giving us a compressed image which looks perceptually the same as the original image, but is much smaller in size.

These are the steps we did for implementing our proposed solution:

1. Collecting the datasets:

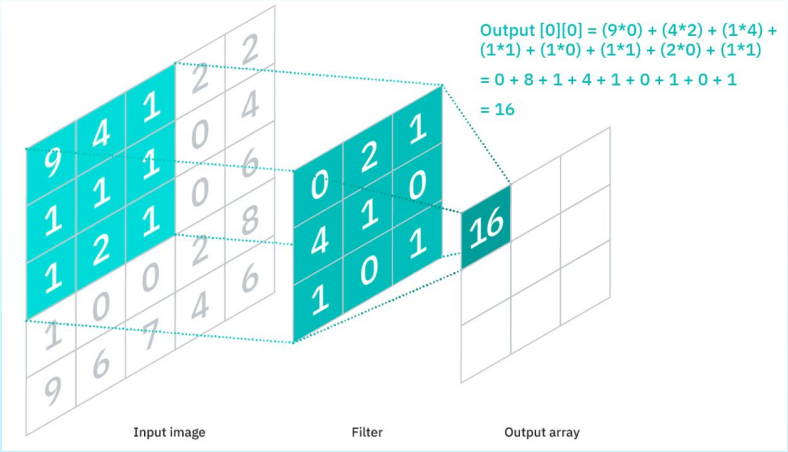
We collected the CLIC2020, Kodak and DIV2K datasets. We also collected a subset of the Imagenet dataset. We divided the collected dataset into training and testing data in the ratio 80% to 20%. This is important to be done, so that the model can learn and get trained on the input images.

2. Preprocess the input image files:

We wrote a python program and used the OpenCV python library for this. This is done, so that the Convolutional Neural Network can learn from these images.

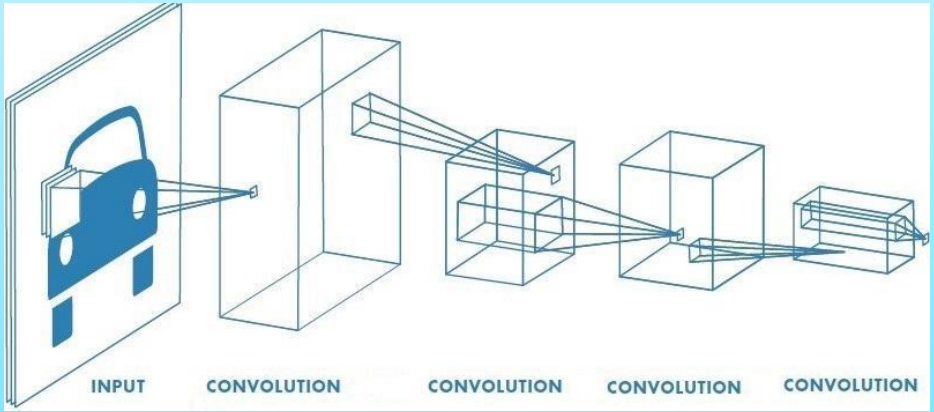
3. Feature Extraction from the images:

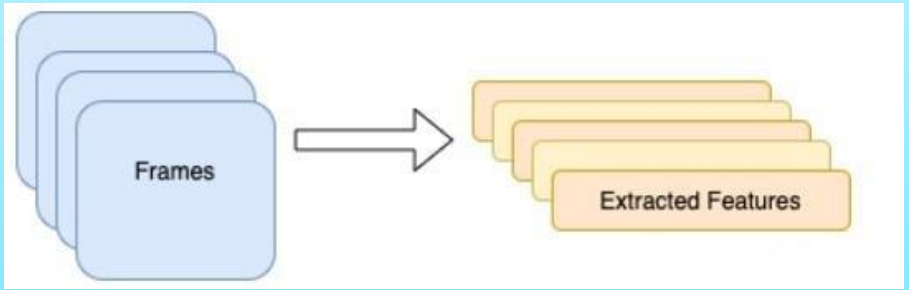
We wrote a Machine learning model made up of Convolutional layers and Maxpool2D layers in between them. The Convolutional Neural Network extracts spatial features from the images and trains on them, so that it can better predict which information in the image is to be selected for compression and which not to be selected.



4. Reducing Overfitting:

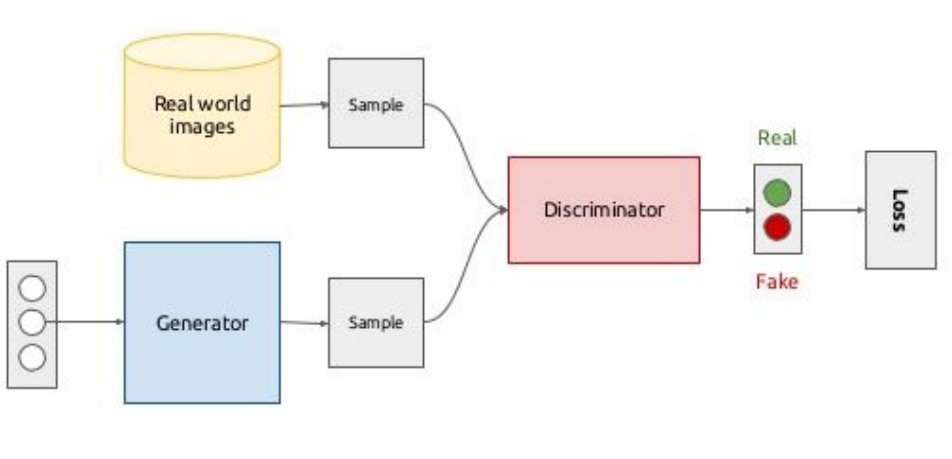
We noticed that the model was overfitting, so we added dropout layers with chance of 0.5 in between the convolutional layers. This helps reduce overfitting. The output of these layers gives us the features our model has extracted from the images





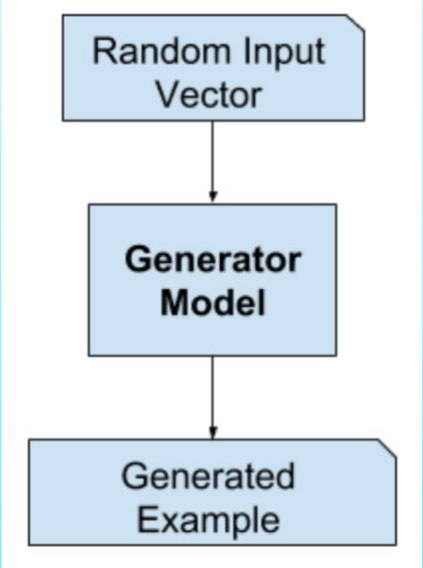
5. Feeding these extracted features into GAN:

After extracting Spatial Features via Convolutional layers, these extracted features are then input into the (GANs) Generative Adversarial Network. The Generative Adversarial Network, consists of two parts -
The Generator and
The Discriminator



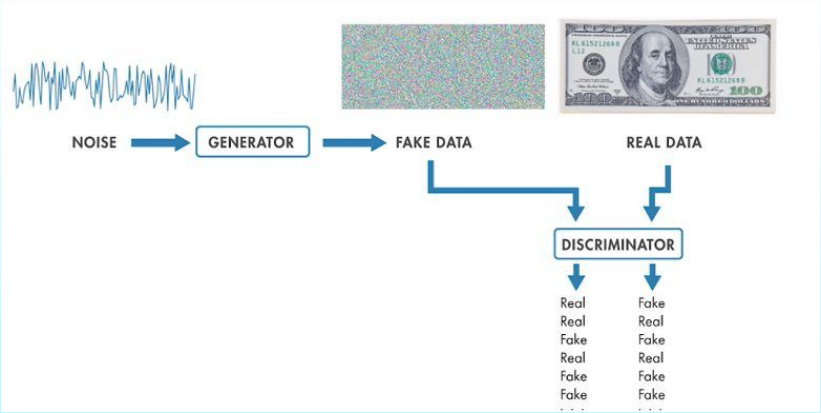
6. Generator learns and produces new images:

The generative model G is trained to generate new examples from random input and the discriminative model D estimates the probability that a sample came from the training data rather than G. The training procedure for G is to maximize the probability of D making a mistake.
The Goal of the Generator is to fool the discriminator.



7. Discriminator starts assessing the images produced by the Generator:

The discriminator is provided both with an input image which is real and the image generated by the Generator.



Discriminator classifies whether the image generated by the Generator is real or fake, and returns it's assessment to the Generator, in turn teaching the generator to generate images which fool the discriminator.

8. Training the model:

We trained the model for 20,000 iterations for 36 hours. We made changes to the model multiple times and trained the model again to increase it's accuracy.

In the end, the Generator is forced to create images which are very similar to the original images.

9. Output Compressed Images:

After the Generator and the Discriminator have been trained completely, then they produce images which are much smaller in size compared to original ones, but of almost the same quality visually.

10. Fine Tuning the Model:

We changed hyperparameters and fine tuned the model in order to decrease the output size of images.

```
nishit@nishit-ubuntu: ~/Downloads/ne...
[2022/05/10 20:19:00] Epoch: 21/300 Loss: 0.5741/0.6385
[2022/05/10 20:19:02] Epoch: 22/300 Loss: 0.5608/0.6241
[2022/05/10 20:19:05] Epoch: 23/300 Loss: 0.5562/0.6175
[2022/05/10 20:19:07] Epoch: 24/300 Loss: 0.5497/0.6143
[2022/05/10 20:19:09] Epoch: 25/300 Loss: 0.5456/0.5866
[2022/05/10 20:19:12] Epoch: 26/300 Loss: 0.5324/0.6024
[2022/05/10 20:19:14] Epoch: 27/300 Loss: 0.5167/0.6271
[2022/05/10 20:19:17] Epoch: 28/300 Loss: 0.5318/0.5948
[2022/05/10 20:19:19] Epoch: 29/300 Loss: 0.5243/0.6057
[2022/05/10 20:19:21] Epoch: 30/300 Loss: 0.5178/0.5773
[2022/05/10 20:19:24] Epoch: 31/300 Loss: 0.5020/0.5733
[2022/05/10 20:19:26] Epoch: 32/300 Loss: 0.4968/0.5636
[2022/05/10 20:19:28] Epoch: 33/300 Loss: 0.5022/0.5620
[2022/05/10 20:19:31] Epoch: 34/300 Loss: 0.4909/0.5665
[2022/05/10 20:19:33] Epoch: 35/300 Loss: 0.4924/0.5656
```

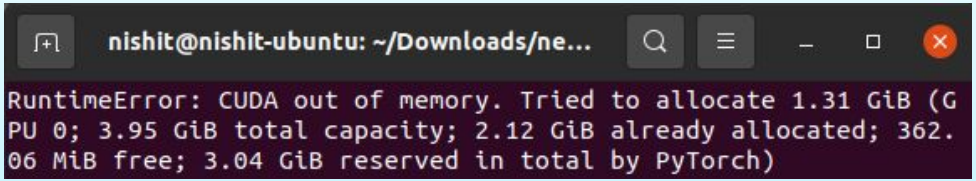
$$MSE = \frac{\sum_{M,N} [I_1(m,n) - I_2(m,n)]^2}{M * N}$$

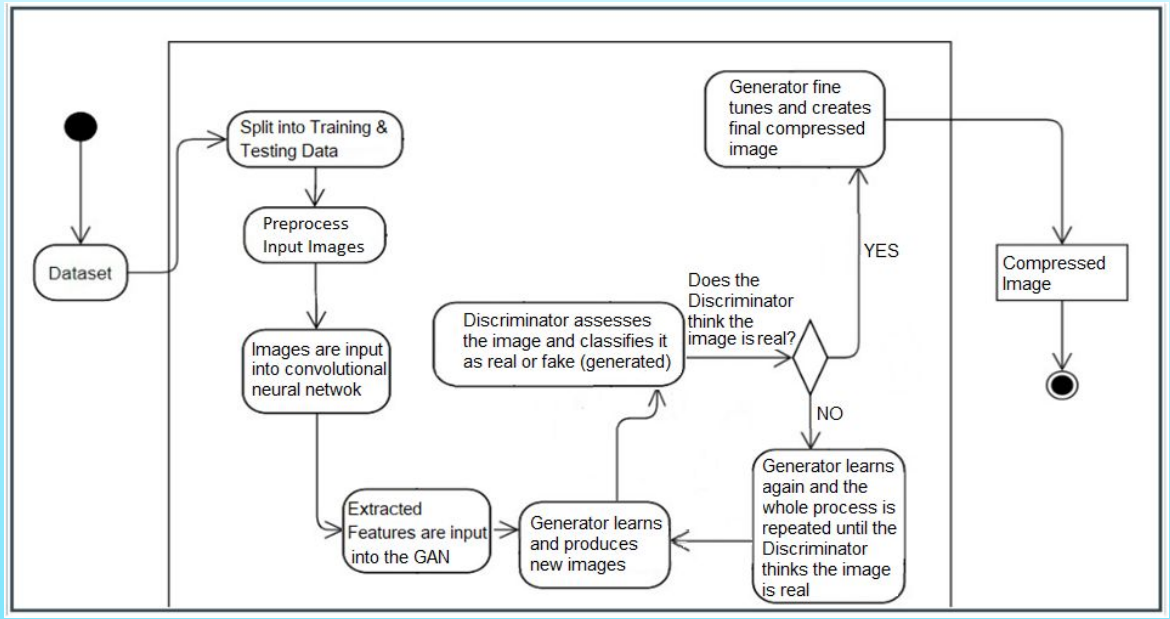
$$PSNR = 10 \log_{10}\left(\frac{R^2}{MSE}\right)$$

11. Optimizing the model:

Earlier our model could only compress images upto 500kB and upto 400x400 resolution. This was because it used a lot of GPU memory and our laptop GPU - GTX 1050Ti Max Q had 4GB of VRAM. Compared to server GPUs which have upto 24GB of GPU VRAM, 4GB was much less. Because of this, it gave out of memory (OOM errors) sometimes. So we optimized our model and made it lightweight compared to other Image Compression models by reducing a few layers from the model.

Now, after this, the model became much faster and now it is able to compress images upto 1500kB (1.5 MB) which is 3 times more as compared to before and upto 1000x1000 resolution which is 6 times more resolution as compared to before.

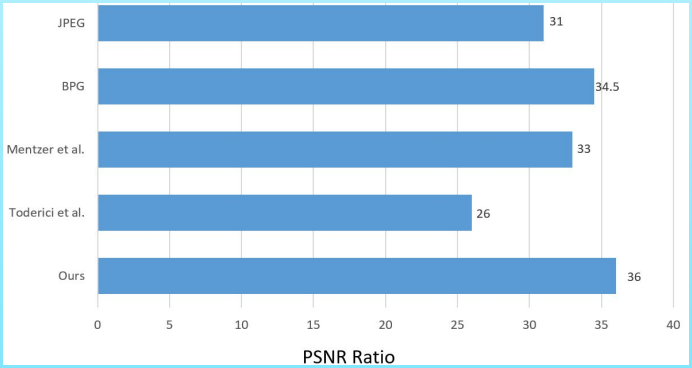


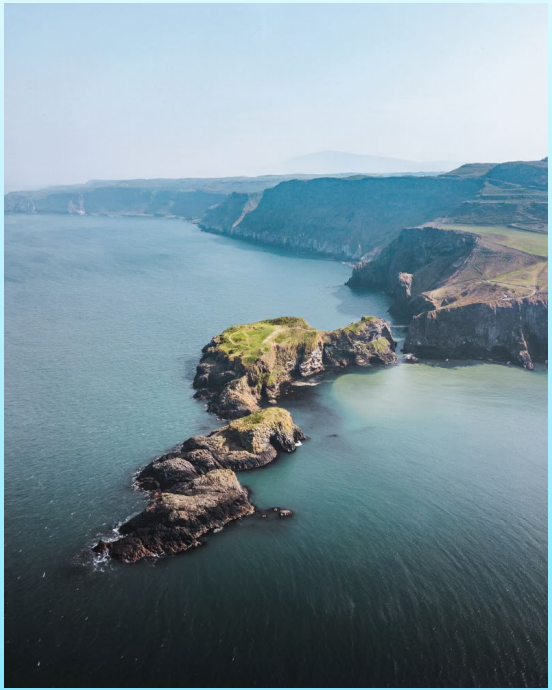


Results and Testing

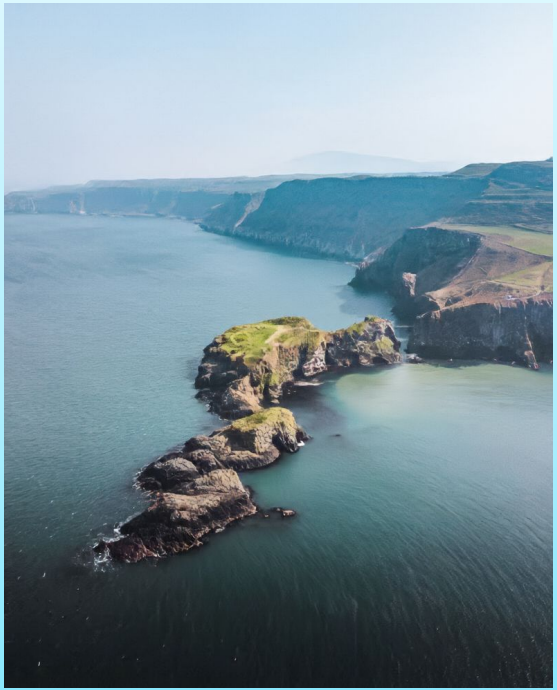
We changed hyperparameters and fine tuned the model in order to decrease the output size of images and to increase the PSNR ratio of the output images.
Finally, we achieved a PSNR Ratio of 36 on the datasets which is quite higher as compared to previous techniques in the field of Image Compression. This is the PSNR (Peak Signal to Noise Ratio) of our Model compared to other works-

Model	PSNR (Peak Signal To noise Ratio)
JPEG	31
BPG	34.5
Mentzer et al.	33
Toderici et al.	26
Ours	36





Before Compression (1.1 MB)



After Compression (23.1 KB)

After Training our model extensively on 20,000 iterations and fine tuning it, these are the size of images before and after compression-

Original Size of Image	Image Size after Compression	Compression Ratio (Original/Compressed)
1500kB	35.0kB	42.85
1000kB	24.7kB	40.48
500kB	17.8kB	28.09
100kB	8.7kB	11.49

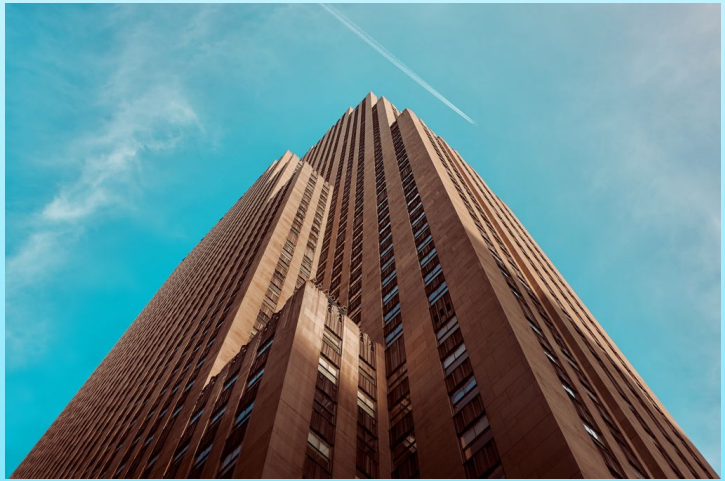
Earlier our model could only compress images upto 500kB and upto 400x400 resolution.
We optimized our model and made it lightweight compared to other Image Compression models by reducing a few layers from the model.
Now, after this, the model became much faster and now it is able to compress images upto 1500kB (1.5 MB) which is 3 times more as compared to before and upto 1000x1000 resolution which is 6 times more resolution as compared to before.

Property of the Model	Our Model Before Optimization	Our Model After Optimization
Maximum Image size which could be compressed	500kB	1500kB (1.5 MB)
Maximum Image resolution which could be compressed	400x400 pixels (160,000 pixels)	1000x1000 pixels (1,000,000 pixels)

Table - Our model before and after optimization

Before Compression (1.5 MB)

After Compression (35.0 KB)



Before Compression (1.0 MB)

After Compression (25.0 KB)

Before Compression (1.1 MB)

After Compression (34.0 KB)

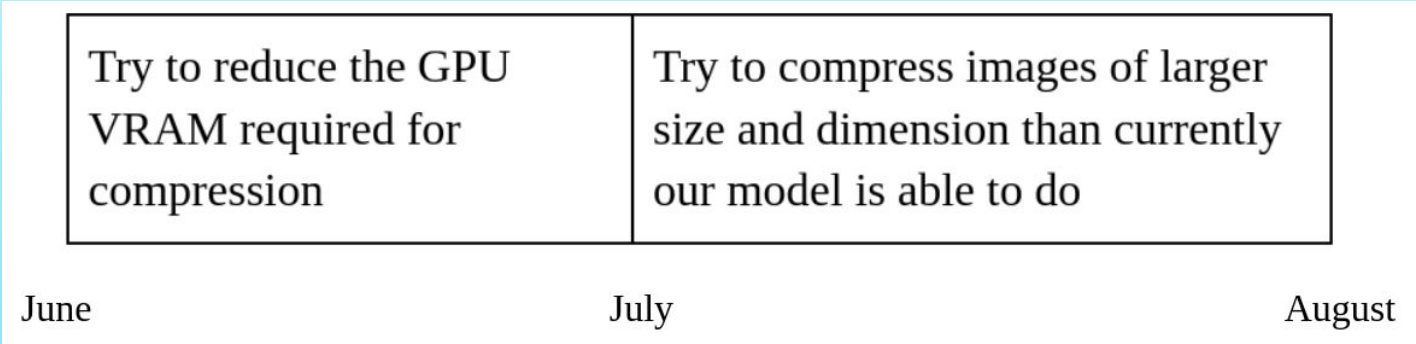
Before Compression (1.1 MB)

After Compression (27.8 KB)

Conclusion and Future Scope

Thus, this is our idea to create a framework for image compression using Deep Learning. We have used GANs (Generative Adversarial Networks), which help reduce the size of output image by quite a bit as well as help in maintaining image quality. In the future, we will attempt to compress images with even less resources and amount of GPU VRAM

Future Plan and Gantt Chart



Thank You