CSE 676: Deep Learning
Project1 Report

Sampreeth Reddy Boddi Reddy (50298591)

sboddire@buffalo.edu

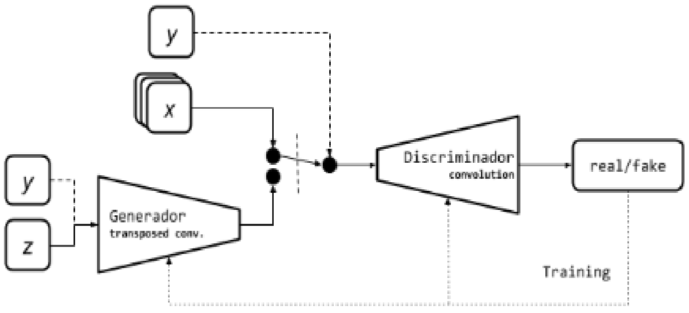
Deep convolutional generative adversarial networks(DCGAN):

Generator:

The Generator network is able to take random noise and map it into images such that the discriminator cannot tell which images came from the dataset and which images came from the generator.

Discriminator:

The discriminator, which learns how to distinguish fake from real objects of the type we'd like to create



Source: google

Cost Function:

$$\min_{G} \max_{D} V_{CGAN}(D, G) = \mathbb{E}_{x \sim p_{data}(x)}[log D(x|y)] + \mathbb{E}_{z \sim p_{z}(z)}[log(1 - D(G(z|y)))]$$

- **Data**
  - Rescale the CIFAR-10 images to be between -1 and 1.
- **Generator**
  - Use the inverse of convolution, called transposed convolution.
  - ReLU activation and Batch Normalization.
  - The last activation is tanh.
- **Discriminator**
  - Use the Convolutional neural network.
  - LeakyReLU activation and Batch Normalization.
  - The last activation is sigmoid.

- **Loss**
  - binary_crossentropy

- **Optimizer**
  - Adam
- batch size = 64
- epochs = 200

Implemented DCGAN as follows according to paper:

- No pre-processing was applied to training images besides scaling to the range of the tanh activation function [-1, 1].
- All models were trained with a mini-batch size of 64.
- In the LeakyReLU, the slope of the leak was set to 0.2 in all models.
- we used the Adam optimizer with tuned hyper-parameters. We found the suggested learning rate of 0.001, to be too high, using 0.0002 instead.
- Additionally, we added the momentum term at the suggested value of 0.9
- Replace any pooling layers with strided convolutions (discriminator) and fractional-strided convolutions (generator).
- Use batch norm in both the generator and the discriminator.
- Use ReLU activation in generator for all layers except for the output, which uses Tanh.
- Use LeakyReLU activation in the discriminator for all layers.

Environment Used to Execute:

GOOGLE COLAB

Generated Images can be viewed in below google drive shared link:
DCGAN:
https://drive.google.com/open?id=1SZdh-qGHQxtviZmm4WLVOvgutLZdfYiN

https://drive.google.com/open?id=1-8NjbGet9FpIG06rTxRgRi-2qswr0OMz

cSNSAGAN:
https://drive.google.com/open?id=1-5tSX5nZmPGdSxCh6zPcjTILKZRQ-5-0

https://drive.google.com/open?id=18wyVobkgwDU2TUYRcET4R4uAe368W5ai

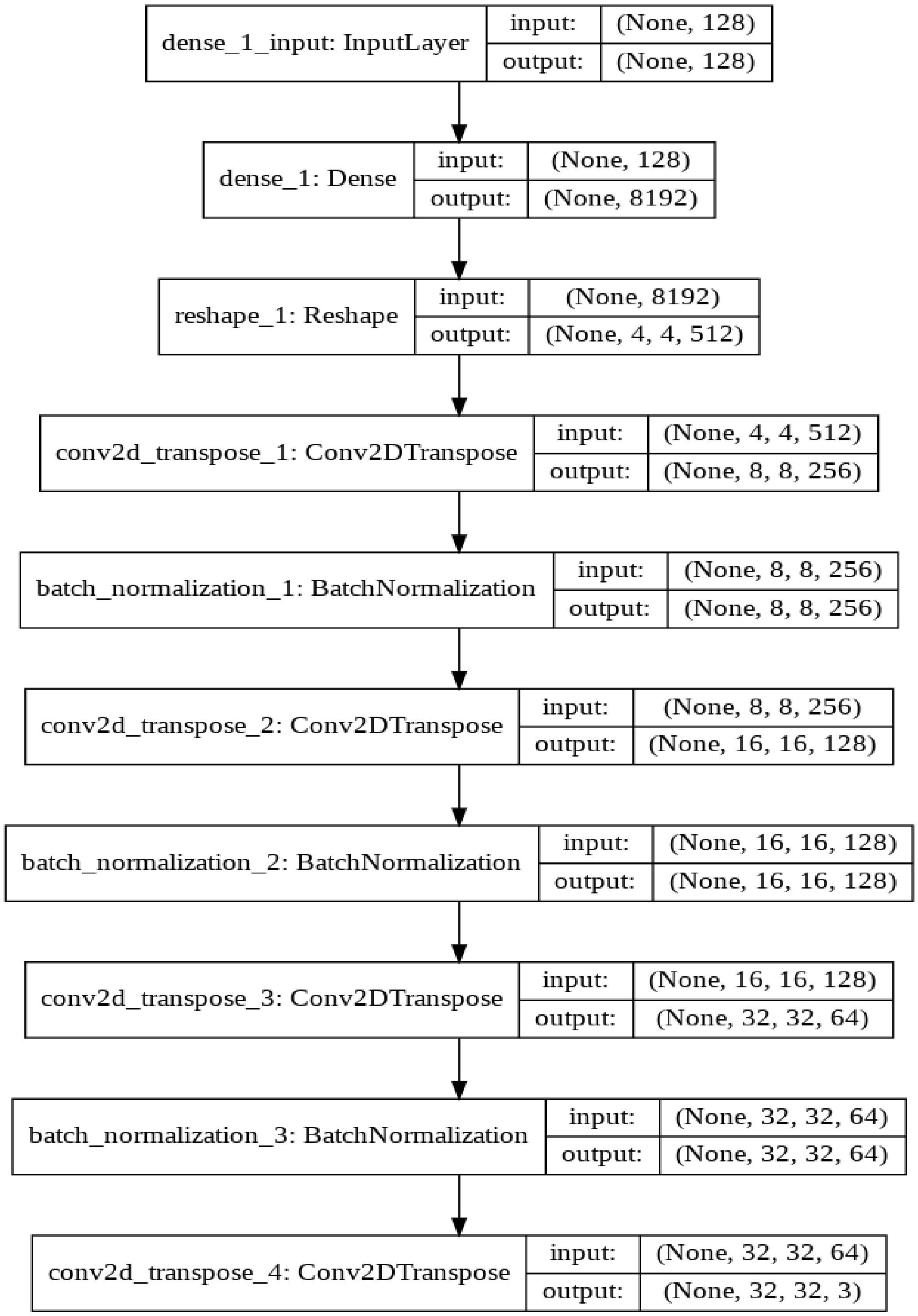
CSNSAGAN Model Weights:

modelsgenerator_model_weights_.h5 modelsdiscriminator_model_weights.h5 modelsgan_model_weights.h5

DCGAN Model Weights:

modelgenerator_model_weights.h5 modeldiscriminator_model_weights.h5

Implemented Generator Model:



dense_1_input: InputLayer
input: (None, 128)
output: (None, 128)
dense_1: Dense
input: (None, 128)
output: (None, 8192)
reshape_1: Reshape
input: (None, 8192)
output: (None, 4, 4, 512)
conv2d_transpose_1: Conv2DTranspose
input: (None, 4, 4, 512)
output: (None, 8, 8, 256)
batch_normalization_1: BatchNormalization
input: (None, 8, 8, 256)
output: (None, 8, 8, 256)
conv2d_transpose_2: Conv2DTranspose
input: (None, 8, 8, 256)
output: (None, 16, 16, 128)
batch_normalization_2: BatchNormalization
input: (None, 16, 16, 128)
output: (None, 16, 16, 128)
conv2d_transpose_3: Conv2DTranspose
input: (None, 16, 16, 128)
output: (None, 32, 32, 64)
batch_normalization_3: BatchNormalization
input: (None, 32, 32, 64)
output: (None, 32, 32, 64)
conv2d_transpose_4: Conv2DTranspose
input: (None, 32, 32, 64)
output: (None, 32, 32, 3)

Fig: DCGAN Generator Model

Implemented Discriminator Model:

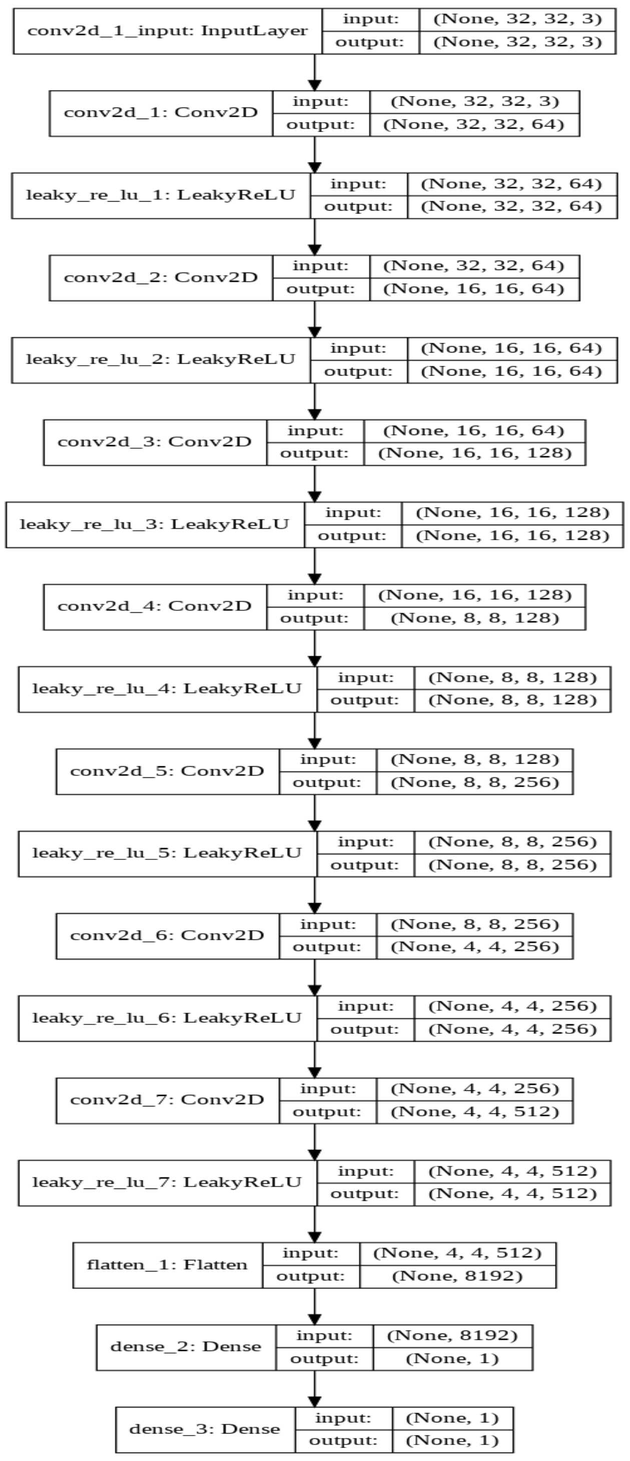
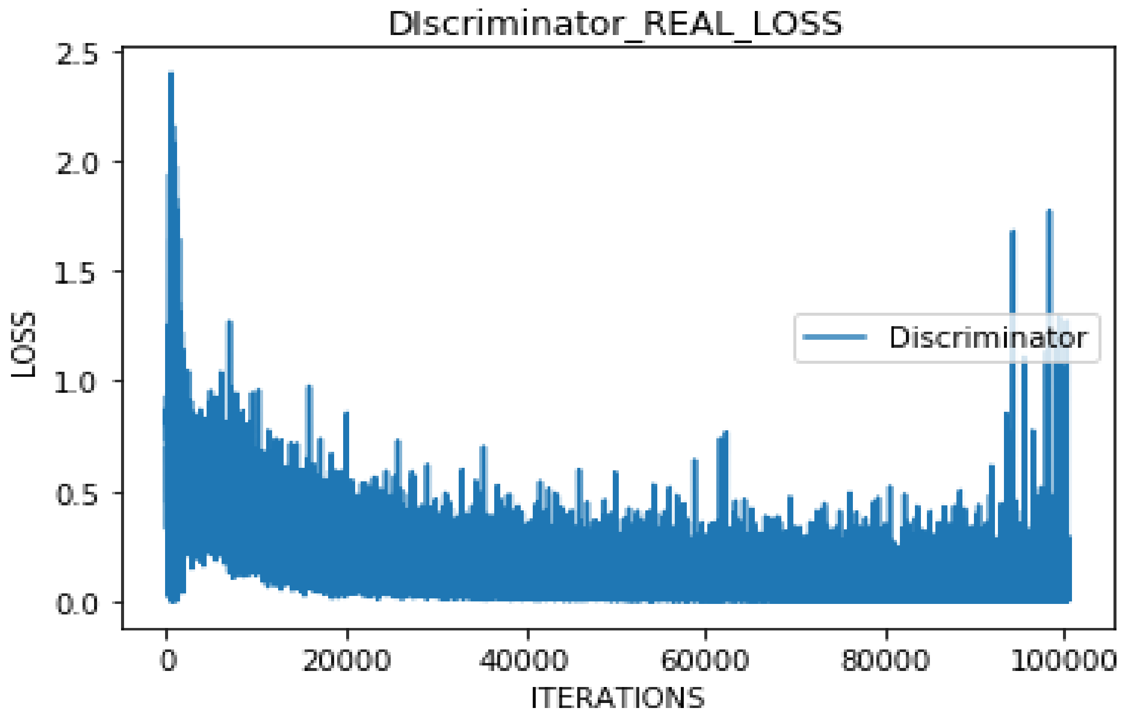
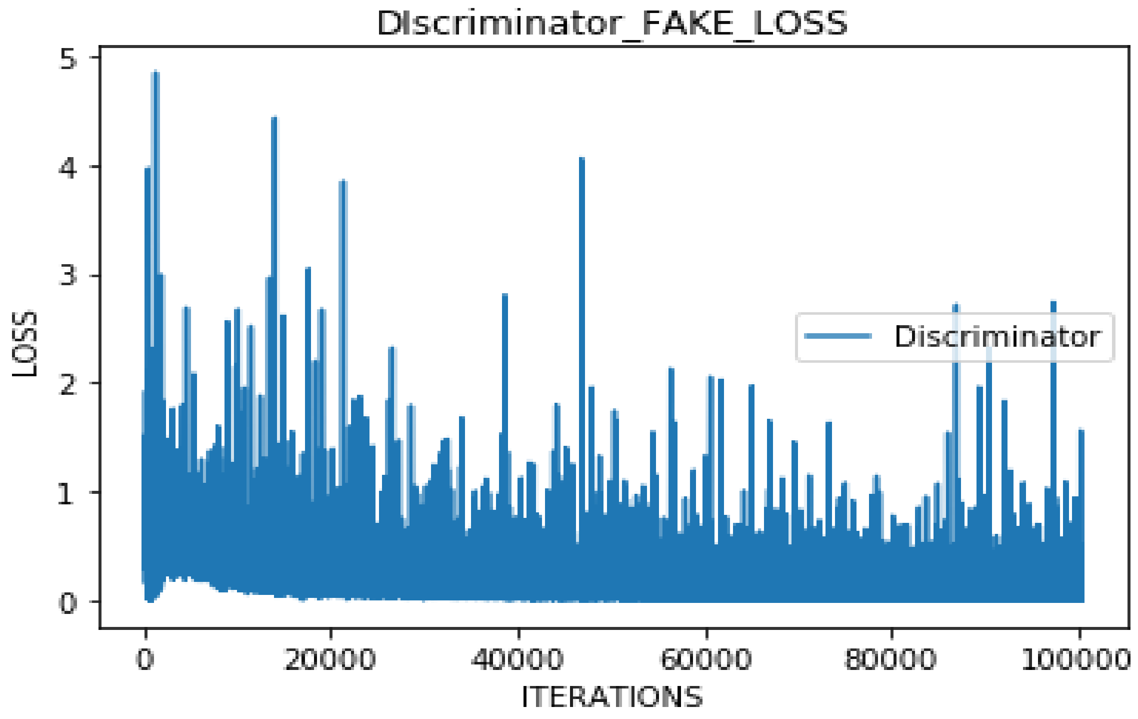
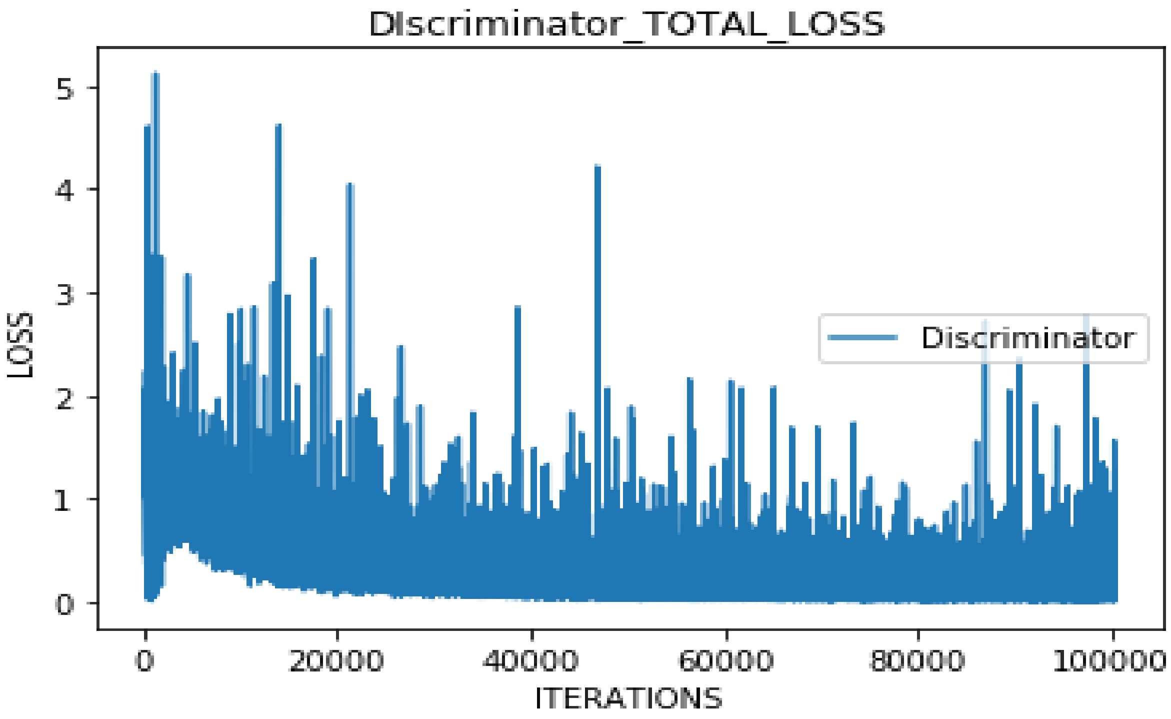

conv2d_1_input: InputLayer | input: (None, 32, 32, 3) | output: (None, 32, 32, 3)
conv2d_1: Conv2D | input: (None, 32, 32, 3) | output: (None, 32, 32, 64)
leaky_re_lu_1: LeakyReLU | input: (None, 32, 32, 64) | output: (None, 32, 32, 64)
conv2d_2: Conv2D | input: (None, 32, 32, 64) | output: (None, 16, 16, 64)
leaky_re_lu_2: LeakyReLU | input: (None, 16, 16, 64) | output: (None, 16, 16, 64)
conv2d_3: Conv2D | input: (None, 16, 16, 64) | output: (None, 16, 16, 128)
leaky_re_lu_3: LeakyReLU | input: (None, 16, 16, 128) | output: (None, 16, 16, 128)
conv2d_4: Conv2D | input: (None, 16, 16, 128) | output: (None, 8, 8, 128)
leaky_re_lu_4: LeakyReLU | input: (None, 8, 8, 128) | output: (None, 8, 8, 128)
conv2d_5: Conv2D | input: (None, 8, 8, 128) | output: (None, 8, 8, 256)
leaky_re_lu_5: LeakyReLU | input: (None, 8, 8, 256) | output: (None, 8, 8, 256)
conv2d_6: Conv2D | input: (None, 8, 8, 256) | output: (None, 4, 4, 256)
leaky_re_lu_6: LeakyReLU | input: (None, 4, 4, 256) | output: (None, 4, 4, 256)
conv2d_7: Conv2D | input: (None, 4, 4, 256) | output: (None, 4, 4, 512)
leaky_re_lu_7: LeakyReLU | input: (None, 4, 4, 512) | output: (None, 4, 4, 512)
flatten_1: Flatten | input: (None, 4, 4, 512) | output: (None, 8192)
dense_2: Dense | input: (None, 8192) | output: (None, 1)
dense_3: Dense | input: (None, 1) | output: (None, 1)

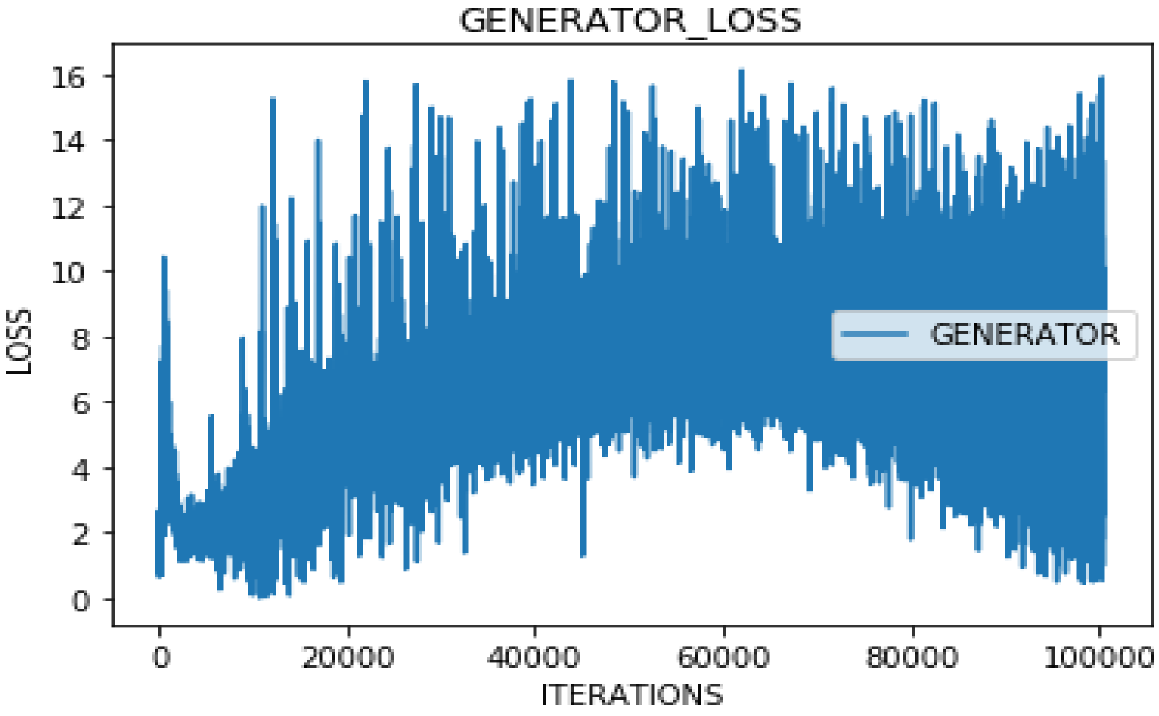
Fig: DCGAN Discriminator Model

Loss plots:

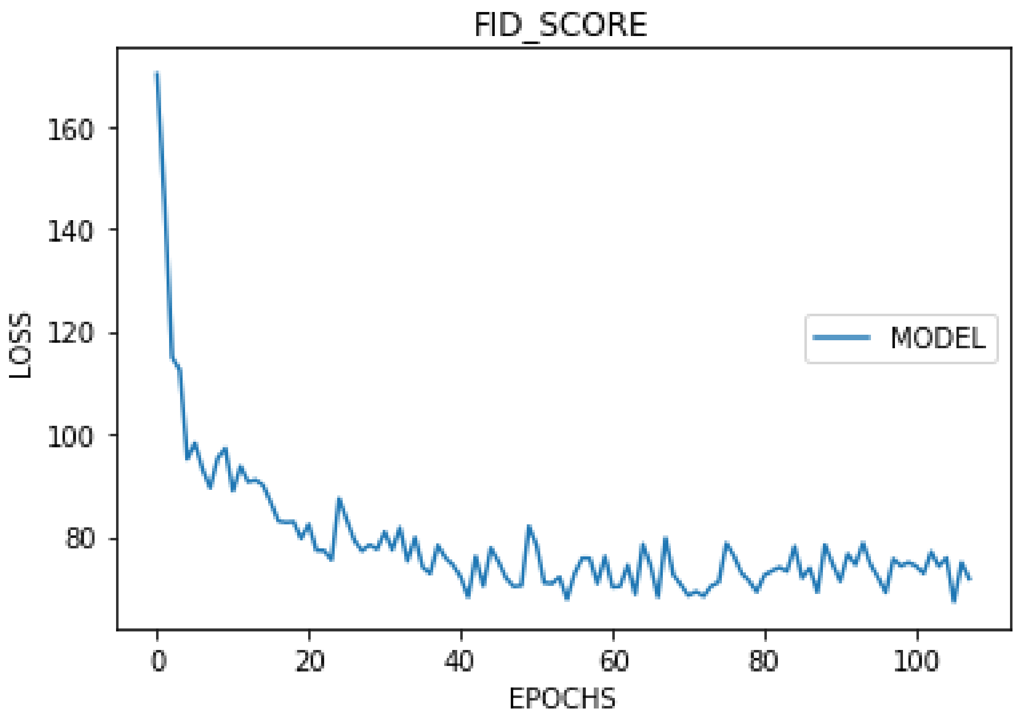

DIscriminator_REAL_LOSS
LOSS
ITERATIONS
Discriminator
0.0
0.5
1.0
1.5
2.0
2.5
0
20000
40000
60000
80000
100000


DIscriminator_FAKE_LOSS
LOSS
ITERATIONS
Discriminator
0
1
2
3
4
5
0
20000
40000
60000
80000
100000


DIscriminator_TOTAL_LOSS
LOSS
ITERATIONS
Discriminator
0
1
2
3
4
5
0
20000
40000
60000
80000
100000


GENERATOR_LOSS
LOSS
ITERATIONS
GENERATOR
0
2
4
6
8
10
12
14
16
0
20000
40000
60000
80000
100000

FID plot:



Sample images generated:

EPOCH 50:

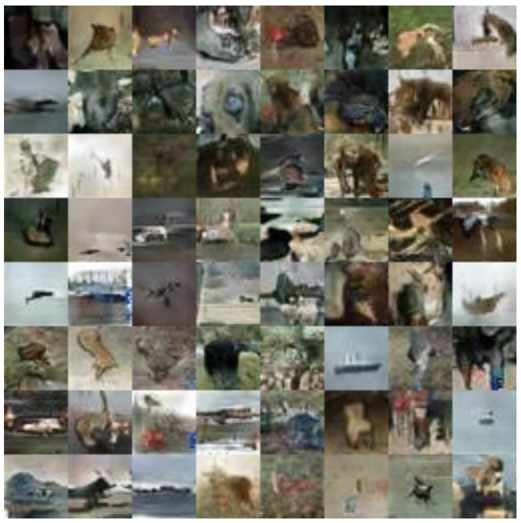


Fig 1.1

EPOCH 100:

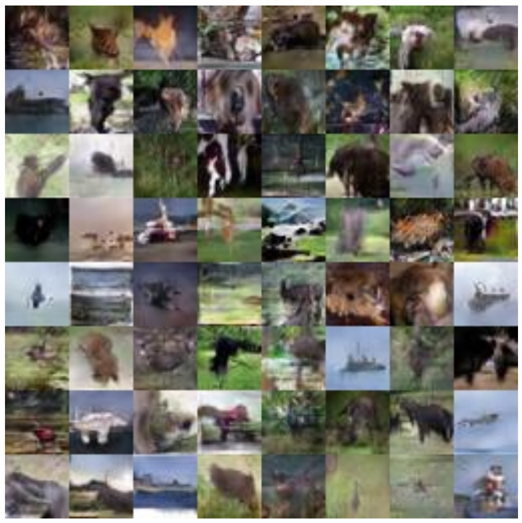


Fig 1.2

EPOCH 130

EPOCH 180:

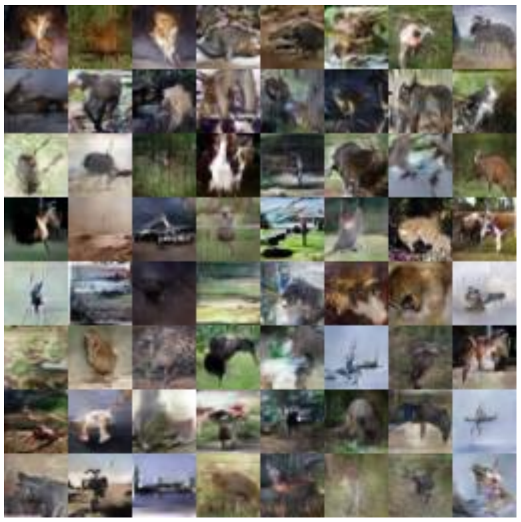


Fig 1.3

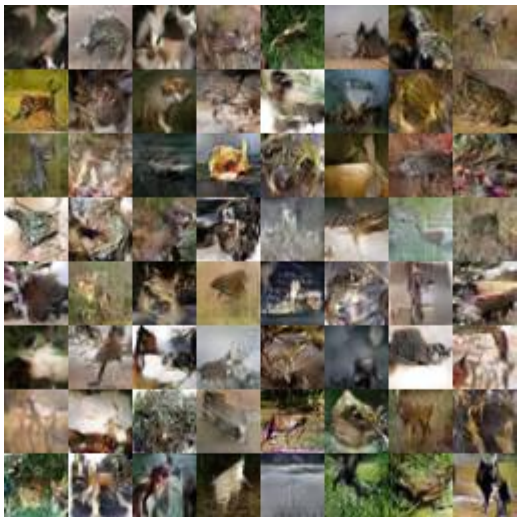


Fig 1.4

Conditional Spectral Normalized Self-Attention GAN(cSNSAGAN):

- **Data**
    - Rescale the Cifar-10 images to be between -1 and 1.
- **Generator**
    - Use the Spectral Normalized inverse of convolution, called transposedSNconvolution.
    - LeakyReLU activation and Batch Normalization.
    - The input to the generator are the normal distribution and. They are combined in joint hidden representation.
        - Concatenate( ).
    - The last activation is tanh.
- **Discriminator**
    - Use the Spectral Normalized Convolutional neural network.
    - LeakyReLU activation.
    - The input to the discriminator are and. They are combined in joint hidden representation.

    - Concatenate(   ).
-
- **Loss**
    - Hinge Loss
- **Optimizer**
    - Adam
- batch size = 64
- epochs = 180

SELF ATTENTION:

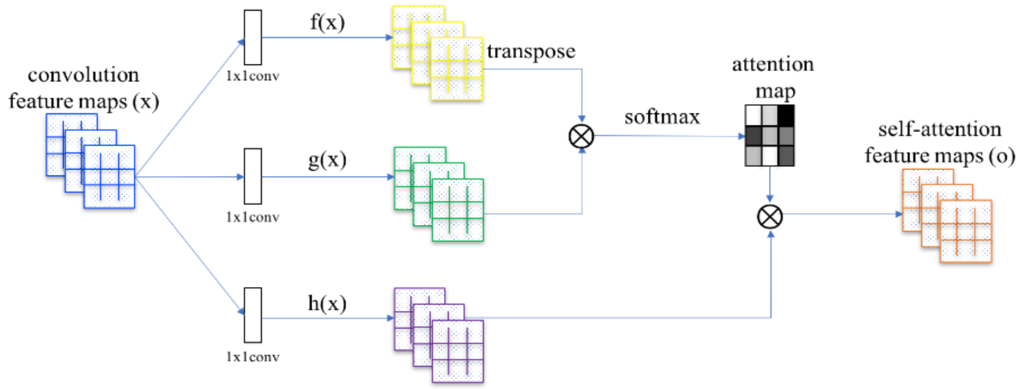


Figure 2: The proposed self-attention mechanism. The ⊗ denotes matrix multiplication. The softmax operation is performed on each row.

Source: Google

IMPLEMENTATION DETAILS:

- Taken the input features to a convolutional layer L we transform L to three different representations.
- We convolve L using 1x1 convolutions to get three feature spaces (f, g, and h)
- We use f and g to calculate the attention. To do this, we linearly combine f and g using a matrix multiplication
- The result is fed into a softmax layer
- The feature vectors f and g have different dimensions to h. As a matter of fact, f and g use eight times fewer convolutional filters than h.
- The resulting tensor is linearly-combined with h and, finally, scaled by gamma.
- Note that gamma starts as 0

Gamma Layer Functioning:

- At the beginning of training, gamma cancels out the attention layers. As a consequence, the network only relies on local representations from the regular convolutional layers.
- As gamma receives gradient descent updates, the network gradually allows the passage of signals from non-local fields

Spectral Normalization:

Functionality:

SN constrains the Lipschitz constant of the convolutional filters

Use:

SN as the best way to stabilize the training of the **Discriminator** network

Need for Spectral Normalization:

- There is a fundamental problem with the normalized Discriminator prior work and models have shown that regularized discriminator makes GAN Training Slower

- To solve the problem of slow learning and imbalanced update steps, there is a simple yet effective approach that is the two-timescale update rule (TTUR) in the GAN training. It consists of providing different learning rates for optimizing **G** and **D**.

- For **Generator**, spectral normalization prevents the parameters getting very big, and avoids unwanted gradients

It defines that the spectral norm used to regularize each Conv layer is the largest singular value of W.

Applying singular value decomposition at each step might be computational expansive. Instead, we use the power iteration method to estimate the spectral normal of each layer

ALGORITHM:

```
Algorithm 1  SGD with spectral normalization

• Initialize $\tilde{u}_l \in \mathcal{R}^{d_l}$ for $l = 1, \dots, L$ with a random vector (sampled from isotropic distribution).
• For each update and each layer $l$:
    1. Apply power iteration method to a unnormalized weight $W^l$:
           $\tilde{v}_l \leftarrow (W^l)^{\mathrm{T}} \tilde{u}_l / \|(W^l)^{\mathrm{T}} \tilde{u}_l\|_2$        (20)
           $\tilde{u}_l \leftarrow W^l \tilde{v}_l / \|W^l \tilde{v}_l\|_2$        (21)
    2. Calculate $\bar{W}_{\mathrm{SN}}$ with the spectral norm:
           $\bar{W}^l_{\mathrm{SN}}(W^l) = W^l / \sigma(W^l), \text{ where } \sigma(W^l) = \tilde{u}_l^{\mathrm{T}} W^l \tilde{v}_l$        (22)
    3. Update $W^l$ with SGD on mini-batch dataset $\mathcal{D}_M$ with a learning rate $\alpha$:
           $W^l \leftarrow W^l - \alpha \nabla_{W^l} \ell(\bar{W}^l_{\mathrm{SN}}(W^l), \mathcal{D}_M)$        (23)
```

Implemented Generator Model:

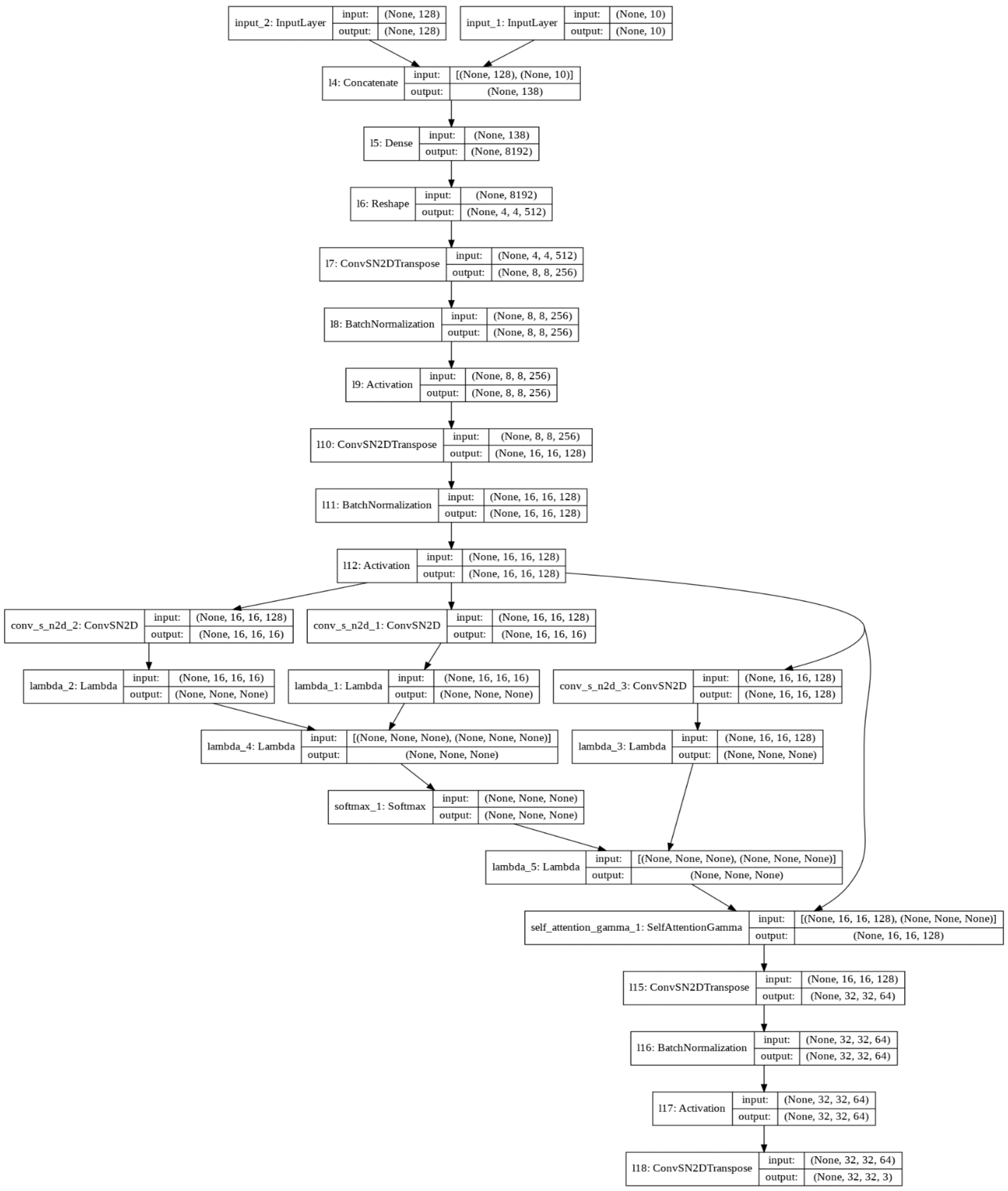

input_2: InputLayer | input: (None, 128) | output: (None, 128)
input_1: InputLayer | input: (None, 10) | output: (None, 10)
l4: Concatenate | input: [(None, 128), (None, 10)] | output: (None, 138)
l5: Dense | input: (None, 138) | output: (None, 8192)
l6: Reshape | input: (None, 8192) | output: (None, 4, 4, 512)
l7: ConvSN2DTranspose | input: (None, 4, 4, 512) | output: (None, 8, 8, 256)
l8: BatchNormalization | input: (None, 8, 8, 256) | output: (None, 8, 8, 256)
l9: Activation | input: (None, 8, 8, 256) | output: (None, 8, 8, 256)
l10: ConvSN2DTranspose | input: (None, 8, 8, 256) | output: (None, 16, 16, 128)
l11: BatchNormalization | input: (None, 16, 16, 128) | output: (None, 16, 16, 128)
l12: Activation | input: (None, 16, 16, 128) | output: (None, 16, 16, 128)
conv_s_n2d_2: ConvSN2D | input: (None, 16, 16, 128) | output: (None, 16, 16, 16)
conv_s_n2d_1: ConvSN2D | input: (None, 16, 16, 128) | output: (None, 16, 16, 16)
lambda_2: Lambda | input: (None, 16, 16, 16) | output: (None, None, None)
lambda_1: Lambda | input: (None, 16, 16, 16) | output: (None, None, None)
conv_s_n2d_3: ConvSN2D | input: (None, 16, 16, 128) | output: (None, 16, 16, 128)
lambda_4: Lambda | input: [(None, None, None), (None, None, None)] | output: (None, None, None)
lambda_3: Lambda | input: (None, 16, 16, 128) | output: (None, None, None)
softmax_1: Softmax | input: (None, None, None) | output: (None, None, None)
lambda_5: Lambda | input: [(None, None, None), (None, None, None)] | output: (None, None, None)
self_attention_gamma_1: SelfAttentionGamma | input: [(None, 16, 16, 128), (None, None, None)] | output: (None, 16, 16, 128)
l15: ConvSN2DTranspose | input: (None, 16, 16, 128) | output: (None, 32, 32, 64)
l16: BatchNormalization | input: (None, 32, 32, 64) | output: (None, 32, 32, 64)
l17: Activation | input: (None, 32, 32, 64) | output: (None, 32, 32, 64)
l18: ConvSN2DTranspose | input: (None, 32, 32, 64) | output: (None, 32, 32, 3)

Fig: cSNSAGAN Generator model

Implemented Discriminator:

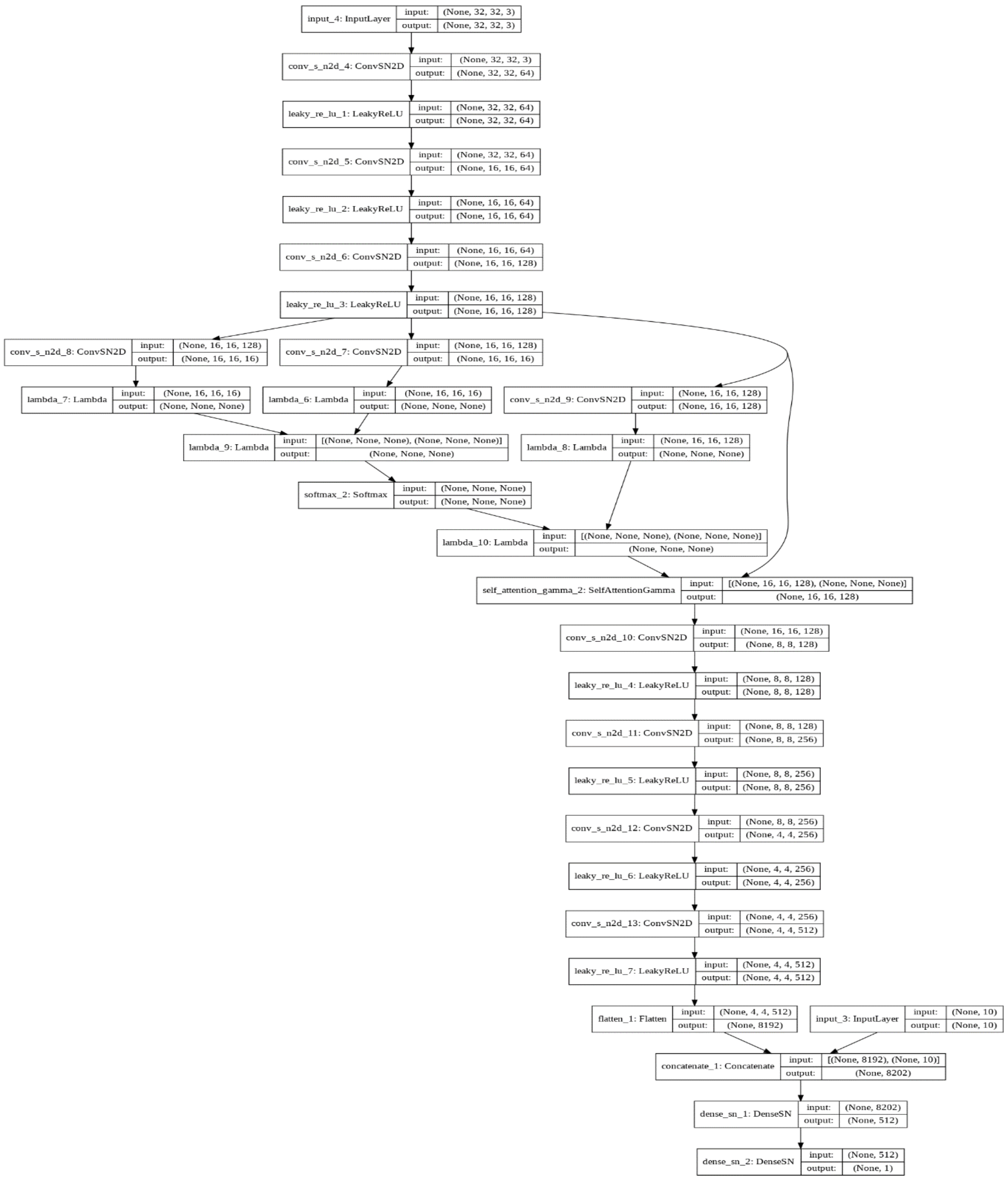
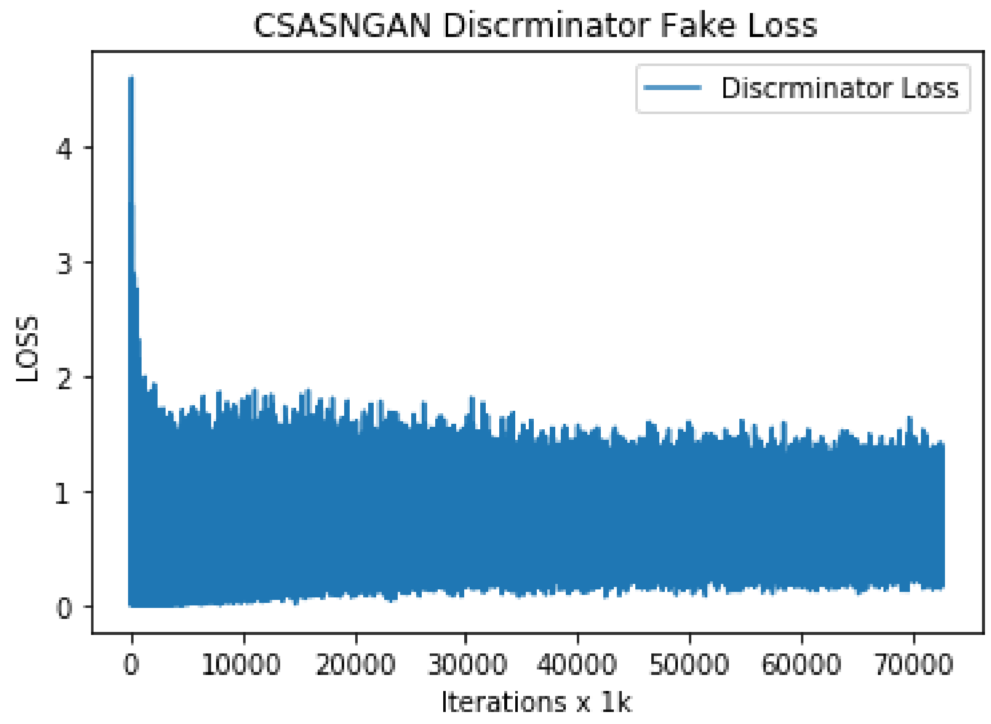
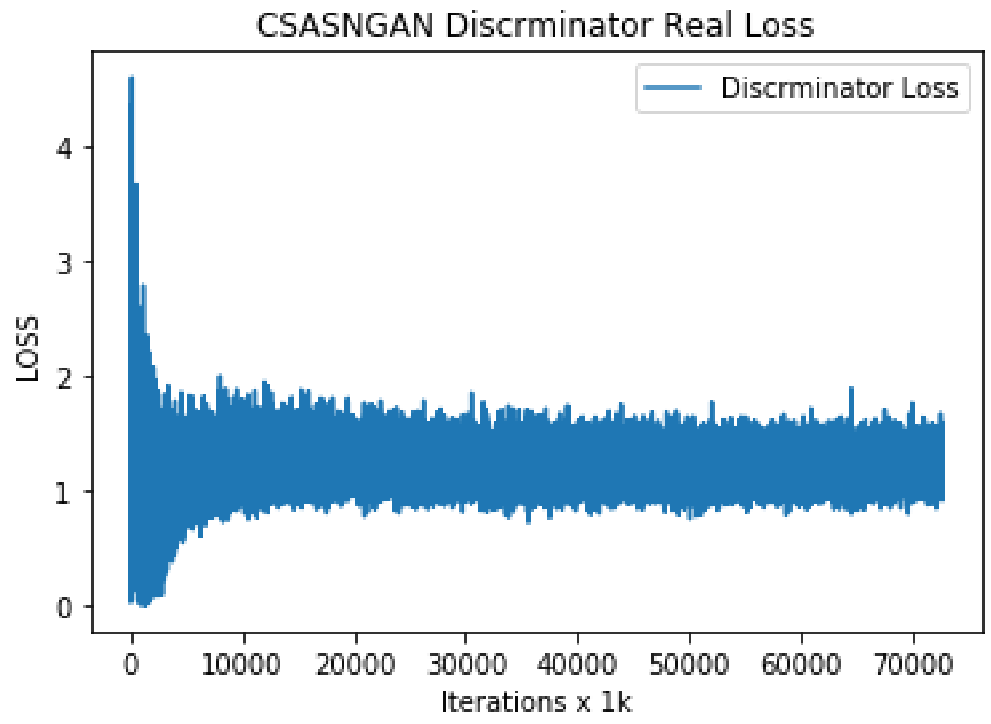
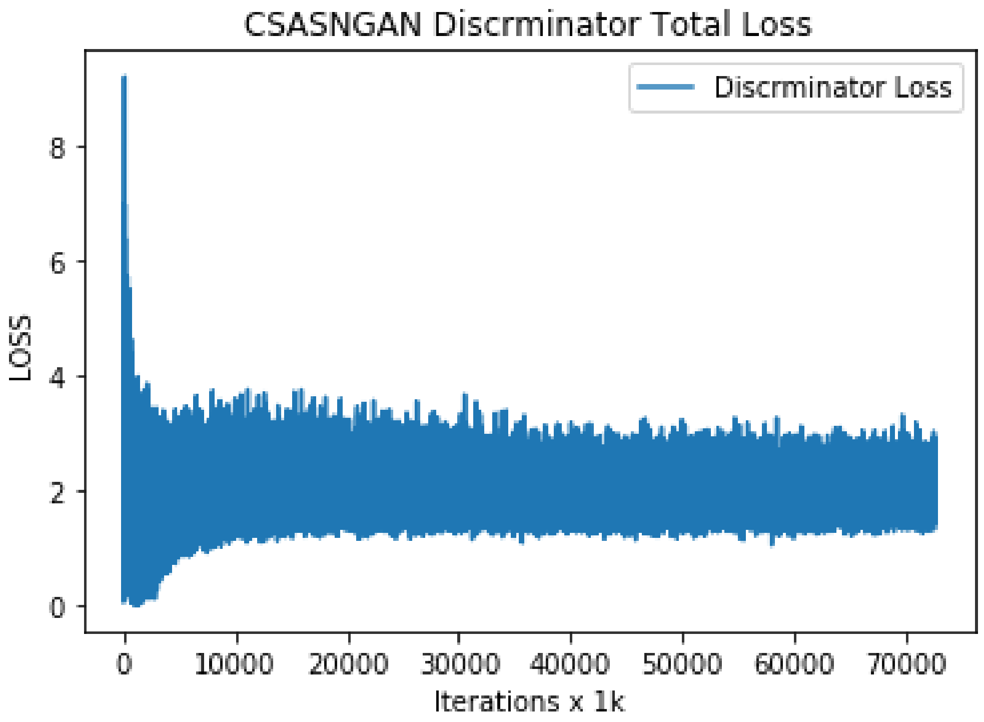

input_4: InputLayer | input: (None, 32, 32, 3) | output: (None, 32, 32, 3)
conv_s_n2d_4: ConvSN2D | input: (None, 32, 32, 3) | output: (None, 32, 32, 64)
leaky_re_lu_1: LeakyReLU | input: (None, 32, 32, 64) | output: (None, 32, 32, 64)
conv_s_n2d_5: ConvSN2D | input: (None, 32, 32, 64) | output: (None, 16, 16, 64)
leaky_re_lu_2: LeakyReLU | input: (None, 16, 16, 64) | output: (None, 16, 16, 64)
conv_s_n2d_6: ConvSN2D | input: (None, 16, 16, 64) | output: (None, 16, 16, 128)
leaky_re_lu_3: LeakyReLU | input: (None, 16, 16, 128) | output: (None, 16, 16, 128)
conv_s_n2d_8: ConvSN2D | input: (None, 16, 16, 128) | output: (None, 16, 16, 16)
conv_s_n2d_7: ConvSN2D | input: (None, 16, 16, 128) | output: (None, 16, 16, 16)
lambda_7: Lambda | input: (None, 16, 16, 16) | output: (None, None, None)
lambda_6: Lambda | input: (None, 16, 16, 16) | output: (None, None, None)
conv_s_n2d_9: ConvSN2D | input: (None, 16, 16, 128) | output: (None, 16, 16, 128)
lambda_9: Lambda | input: [(None, None, None), (None, None, None)] | output: (None, None, None)
lambda_8: Lambda | input: (None, 16, 16, 128) | output: (None, None, None)
softmax_2: Softmax | input: (None, None, None) | output: (None, None, None)
lambda_10: Lambda | input: [(None, None, None), (None, None, None)] | output: (None, None, None)
self_attention_gamma_2: SelfAttentionGamma | input: [(None, 16, 16, 128), (None, None, None)] | output: (None, 16, 16, 128)
conv_s_n2d_10: ConvSN2D | input: (None, 16, 16, 128) | output: (None, 8, 8, 128)
leaky_re_lu_4: LeakyReLU | input: (None, 8, 8, 128) | output: (None, 8, 8, 128)
conv_s_n2d_11: ConvSN2D | input: (None, 8, 8, 128) | output: (None, 8, 8, 256)
leaky_re_lu_5: LeakyReLU | input: (None, 8, 8, 256) | output: (None, 8, 8, 256)
conv_s_n2d_12: ConvSN2D | input: (None, 8, 8, 256) | output: (None, 4, 4, 256)
leaky_re_lu_6: LeakyReLU | input: (None, 4, 4, 256) | output: (None, 4, 4, 256)
conv_s_n2d_13: ConvSN2D | input: (None, 4, 4, 256) | output: (None, 4, 4, 512)
leaky_re_lu_7: LeakyReLU | input: (None, 4, 4, 512) | output: (None, 4, 4, 512)
flatten_1: Flatten | input: (None, 4, 4, 512) | output: (None, 8192)
input_3: InputLayer | input: (None, 10) | output: (None, 10)
concatenate_1: Concatenate | input: [(None, 8192), (None, 10)] | output: (None, 8202)
dense_sn_1: DenseSN | input: (None, 8202) | output: (None, 512)
dense_sn_2: DenseSN | input: (None, 512) | output: (None, 1)

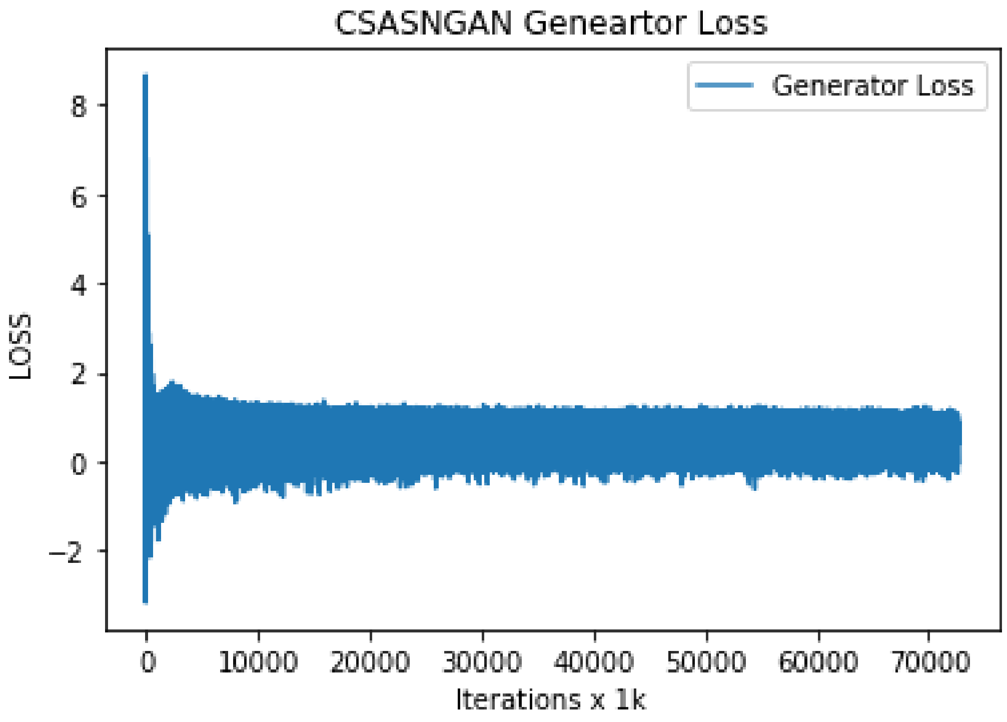
Fig: cSNSAGAN Generator model

LOSS PLOTS:

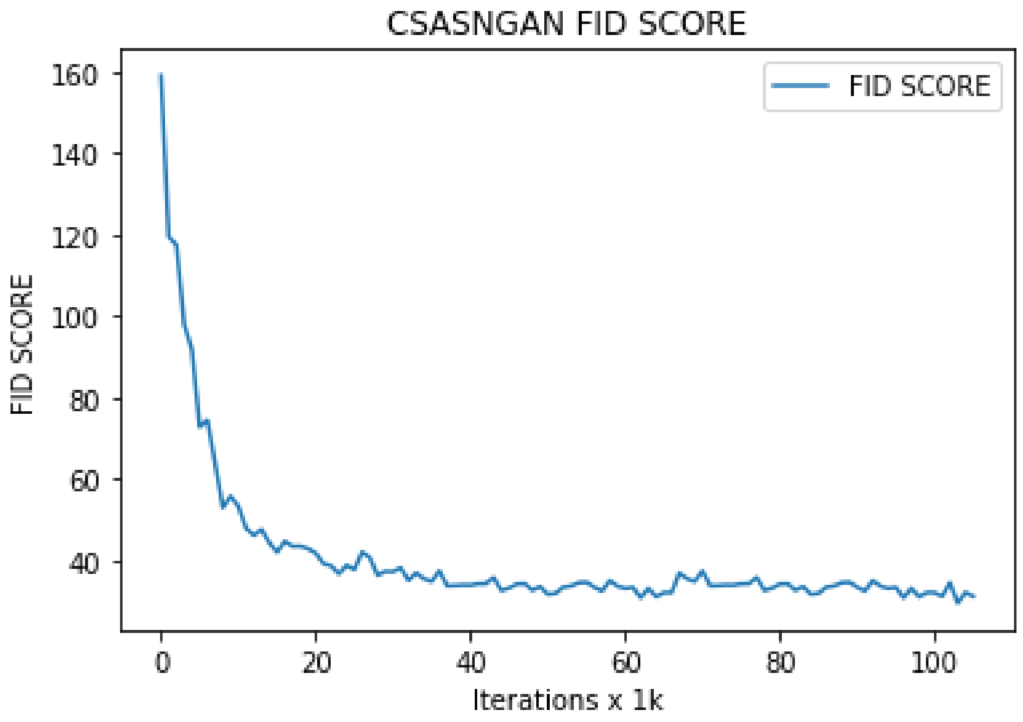

CSASNGAN Discrminator Fake Loss
Discrminator Loss
LOSS
0
1
2
3
4
0
10000
20000
30000
40000
50000
60000
70000
Iterations x 1k


CSASNGAN Discrminator Real Loss
Discrminator Loss
LOSS
0
1
2
3
4
0
10000
20000
30000
40000
50000
60000
70000
Iterations x 1k


CSASNGAN Discrminator Total Loss
Discrminator Loss
LOSS
Iterations x 1k
0
2
4
6
8
0
10000
20000
30000
40000
50000
60000
70000


CSASNGAN Geneartor Loss
Generator Loss
LOSS
Iterations x 1k
−2
0
2
4
6
8
0
10000
20000
30000
40000
50000
60000
70000

FID PLOT:



Sample images generated:

EPOCH 50

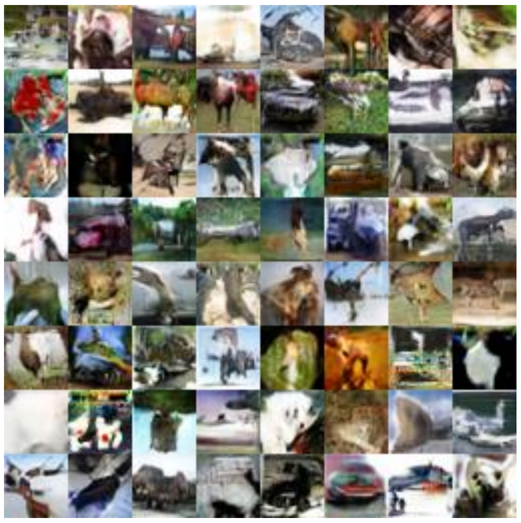


Fig 2.1

EPOCH 73

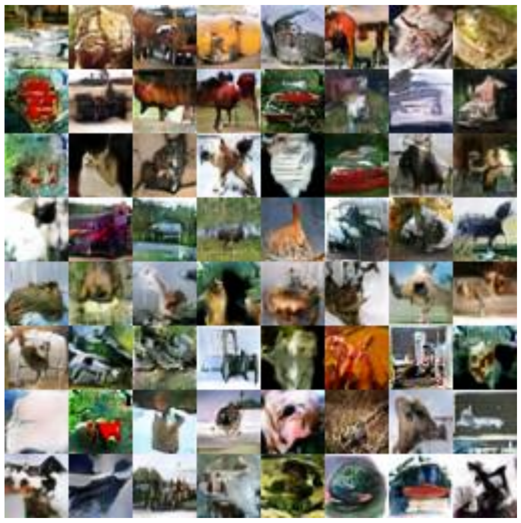


Fig 2.2

EPOCH 92

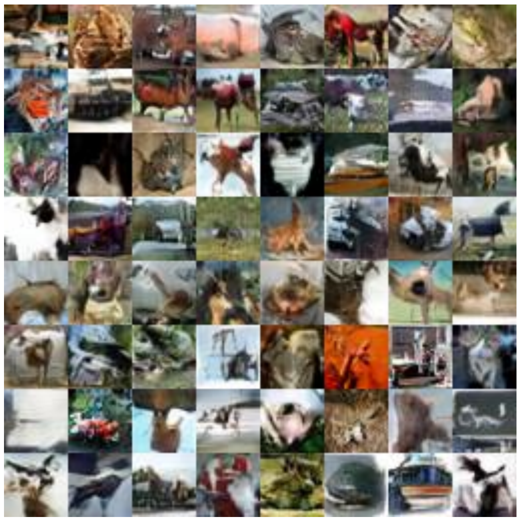


Fig 2.3

EPOCH 120

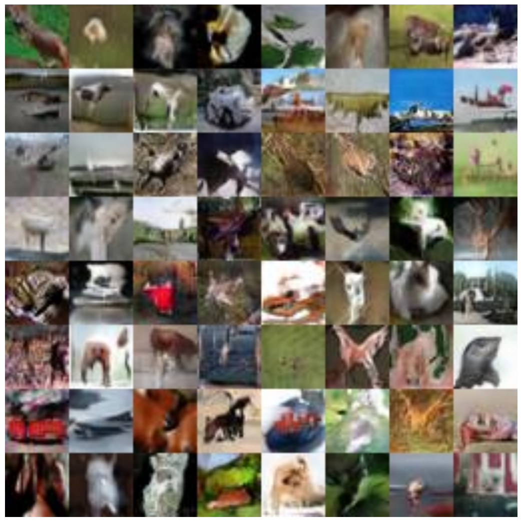


Fig 2.4

EPOCH 150

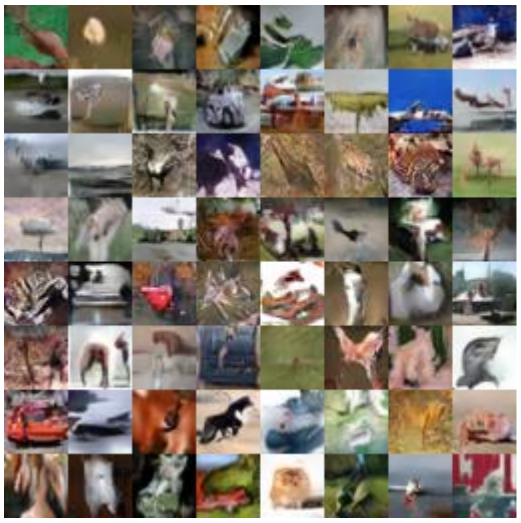


Fig 5

Results:

Table 1:

MODEL	**EPOCHS**	**FID**
DCGAN	**50**	**78.13467**
CSNSAGAN	**50**	**48.984321**

Table 2:

MODEL	**EPOCHS**	**FID**
DCGAN	**100**	**63. 6250589**
CSNSAGAN	**100**	**35.87096555**

Table 3:

MODEL	**EPOCHS**	**FID**
DCGAN	**120**	**55.81421681247131**
CSNSAGAN	**120**	**34.236131287577**

Table4:

MODEL	**EPOCHS**	**FID**
DCGAN	**150**	**43. 6250589**
CSNSAGAN	**150**	**29.24059076483727**

BEST FID SEEN FOR DCGAN: 43. 6250589

BEST FID SEEN FOR cSNSAGAN: 29.24059076483727

EPOCH 50 DCGAN vs cSNSAGAN IMAGES:

DCGAN(EPOCH:50)

DCGAN(EPOCH:50)

EPOCH 100 DCGAN vs cSNSAGAN IMAGES:

DCGAN(EPOCH:100)

cSNSAGAN(EPOCH:100)