ELL793 REPORT

ASSIGNMENT 3 (GAN)

SARANSH AGARWAL (2019MT60763)
MANISH BORTHAKUR (2019MT60493)

INTRODUCTION

Generative Adversarial Networks (GANs) have proven to be successful in generating high-quality images of large size. However, most of the progress has been focused on improving discriminator models to train more effective generator models, and less emphasis has been placed on improving the generator models themselves. To address this, a new variant of GAN architecture called Style Generative Adversarial Network (Style GAN) has been proposed. Style GAN introduces significant changes to the generator model, such as incorporating a mapping network to map points in latent space to an intermediate latent space. For this assignment, we will apply GAN on the CIFAR10 dataset in two parts.

PART 1

A) Load a pretrained Style GAN model and print its architecture. Choose a layer of interest to visualize and print its weights.

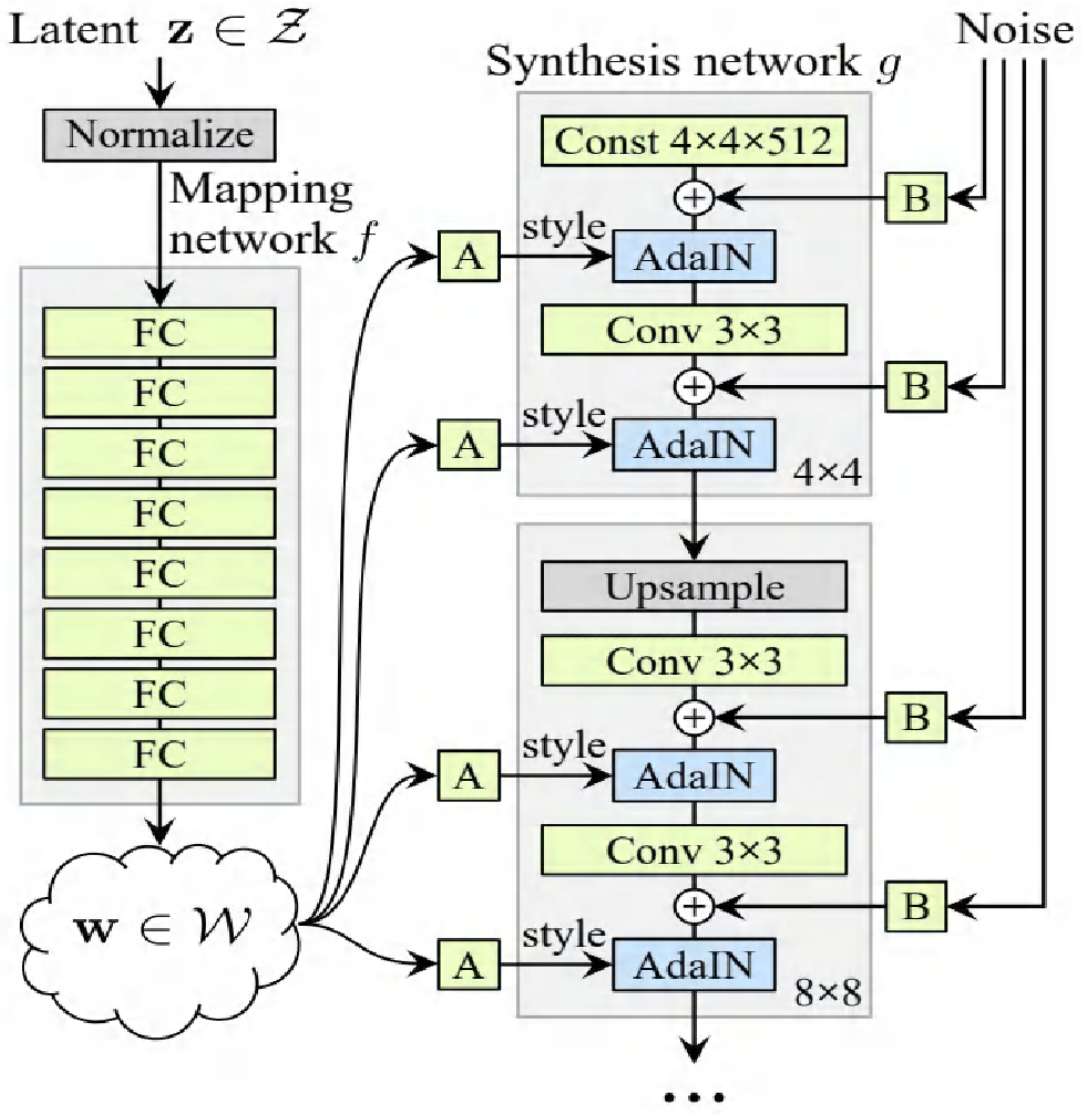

Figure 1: Architecture of Style GAN

ARCHITECTURE OF THE PRETRAINED GAN

Gs	Params	OutputShape	WeightShape
latents_in	-	(?, 512)	-
labels_in	-	(?, 0)	-
lod	-	()	-
dlatent_avg	-	(512,)	-
G_mapping/latents_in	-	(?, 512)	-
G_mapping/labels_in	-	(?, 0)	-
G_mapping/PixelNorm	-	(?, 512)	-
G_mapping/Dense0	262656	(?, 512)	(512, 512)
G_mapping/Dense1	262656	(?, 512)	(512, 512)
G_mapping/Dense2	262656	(?, 512)	(512, 512)
G_mapping/Dense3	262656	(?, 512)	(512, 512)
G_mapping/Dense4	262656	(?, 512)	(512, 512)
G_mapping/Dense5	262656	(?, 512)	(512, 512)
G_mapping/Dense6	262656	(?, 512)	(512, 512)
G_mapping/Dense7	262656	(?, 512)	(512, 512)
G_mapping/Broadcast	-	(?, 18, 512)	-
G_mapping/dlatents_out	-	(?, 18, 512)	-
Truncation	-	(?, 18, 512)	-
G_synthesis/dlatents_in	-	(?, 18, 512)	-
G_synthesis/4x4/Const	534528	(?, 512, 4, 4)	(512,)
G_synthesis/4x4/Conv	2885632	(?, 512, 4, 4)	(3, 3, 512, 512)
G_synthesis/ToRGB_lod8	1539	(?, 3, 4, 4)	(1, 1, 512, 3)
G_synthesis/8x8/Conv0_up	2885632	(?, 512, 8, 8)	(3, 3, 512, 512)
G_synthesis/8x8/Conv1	2885632	(?, 512, 8, 8)	(3, 3, 512, 512)
G_synthesis/ToRGB_lod7	1539	(?, 3, 8, 8)	(1, 1, 512, 3)
G_synthesis/Upscale2D	-	(?, 3, 8, 8)	-
G_synthesis/Grow_lod7	-	(?, 3, 8, 8)	-
G_synthesis/16x16/Conv0_up	2885632	(?, 512, 16, 16)	(3, 3, 512, 512)
G_synthesis/16x16/Conv1	2885632	(?, 512, 16, 16)	(3, 3, 512, 512)
G_synthesis/ToRGB_lod6	1539	(?, 3, 16, 16)	(1, 1, 512, 3)
G_synthesis/Upscale2D_1	-	(?, 3, 16, 16)	-
G_synthesis/Grow_lod6	-	(?, 3, 16, 16)	-
G_synthesis/32x32/Conv0_up	2885632	(?, 512, 32, 32)	(3, 3, 512, 512)
G_synthesis/32x32/Conv1	2885632	(?, 512, 32, 32)	(3, 3, 512, 512)
G_synthesis/ToRGB_lod5	1539	(?, 3, 32, 32)	(1, 1, 512, 3)
G_synthesis/Upscale2D_2	-	(?, 3, 32, 32)	-
G_synthesis/Grow_lod5	-	(?, 3, 32, 32)	-
G_synthesis/64x64/Conv0_up	1442816	(?, 256, 64, 64)	(3, 3, 512, 256)
G_synthesis/64x64/Conv1	852992	(?, 256, 64, 64)	(3, 3, 256, 256)
G_synthesis/ToRGB_lod4	771	(?, 3, 64, 64)	(1, 1, 256, 3)

G_synthesis/ToRGB_lod4	771	(?, 3, 64, 64)	(1, 1, 256, 3)
G_synthesis/Upscale2D_3	-	(?, 3, 64, 64)	-
G_synthesis/Grow_lod4	-	(?, 3, 64, 64)	-
G_synthesis/128x128/Conv0_up	426496	(?, 128, 128, 128)	(3, 3, 256, 128)
G_synthesis/128x128/Conv1	279040	(?, 128, 128, 128)	(3, 3, 128, 128)
G_synthesis/ToRGB_lod3	387	(?, 3, 128, 128)	(1, 1, 128, 3)
G_synthesis/Upscale2D_4	-	(?, 3, 128, 128)	-
G_synthesis/Grow_lod3	-	(?, 3, 128, 128)	-
G_synthesis/256x256/Conv0_up	139520	(?, 64, 256, 256)	(3, 3, 128, 64)
G_synthesis/256x256/Conv1	102656	(?, 64, 256, 256)	(3, 3, 64, 64)
G_synthesis/ToRGB_lod2	195	(?, 3, 256, 256)	(1, 1, 64, 3)
G_synthesis/Upscale2D_5	-	(?, 3, 256, 256)	-
G_synthesis/Grow_lod2	-	(?, 3, 256, 256)	-
G_synthesis/512x512/Conv0_up	51328	(?, 32, 512, 512)	(3, 3, 64, 32)
G_synthesis/512x512/Conv1	42112	(?, 32, 512, 512)	(3, 3, 32, 32)
G_synthesis/ToRGB_lod1	99	(?, 3, 512, 512)	(1, 1, 32, 3)
G_synthesis/Upscale2D_6	-	(?, 3, 512, 512)	-
G_synthesis/Grow_lod1	-	(?, 3, 512, 512)	-
G_synthesis/1024x1024/Conv0_up	21056	(?, 16, 1024, 1024)	(3, 3, 32, 16)
G_synthesis/1024x1024/Conv1	18752	(?, 16, 1024, 1024)	(3, 3, 16, 16)
G_synthesis/ToRGB_lod0	51	(?, 3, 1024, 1024)	(1, 1, 16, 3)
G_synthesis/Upscale2D_7	-	(?, 3, 1024, 1024)	-
G_synthesis/Grow_lod0	-	(?, 3, 1024, 1024)	-
G_synthesis/images_out	-	(?, 3, 1024, 1024)	-
G_synthesis/lod	-	()	-
G_synthesis/noise0	-	(1, 1, 4, 4)	-
G_synthesis/noise1	-	(1, 1, 4, 4)	-
G_synthesis/noise2	-	(1, 1, 8, 8)	-
G_synthesis/noise3	-	(1, 1, 8, 8)	-
G_synthesis/noise4	-	(1, 1, 16, 16)	-
G_synthesis/noise5	-	(1, 1, 16, 16)	-
G_synthesis/noise6	-	(1, 1, 32, 32)	-
G_synthesis/noise7	-	(1, 1, 32, 32)	-
G_synthesis/noise8	-	(1, 1, 64, 64)	-
G_synthesis/noise9	-	(1, 1, 64, 64)	-
G_synthesis/noise10	-	(1, 1, 128, 128)	-
G_synthesis/noise11	-	(1, 1, 128, 128)	-
G_synthesis/noise12	-	(1, 1, 256, 256)	-
G_synthesis/noise13	-	(1, 1, 256, 256)	-
G_synthesis/noise14	-	(1, 1, 512, 512)	-
G_synthesis/noise15	-	(1, 1, 512, 512)	-
G_synthesis/noise16	-	(1, 1, 1024, 1024)	-
G_synthesis/noise17	-	(1, 1, 1024, 1024)	-
images_out	-	(?, 3, 1024, 1024)	-
---	---	---	---
Total	26219627		

We have chosen the convolution layer of size as layer of interest.

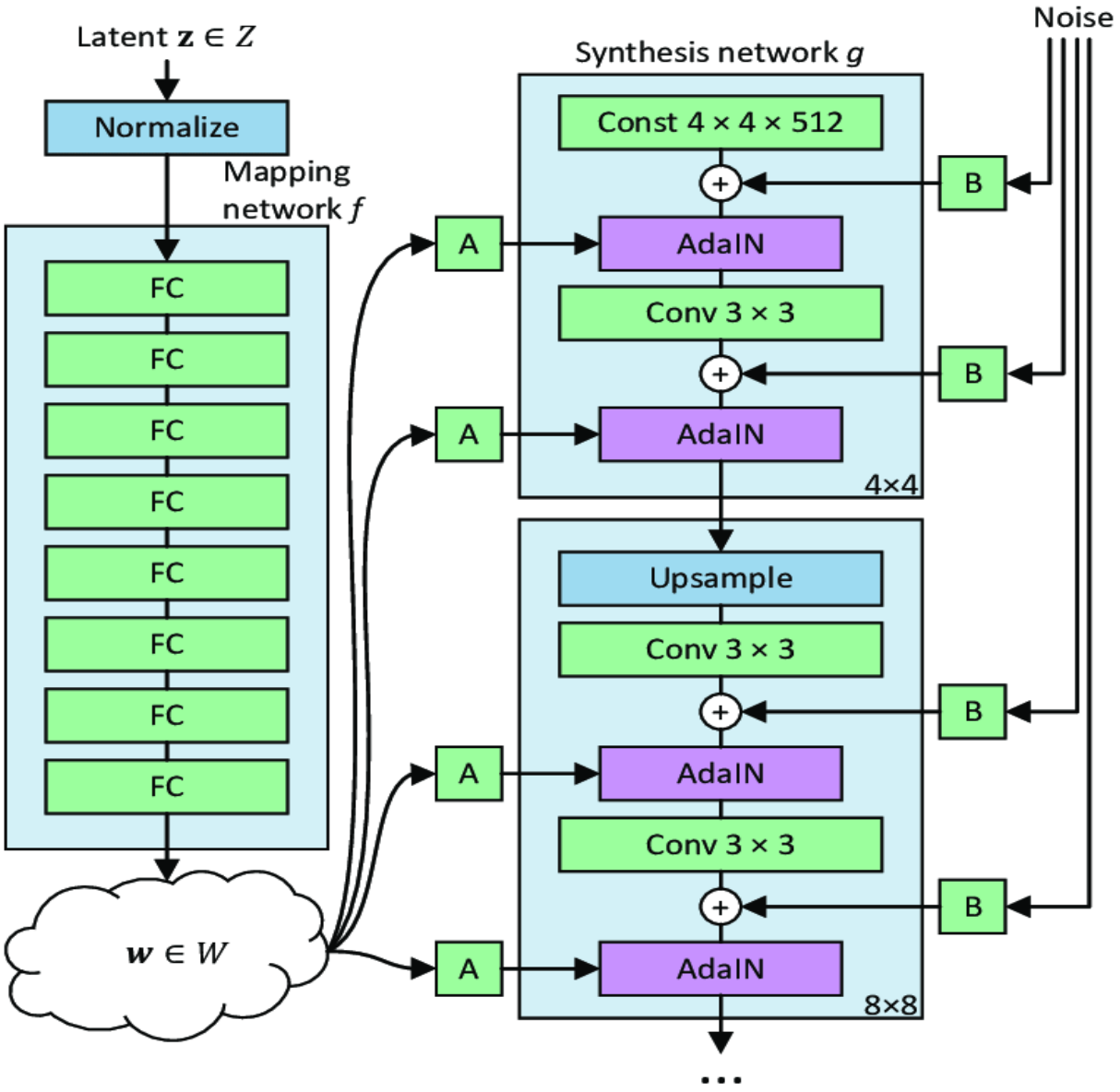

Figure 2: Convolution layer

The bias will be a zero vector of size (512,1) and the weight vector is of size (3x3x512x512). The weight values are shown:

```
[[[[-1.13293445e+00 -1.42998123e+00 -3.75741661e-01 ...  1.86936009e+00
    -1.14307813e-02 -6.07937455e-01]
   [-1.53560174e+00 -5.12830853e-01 -1.51990280e-01 ...  1.15078318e+00
     1.26041913e+00 -2.44475365e+00]
   [ 3.93578291e-01  1.53762609e-01 -2.36097053e-01 ... -3.88783902e-01
    -6.36849031e-02  1.03212881e+00]
   ...
   [ 5.79855800e-01  9.25494283e-02  2.54354268e-01 ...  2.04634070e+00
    -6.15697920e-01  1.44777882e+00]
   [ 4.31820720e-01 -1.05570488e-01 -6.24375522e-01 ...  9.23863649e-01
     6.50121748e-01  5.91381490e-01]
   [-6.62789028e-03 -1.30652413e-01  8.69403899e-01 ...  1.47768712e+00
     8.84303600e-02  1.94287229e+00]]

  [[-1.15202844e+00 -6.30349994e-01 -5.76063395e-02 ...  1.05690324e+00
     2.42998457e+00  1.21174419e+00]
   [ 4.64865230e-02 -4.74074930e-01  2.00367045e+00 ...  2.47359419e+00
     4.00459707e-01  3.02245855e-01]
   [ 1.62109077e-01 -3.67039710e-01 -1.08224249e+00 ...  8.34503949e-01
    -1.09260166e+00  6.52087808e-01]
   ...
   [-3.15792561e-01 -3.31971437e-01  9.05014753e-01 ...  3.70770305e-01
     5.52157938e-01  3.97688121e-01]
   [-7.88702667e-01 -7.64045477e-01  3.63406651e-02 ... -1.29433620e+00
    -6.18716955e-01  9.25750613e-01]
   [-3.96151721e-01 -5.52355826e-01  4.60120410e-01 ...  8.37685645e-01
    -7.86637962e-02 -9.48093176e-01]]

  [[-1.83778536e+00 -2.44626254e-01  7.37150371e-01 ...  1.04679537e+00
     3.07795376e-01 -3.21102548e+00]
   [ 7.99103856e-01 -1.09945841e-01  1.04090512e+00 ... -5.45099258e-01
     3.21904212e-01 -1.32914126e+00]
   [-7.82805324e-01  5.57125151e-01 -7.09319711e-01 ... -4.26857412e-01
     7.59805739e-01  9.07638490e-01]


  [-5.25732040e-01 -1.29253536e-01 -4.12041754e-01 ...  1.56785607e+00
   -1.70698607e+00 -1.23246515e+00]
  [-1.42957127e+00  2.61636376e-01  1.24358487e+00 ... -3.36405337e-01
   -2.82396197e-01  1.68672431e+00]
  [ 2.52353638e-01  3.41646999e-01 -5.60927801e-02 ...  2.37745449e-01
   -5.73104858e-01  1.66966927e+00]]]


[[[-2.57705426e+00 -9.32451248e-01 -8.03681970e-01 ...  1.40349936e+00
    -1.27844977e+00  8.05416256e-02]
  [-1.23940229e+00  1.22280276e+00 -9.16538298e-01 ... -2.72155583e-01
     2.98945636e-01  1.04267704e+00]
  [ 1.63007689e+00 -9.56796467e-01  5.51825427e-02 ...  2.12952793e-02
    -3.31056118e-01 -8.49066436e-01]
  ...
  [-3.04975733e-03 -4.27514553e-01  4.28357691e-01 ...  8.10699701e-01
     8.12802732e-01 -1.43254414e-01]
  [-9.25355375e-01 -5.53304315e-01 -1.01611733e+00 ... -1.78088081e+00
    -1.36055267e+00  1.24304474e-03]
  [-9.91553307e-01  4.15597022e-01  1.57744730e+00 ... -4.54828620e-01
    -1.04595292e+00 -1.75316021e-01]]

 [[ 6.17585599e-01  9.28259552e-01 -5.89361191e-01 ... -1.34954989e+00
    1.10532272e+00  4.98156577e-01]
  [ 5.18379271e-01 -1.22139001e+00 -2.42848560e-01 ...  1.14288354e+00
    1.50604099e-01 -1.39601707e+00]
  [-9.98146474e-01  5.24348855e-01  3.06943282e-02 ... -5.62321484e-01
   -7.30551958e-01  2.22616628e-01]
  ...
  [-1.13698936e+00  1.89175272e+00 -4.53648537e-01 ...  7.38220930e-01
    2.04561770e-01  7.54756987e-01]
  [ 1.86266458e+00 -5.34334004e-01 -1.41575289e+00 ...  8.02531004e-01
   -7.86405802e-01  2.84187734e-01]
  [ 3.16918612e-01  7.93926895e-01  3.06249547e+00 ... -3.96274954e-01
   -7.81009555e-01 -1.20478129e+00]]
```

```
[[-2.68476963e-01 -3.91779751e-01  3.79102856e-01 ... -1.38852763e+00
  -4.25549932e-02 -1.03380084e+00]
 [ 1.43209562e-01 -1.23533964e-01  1.77630648e-01 ...  5.03856182e-01
  -2.27858037e-01  5.95074832e-01]
 [ 1.88597322e+00  8.56635496e-02 -5.28705716e-01 ...  1.69027925e+00
   1.19478822e+00  1.78510696e-01]
 ...
 [ 1.28433466e+00 -8.18745345e-02  4.68607768e-02 ...  1.09599268e+00
   6.64497018e-01 -1.41541636e+00]
 [-5.06259859e-01  7.20252469e-03 -1.96748567e+00 ... -7.63501227e-01
   9.38682914e-01  1.08991313e+00]
 [ 1.02731757e-01 -2.09839869e+00 -1.32793438e+00 ...  5.95736839e-02
   2.94696093e-01  1.44113168e-01]]]


[[[-5.23059517e-02  1.55606544e+00 -8.71346518e-02 ... -2.30648392e-03
   5.85912615e-02 -1.10770249e+00]
 [ 8.19743216e-01 -5.21235883e-01  7.68026114e-01 ... -6.67652965e-01
   2.20995441e-01 -5.19856036e-01]
 [ 1.49993098e+00  5.83332896e-01 -7.30013311e-01 ...  1.28559303e+00
   1.45199956e-04 -3.03802550e-01]
 ...
 [-1.03089428e+00  1.12452745e+00  5.33390641e-01 ...  5.89440107e-01
  -6.23200536e-01 -1.07002270e+00]
 [ 7.04052925e-01 -1.13258727e-01 -2.08794296e-01 ... -1.06602466e+00
   2.97809899e-01  9.39938605e-01]
 [ 6.07638121e-01 -7.42266715e-01  5.21138728e-01 ... -1.42573047e+00
  -1.54778689e-01 -2.72655666e-01]]

 [[ 1.74916118e-01  5.95057189e-01 -1.05458069e+00 ... -1.46648943e+00
   6.09826922e-01 -7.93207347e-01]
 [ 2.11619481e-01  1.16026413e+00  7.69043863e-01 ... -5.47285080e-01
  -5.75001121e-01  4.62209612e-01]
 [-1.52315140e-01 -1.43007293e-01  4.03319210e-01 ... -1.13998461e+00
  -3.99465680e-01  2.10668349e+00]


  [ 1.31427169e+00  7.17671514e-01  1.13368638e-01 ... -1.37086332e+00
    2.33533934e-01 -3.01784903e-01]
  [ 1.25936520e+00  1.08806622e+00  1.03043401e+00 ... -4.98060524e-01
   -9.64573145e-01  3.43480319e-01]
  [ 4.69046608e-02  4.40481931e-01 -6.38667792e-02 ...  1.47246754e+00
    1.98029697e-01  1.16861367e+00]]

 [[-6.91668034e-01 -9.44666147e-01 -2.22902223e-01 ...  5.63651025e-01
   -1.67968833e+00  4.02209133e-01]
  [-1.27699423e+00  2.89387178e+00 -5.48827887e-01 ...  1.38898504e+00
    1.17784739e+00 -3.48731279e-01]
  [-1.41688263e+00  1.20067739e+00  3.79057169e-01 ...  1.28276825e+00
   -4.55409020e-01  7.57806599e-01]
  ...
  [ 1.99715823e-01  4.44640666e-01 -2.76653349e-01 ... -1.47913730e+00
    3.89691472e-01  1.34948239e-01]
  [ 6.64133608e-01 -1.09340322e+00  1.34910345e+00 ... -1.15093052e-01
   -1.03495610e+00  1.55428708e-01]
  [ 1.66217172e+00 -1.37466505e-01  7.65380681e-01 ...  1.28180349e+00
   -7.75153399e-01  4.76082206e-01]]]]
```

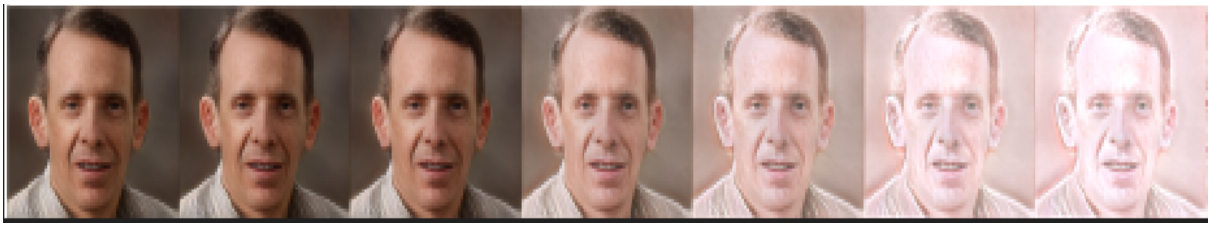

Figure 3: image generated from convolution layer

B)_Modify the code to perform style mixing between two randomly generated latent vectors at the chosen layer. Display the resulting images.

Figure 4: latent vector1

Figure 5: latent vector2

Figure 6: Images after style mixing

We have taken two latent vectors one a picture of a boy and another a picture of girl having dark complexion. When we mix the latent at the hidden layer, and generate the final image, it contains mix of features from both the images. For ex. the image generated after style mixing is a boy having dark complexion it inherits the feature of boy from image 1 and dark complexion from image 2.

C) Choose another layer of interest and visualize its weights. Explain any differences or similarities with the weights of the previous layer.

Noise Layer: Noise is added to the feature maps at each layer of the synthesis network to add diversity and randomness to the generated images.

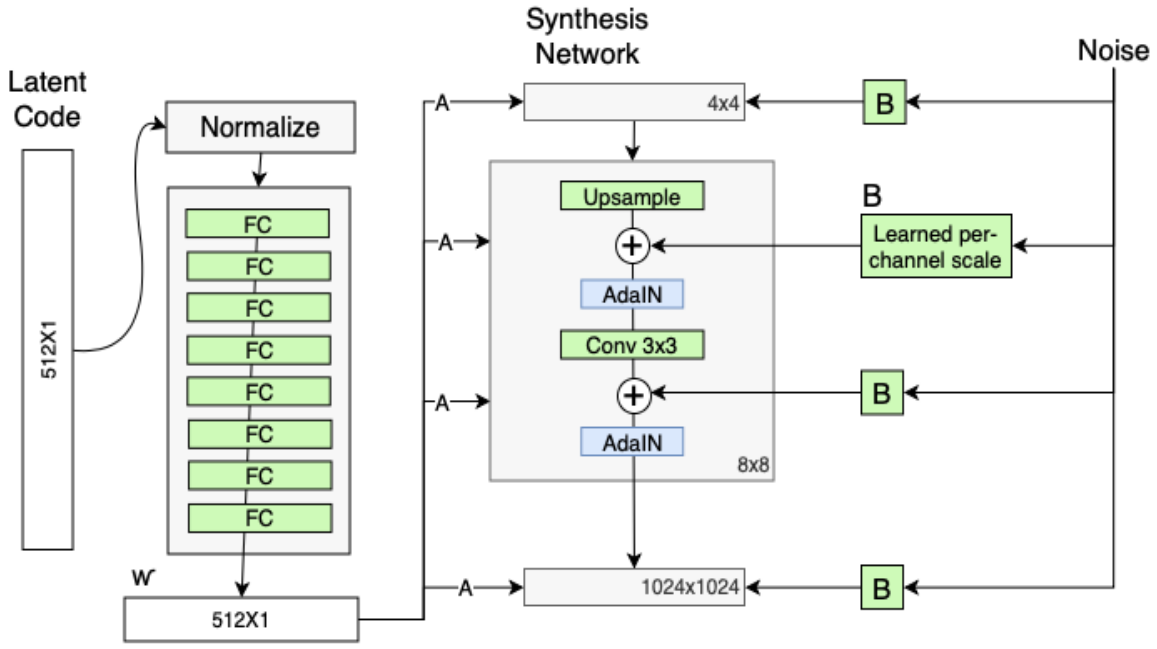

Figure 7: Noise layer

Weights of the noise layer of size (1x1x4x4)

```
[[[[-1.3416071  -1.0006171  -0.56084055  1.0860376 ]
   [ 0.88983756  0.08972804  0.17255172  0.9625934 ]
   [-0.79426926  0.5256566   1.0282614   0.1510227 ]
   [ 2.294197   -0.6398764  -1.042719    0.30641073]]]]
```

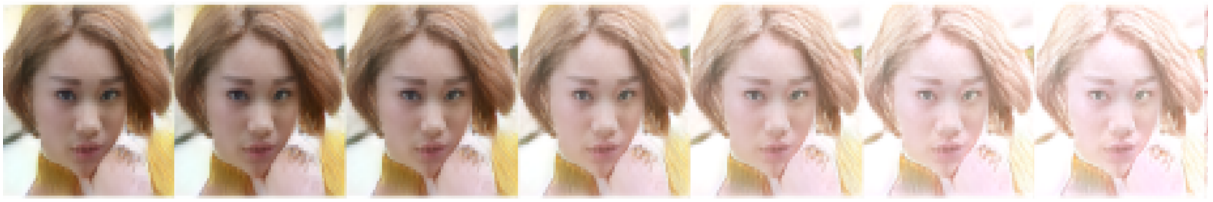

Figure 8: image generated from noise layer

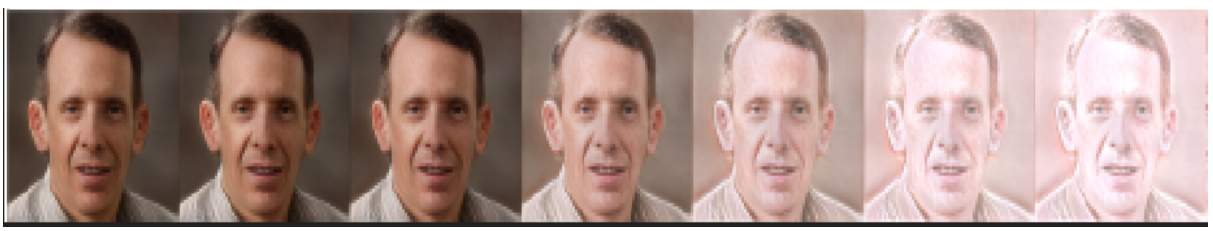

Figure 9: image generated from convolution layer

Figure 8 depicts the image generated from the noise layer and figure 9 depicts the image generated from the convolution layer. We can see that from the both layers the images generated stated faded after few runs. The weights generated from convolution layer are much stronger than the noise layer. The convolution layer captures more features than the noise layer.

D) Explain the concept of style mixing in Style GAN. How does it differ from traditional style transfer methods? How does it affect the generated images?

Style mixing is a technique used in Style GAN, a type of Generative Adversarial Network (GAN), to generate high-quality synthetic images. It involves combining the style vectors of two or more different images to create a new image with a combination of their styles.

In Style GAN, the generator network is divided into two parts: the mapping network and the synthesis network. The mapping network takes a random noise vector and transforms it into a style vector that is used by the synthesis network to generate an image. The synthesis network takes the style vector and generates the image.

Style mixing involves generating two or more random style vectors, each corresponding to a different level of abstraction in the network, and then using them to generate an image. Specifically, the network is designed in a way that allows for the mixing of style vectors at different resolutions, creating a range of possible outputs.

By mixing styles, Style GAN can generate images that combine the characteristics of two or more different images. This technique allows for greater control over the generation process and makes it possible to create more diverse and interesting images. Additionally, style mixing enables the exploration of the latent space in a more nuanced and flexible way, making it easier to create images that exhibit specific characteristics or styles.

Style mixing is a technique in generative models for creating images that combines the styles of multiple reference images in a more controlled manner than traditional style transfer methods. Traditional style transfer methods typically use a pre-trained convolutional neural network to extract style features from a single style image, which are then transferred to a content image to create a stylized image.

Figure 10: Images generated after style mixing

Figure 11: Images generated after traditional mixing

Figure 10 depicts the images generated after style mixing and Figure 11 depicts the images generated after traditional mixing. From the above figures we can observed that the images generated by style mixing are much better than the traditional mixing.

E) How does the depth of the chosen layer affect the quality of the generated images in style mixing? Why?

In style mixing, the depth of the chosen layer can have a significant impact on the quality of the generated images. The depth of the layer determines the level of abstraction and the spatial scale of the features that the generator network can capture and manipulate.

Choosing a layer that is too shallow may limit the generator's ability to capture high-level features, resulting in generated images that are too detailed and lack coherence. On the other hand, choosing a layer that is too deep may limit the generator's ability to capture low-level features, resulting in generated images that are too abstract and lack detail.

In style mixing, the goal is to combine the style of one image with the content of another image by manipulating the latent space. By choosing a layer that strikes a balance between capturing both high-level and low-level features, the generator can produce high-quality images with a good mix of style and content.

In practice, the optimal layer depth for style mixing can vary depending on the specific generator architecture and the dataset used. Empirical experiments and iterative refinement are often required to determine the optimal layer depth for a particular task.

F) Modify the code to perform style mixing with more than two latent vectors. Display the resulting images.

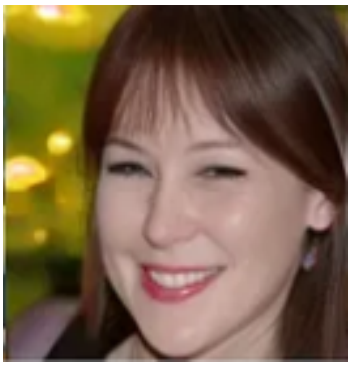

Figure 12: latent vector1

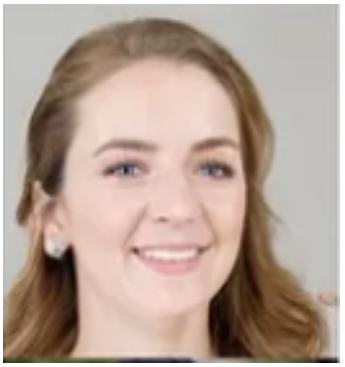

Figure13: latent vector2

Figure 14: latent vector3

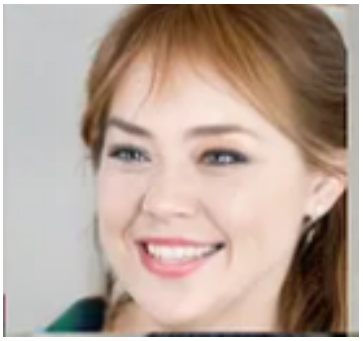

Figure 15: image generated after style-mixing

In this part we have taken three latent vectors and done style mixing. When we mix the latent at the hidden layer, and generate the final image, it contains mix of features from all the images. The image generated after style mixing has features from all three latent vectors which can be seen from the image generated after style mixing.

PART 2

CIFAR 10 Dataset

The CIFAR-10 dataset (Canadian Institute For Advanced Research) is a collection of images that are commonly used to train machine learning and computer vision algorithms. It is one of the most widely used datasets for machine learning research. The CIFAR-10 dataset contains 60,000 32x32 color images in 10 different classes. The 10 different classes represent airplanes, cars, birds, cats, deer, dogs, frogs, horses, ships, and trucks. There are 6,000 images of each class.

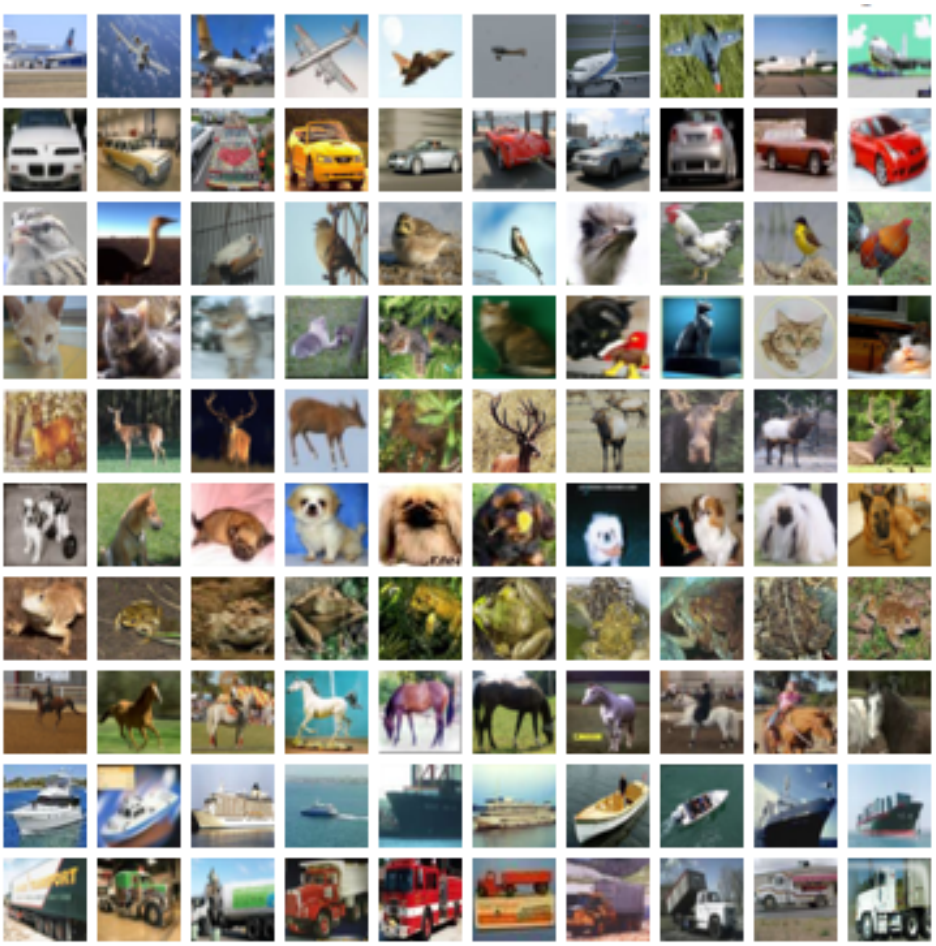

Figure 16: CIFAR10 Dataset

Model Training

In this question, we make CNN models to make GANs to generate images mimicking CIFAR 10 images. We do our experiments with multiple learning rates, while keeping model architecture same in all the models. Both the generator and discriminator models have 5 layers in them. We use Adam optimizer with beta=0.5, a batch size of 128 and a latent size of 100 for the GAN. We do our experiments with the learning rates 0.01, 0.001, 0.0001, 0.00001, and 0.000001.

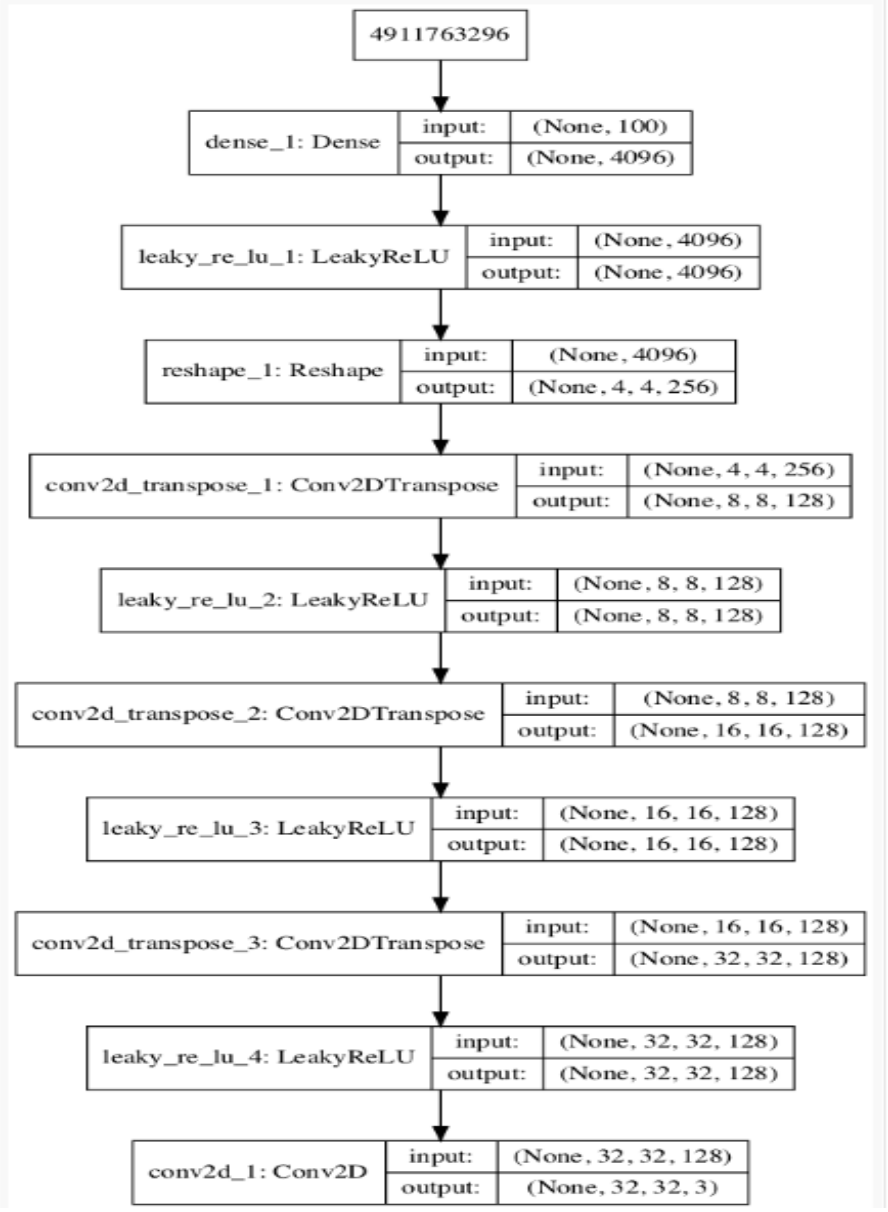

Figure 17: Plot of Generator model in CIFAR10 GAN

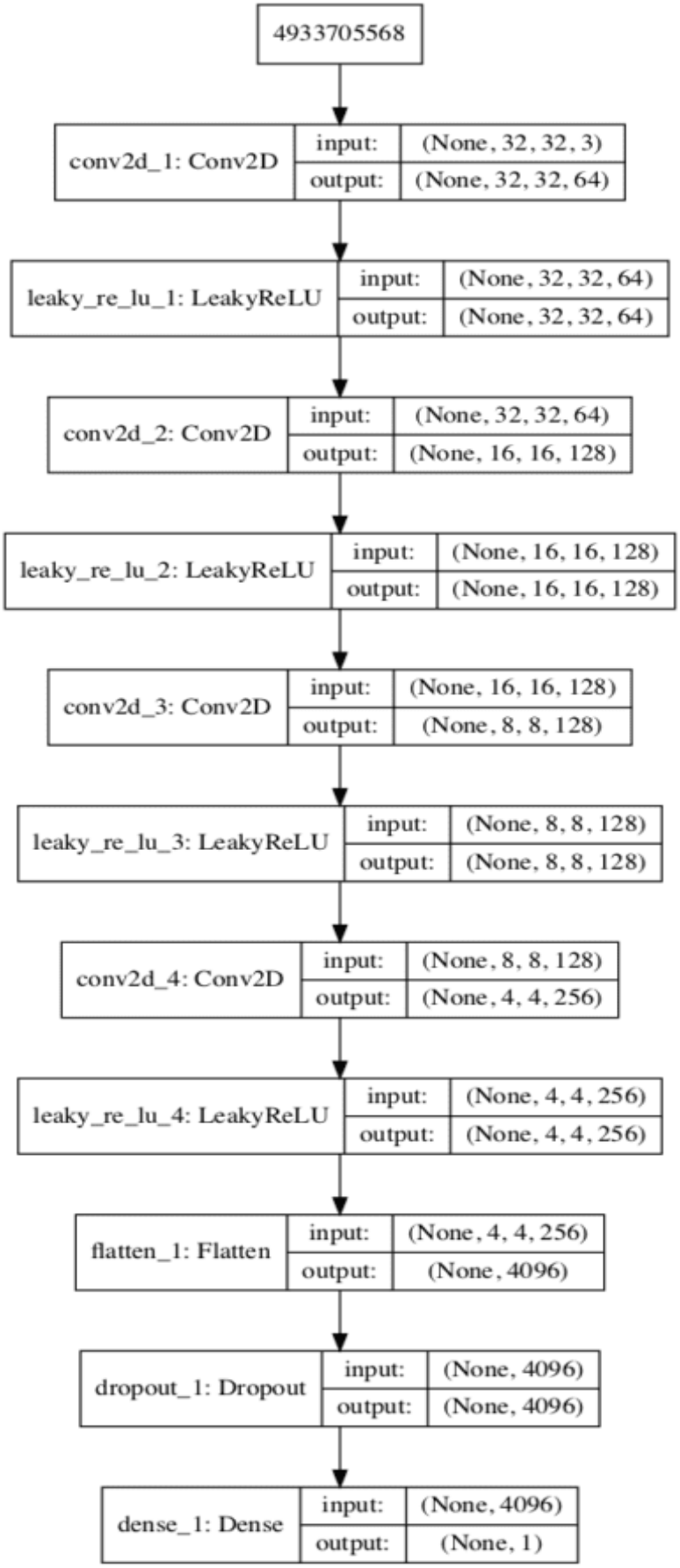

Figure 18: Plot of Discriminator model in CIFAR10 GAN

For choosing the learning rate, we first plot the generator and discriminator model losses for all the learning rates for 20 epochs. We get the following results.

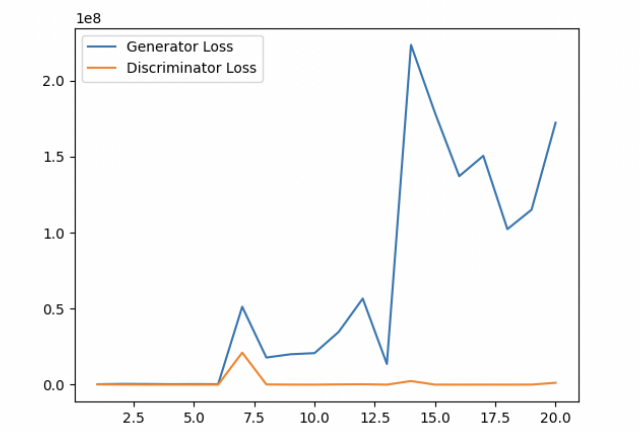

Figure 19: Learning Rate = 0.01

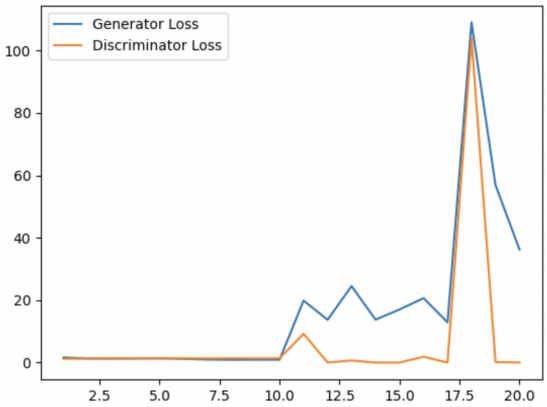

Figure 20: Learning Rate = 0.001

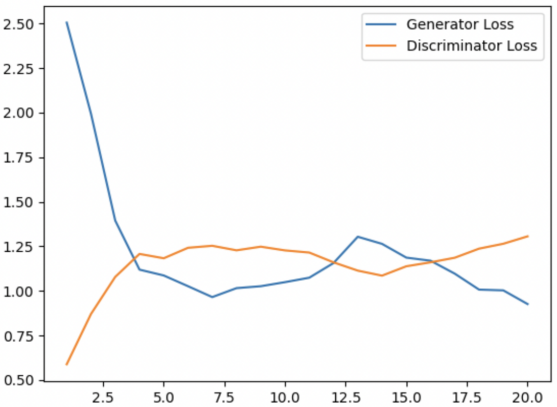

Figure 21: Learning Rate = 0.0001

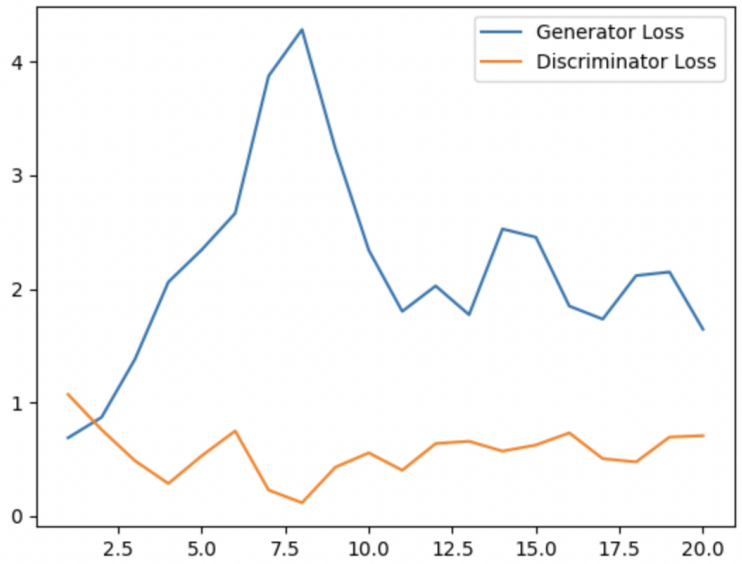

Figure 22: Learning Rate = 0.00001

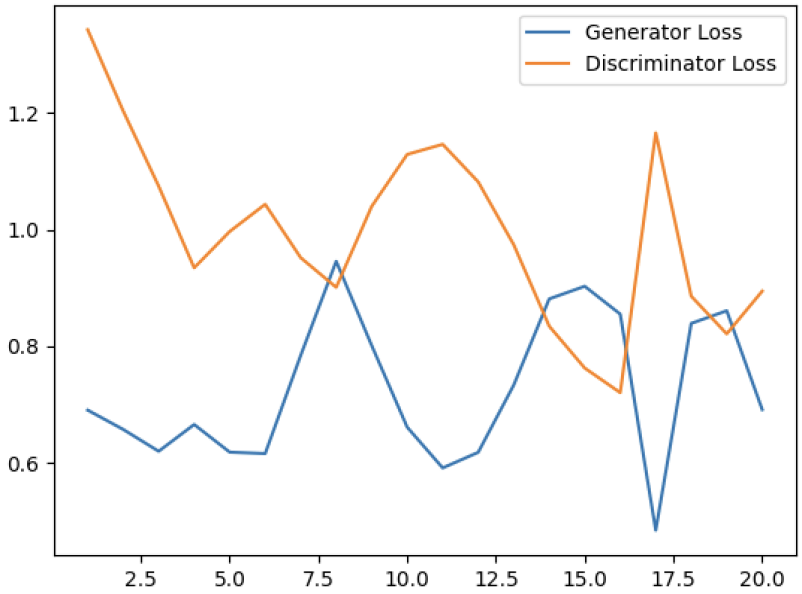

Figure 23: Learning Rate = 0.000001

From the results above, we can see that for learning rate = 0.0001, the generator and discriminator losses have stabilized around the middle of the graph. For all the other plots, the losses are not stable, and we can expect that the GAN models are not being trained effectively for these learning rates.

We also plot some outputs of the generator at different stages of GAN training.

Stage 1 (0 Epochs)

At the beginning, since none of the models are trained, all the GAN networks only generate gray images. Hence, the output for all the models are the same:

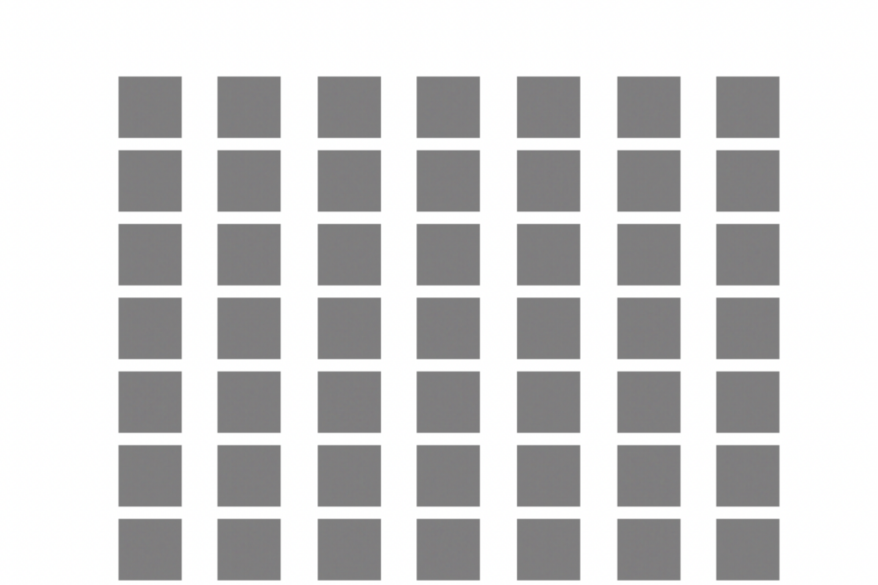

Figure 24: For all learning rates

Stage 2 (10 Epochs)

After training for 10 epochs, the differences in all the GANs start becoming more apparent.

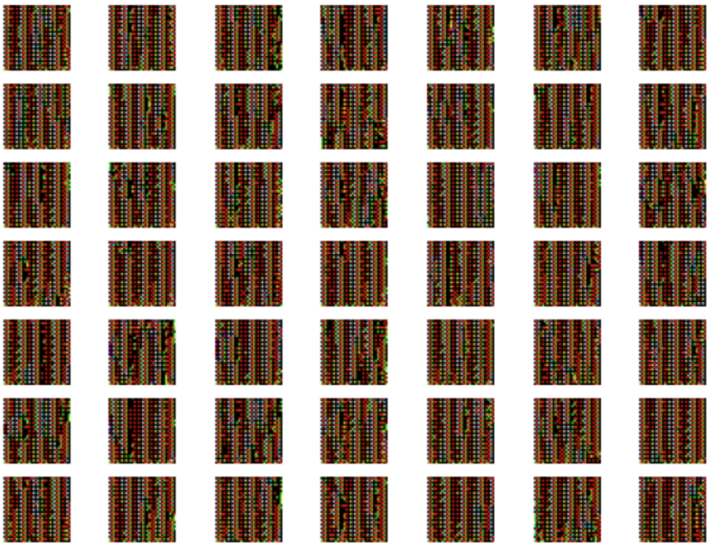

Figure 25: Learning Rate = 0.01

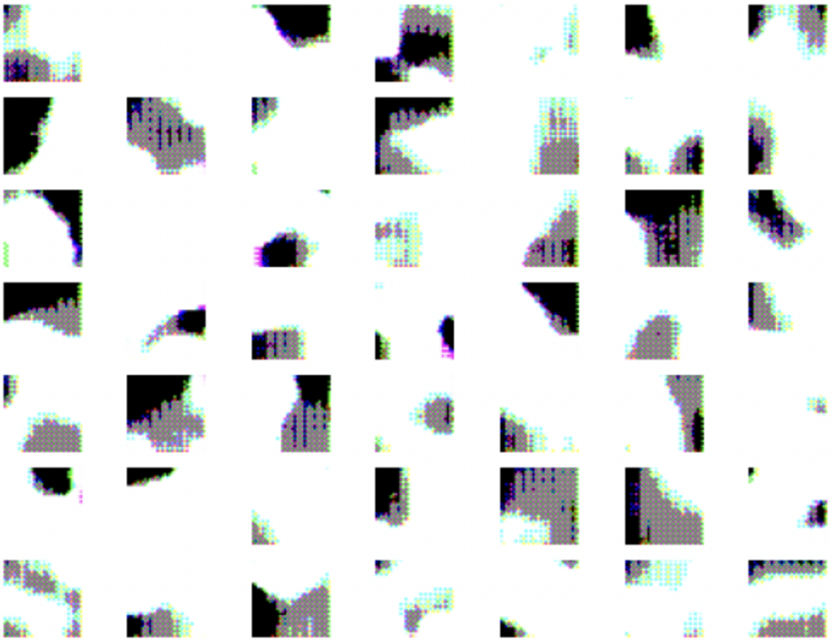

Figure 26: Learning Rate = 0.001

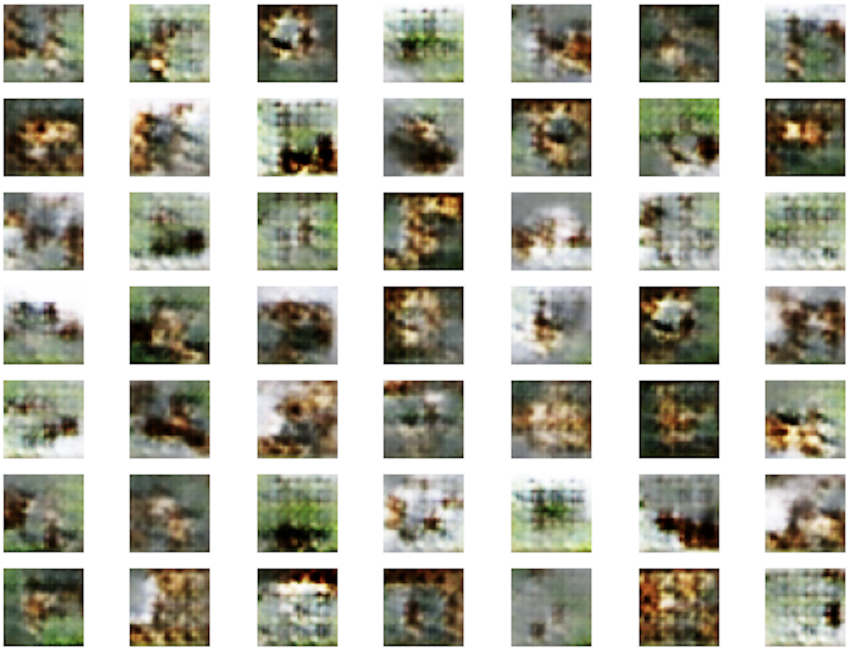

Figure 27: Learning Rate = 0.0001

Figure 28: Learning Rate = 0.00001

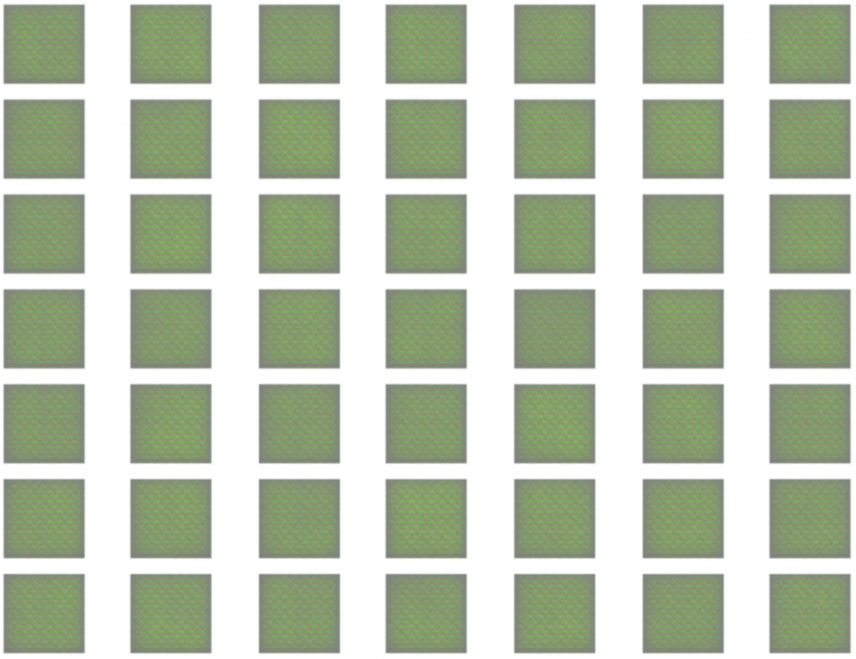

Figure 29: Learning Rate = 0.000001

Stage 3 (20 Epochs)

After complete training, we find that learning rate = 0.0001 gives the best outputs for mimicking the images of CIFAR 10.

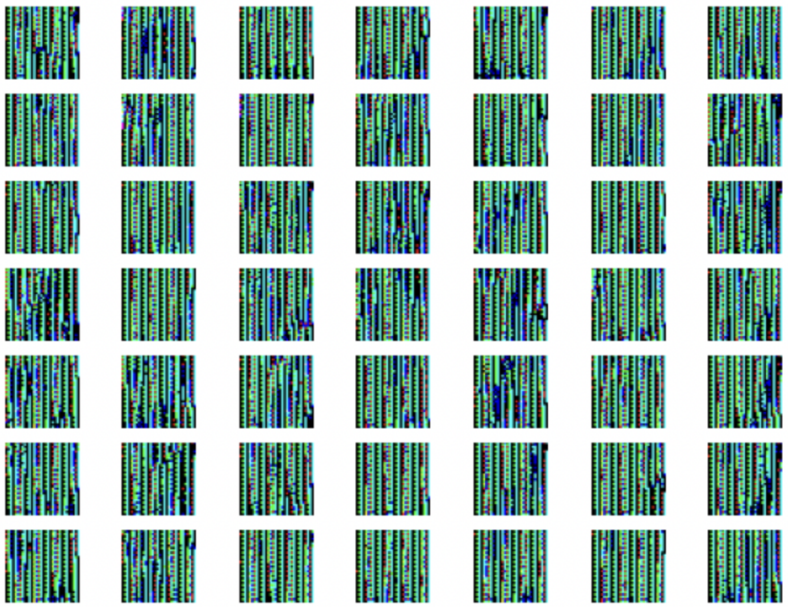

Figure 30: Learning Rate = 0.01

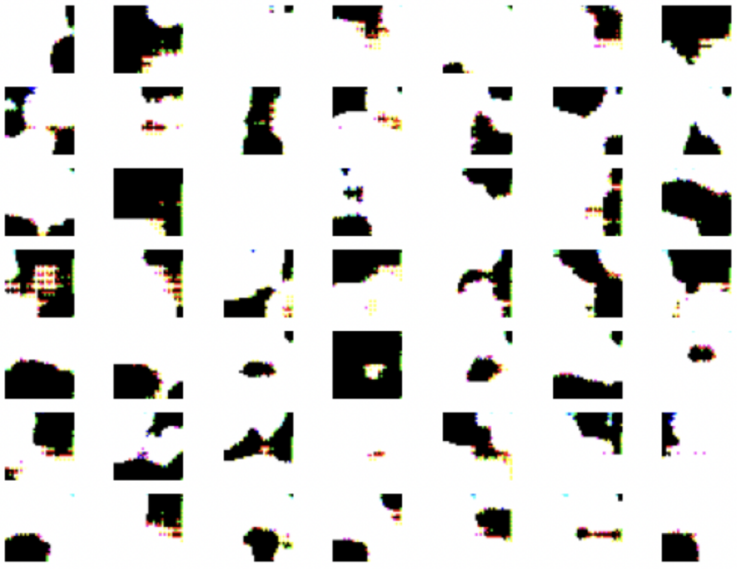

Figure 31: Learning Rate = 0.001

Figure 32: Learning Rate = 0.0001

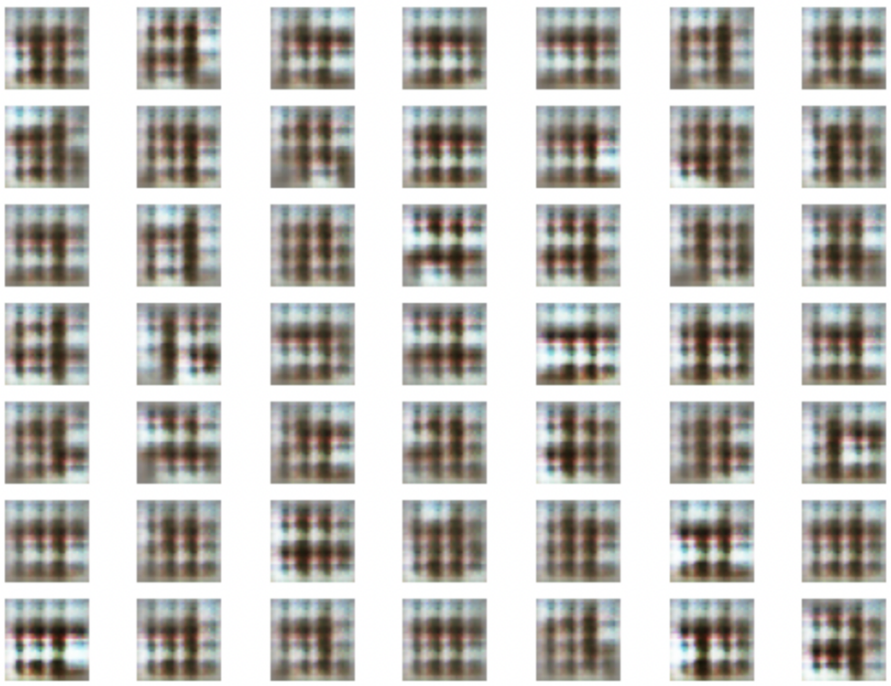

Figure 33: Learning Rate = 0.00001

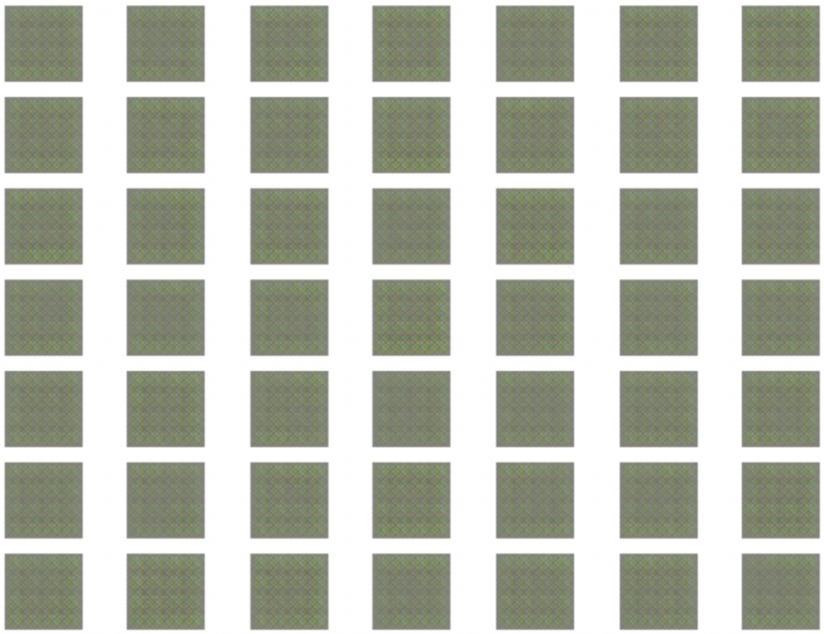

Figure 34: Learning Rate = 0.000001

Final Results with Learning Rate = 0.0001

After finding out that learning rate = 0.001 works the best, we run the GAN model for this learning rate for 40 epochs to see the final results. Here are the results:

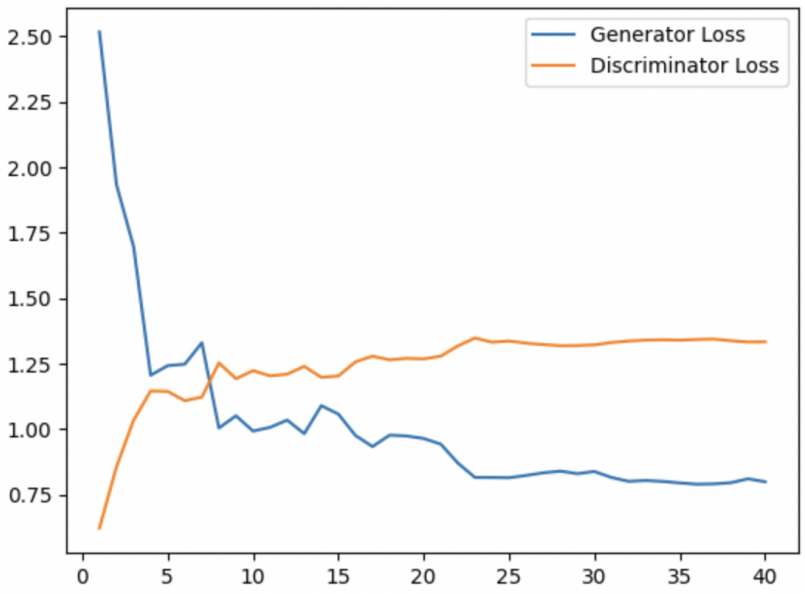

Figure 35: Generator and Discriminator loss for learning rate = 0.0001

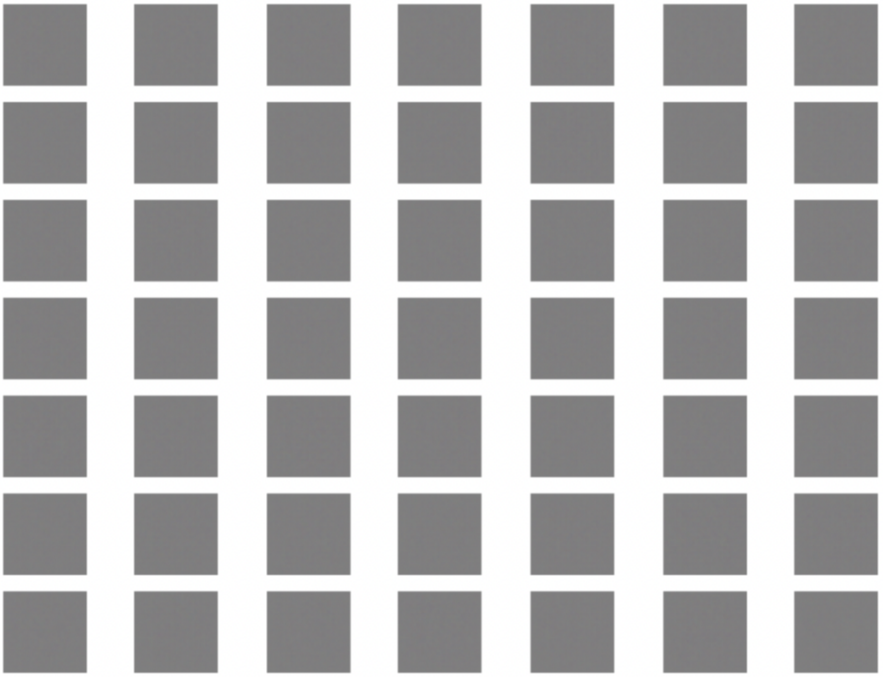

Figure 36: Generator outputs after training 0 epochs for learning rate = 0.0001

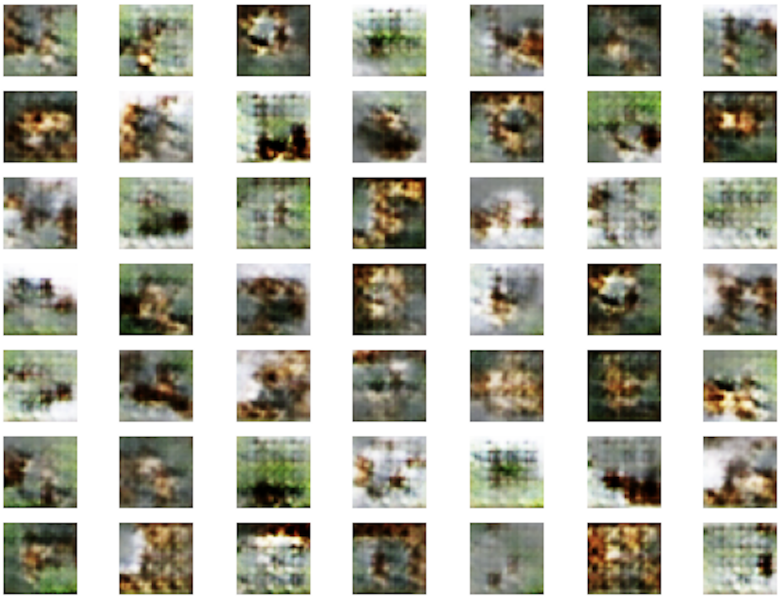

Figure 37: Generator outputs after training 10 epochs for learning rate = 0.0001

Figure 38: Generator outputs after training 20 epochs for learning rate = 0.0001

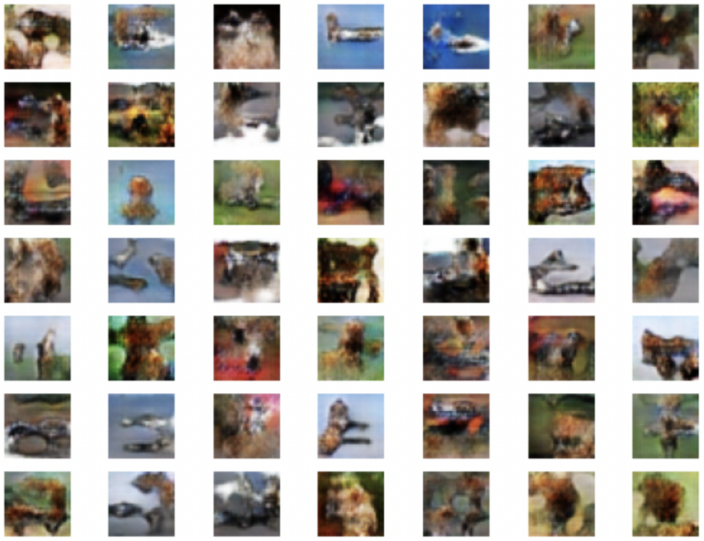

Figure 39: Generator outputs after training 40 epochs for learning rate = 0.0001

CONCLUSION

We observe that learning rate has a huge impact on training Generative Adversarial Networks. For us only the learning rate of 0.0001 gave good results. We saw that in the beginning of training the model, the model wasn't good at generating good images similar to CIFAR 10, but as training continued, by epoch 10, the generator started getting better and but epoch 20, it started generating images resembling the images of CIFAR 10. The performance was best with 40 epochs, till where we ran the experiments.