Exploring Techniques for Generative Model Authorship

Eesh Joshi
eesh.joshi@ufl.edu

Himanshu Gupta
himanshu.gupta@ufl.edu

Kunwardeep Singh
kunwardeep.singh@ufl.edu

January 7, 2020

1 Summary

The purpose of this project is to explore the extent upto which images generated using generative models can be identified. Two major techniques are used. One of them is to model this problem as a classification task and use a Convolutional Neural Network to classify the images generated from respective GANs. The other technique is to train an inverse model which tries to learn the latent space of the GAN. The image is then reconstructed and compared with the original input image. If the L2 norm of the distance between two images is less than a given threshold, then it is highly likely that the image is generated with the same Generator.

2 Datasets

The datasets used are MNIST and CIFAR-10.

3 Experimental Results

3.1 Experiments with MNIST dataset

Simple GANs on MNIST data were trained. The architecture of each GAN was same. Training method of each GAN was also same (i.e. training epochs, optimizers, etc). The training data of each GAN was different; it was some combinations of MNIST digits.

3.1.1 GAN Architecture

The discriminator has the following architecture (refer Fig: 1):

- 2D Convolution Layer 64 filter (3,3)
- LeakyReLu with alpha = 0.2
- Dropout with drop probability 0.4
- 2D Convolution Layer 64 filter (3,3)
- LeakyReLu with alpha = 0.2
- Dropout drop probability 0.4
- Flatten Layer
- Output Layer with 1 Neuron

Layer (type)	Output Shape	Param #
conv2d_1 (Conv2D)	(None, 14, 14, 64)	640
leaky_re_lu_1 (LeakyReLU)	(None, 14, 14, 64)	0
dropout_1 (Dropout)	(None, 14, 14, 64)	0
conv2d_2 (Conv2D)	(None, 7, 7, 64)	36928
leaky_re_lu_2 (LeakyReLU)	(None, 7, 7, 64)	0
dropout_2 (Dropout)	(None, 7, 7, 64)	0
flatten_1 (Flatten)	(None, 3136)	0
dense_1 (Dense)	(None, 1)	3137

Total params: 81,410
Trainable params: 40,705
Non-trainable params: 40,705

Figure 1: GAN Discriminator

Layer (type)	Output Shape	Param #
dense_2 (Dense)	(None, 6272)	633472
leaky_re_lu_3 (LeakyReLU)	(None, 6272)	0
reshape_1 (Reshape)	(None, 7, 7, 128)	0
conv2d_transpose_1 (Conv2DTr	(None, 14, 14, 128)	262272
leaky_re_lu_4 (LeakyReLU)	(None, 14, 14, 128)	0
conv2d_transpose_2 (Conv2DTr	(None, 28, 28, 128)	262272
leaky_re_lu_5 (LeakyReLU)	(None, 28, 28, 128)	0
conv2d_3 (Conv2D)	(None, 28, 28, 1)	6273

Total params: 1,164,289
Trainable params: 1,164,289
Non-trainable params: 0

Figure 2: GAN Generator

The Generator has the following architecture (refer Fig: 2):

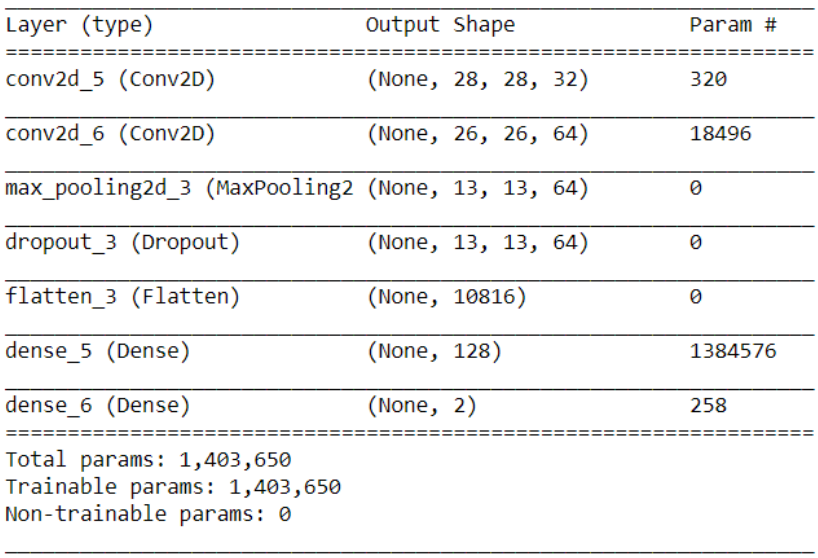
- Fully Connected Layer with 6272 Neuron (128*7*7)
- Leaky ReLu with alpha 0.2
- Reshape Layer shape = (7,7,128)
- 2D Convolutional Layer(Transpose) 128 filters shape = (4,4)
- Leaky ReLu with alpha = 0.2

- 2D Convolutional Layer(Transpose) 128 filters shape = (4,4)
- Leaky ReLu with alpha = 0.2
- 2D Convolution Layer 1 filter, shape = (7,7)

For training the GAN, we used Adam optimizer with learning rate = 0.0002. We are starting with 100-dimensional random noise for generating images. We used 200 epochs and batch size of 256.

3.1.2 Architecture of the Model for Authorship Determination

Our model is a simple ConvNet with the following architecture (refer Fig: 3):



Layer (type)	Output Shape	Param #
conv2d_5 (Conv2D)	(None, 28, 28, 32)	320
conv2d_6 (Conv2D)	(None, 26, 26, 64)	18496
max_pooling2d_3 (MaxPooling2	(None, 13, 13, 64)	0
dropout_3 (Dropout)	(None, 13, 13, 64)	0
flatten_3 (Flatten)	(None, 10816)	0
dense_5 (Dense)	(None, 128)	1384576
dense_6 (Dense)	(None, 2)	258

Total params: 1,403,650
Trainable params: 1,403,650
Non-trainable params: 0

Figure 3: Model for GAN Authorship

- 2D Convolution Layer with 32 filters, shape (3,3)
- 2D Convolution Layer with 64 filters, shape (3,3)
- Max Pool Layer Shape(2,2)
- Drop Out Layer with 0.25 dropout probability
- Flatten Layer
- Dense Layer 128 Neurons
- Softmax Layer

We trained 13 different GAN's. We named these GAN's GAN_1, GAN_2, etc. These GAN's have same architecture but the training data was different for each. The details of training data for each GAN is given below.

3.1.3 Training Data for GAN

The GAN's only differed in the training data. We trained 13 GAN's altogether.

- GAN_1 : Trained on MNIST digits [1,2,7,5]
- GAN_2 : Trained on MNIST digits [1,4,5,8,9]
- GAN_3: Trained on MNIST digits [0,3,8]
- GAN_4: Trained on MNIST digits [1,2,3,5,7,8,9]
- GAN_5: Trained on MNIST digits [0,1,2,3,4,5,6]
- GAN_6: Trained on MNIST digits [0,1,2,3]
- GAN_7: Trained on MNIST digits [0,1,2,3,8]
- GAN_8 and GAN_9: Trained on disjoint partition of MNIST digits [3,6,8]
- GAN_10 and GAN_11: Trained on disjoint partition of MNIST digits[1]
- GAN_12 and GAN_13: Trained on same training set MNIST digit [1]

These GAN's were then divided into groups (details given below). For each GAN in each group, we generated 20000 images. A dataset was thus formed per group. This dataset was divided into training set, validation set and testing set. The authorship model was trained on training set. We measured the test accuracy and calculated the confusion matrix for each group.

3.1.4 Groups of GANs

- Group_1 : GAN_1, GAN_2, GAN_3 and Real Samples
- Group_2: GAN_4, GAN_5 and Real Samples
- Group_3 : GAN_6 , GAN_7 and Real Samples
- Group_4: GAN_8, GAN_9 and Real Samples
- Group_5: GAN_10 and GAN_11
- Group_6: GAN_12 and GAN_13

3.1.5 Results

Group_1 : Test Accuracy : 0.974975

	GAN_1	**GAN_2**	**GAN_3**	**Real**
GAN_1	6312	112	9	281
GAN_2	73	6449	1	180
GAN_3	16	12	6523	175
Real	52	63	27	19175

Group_2: Test Accuracy : 0.9651

	GAN_4	GAN_5	Real
GAN_4	5684	157	200
GAN_5	99	5693	140
Real	190	259	17578

Group_3: Test Accuracy : 0.9573

	GAN_6	GAN_7	Real
GAN_6	5542	66	214
GAN_7	92	5573	190
Real	221	96	8611

Group_4: Test Accuracy: 0.9877

	GAN_8	GAN_9	Real
GAN_8	4407	27	15
GAN_9	58	4373	15
Real	9	34	3962

Group_5: Test Accuracy: 0.9906

	GAN_10	GAN_11
GAN_10	2416	26
GAN_11	21	2537

Group_6: Test Accuracy: 1.0

	GAN_12	GAN_13
GAN_12	2495	0
GAN_13	0	2505

From the results we see that even a simple Machine Learning model such as ours, was able to discriminate between images from different GAN's with high accuracy. Even in the case when training data was same (Group_6) we are getting very good accuracy. The result of Group_6 are especially surprising, as we were expecting out model to perform poorly in this case. We speculate that GAN's must be leaving some kind of "fingerprint" in the image which is learnt by our ML model. Further, this fingerprint seems to be very sensitive to training (from Group_6 results). This leads immediately to the following questions:

- Can we isolate the fingerprint?
- What are the characteristics of this fingerprint?
- Is the fingerprint unique for a GAN?

3.1.6 Other Experiments

- Two GANs having different architectures were used. GAN1 was DCGAN and GAN2 was SGAN. A logistic regression classifier to classify the images from the two GANs. High classification accuracy was achieved(94%). The reason being the two images generated from the two different GANs could be easily distinguished by a human as well. The images generated by GAN2 had a higher quality than GAN1. Results are shown in Fig 4.

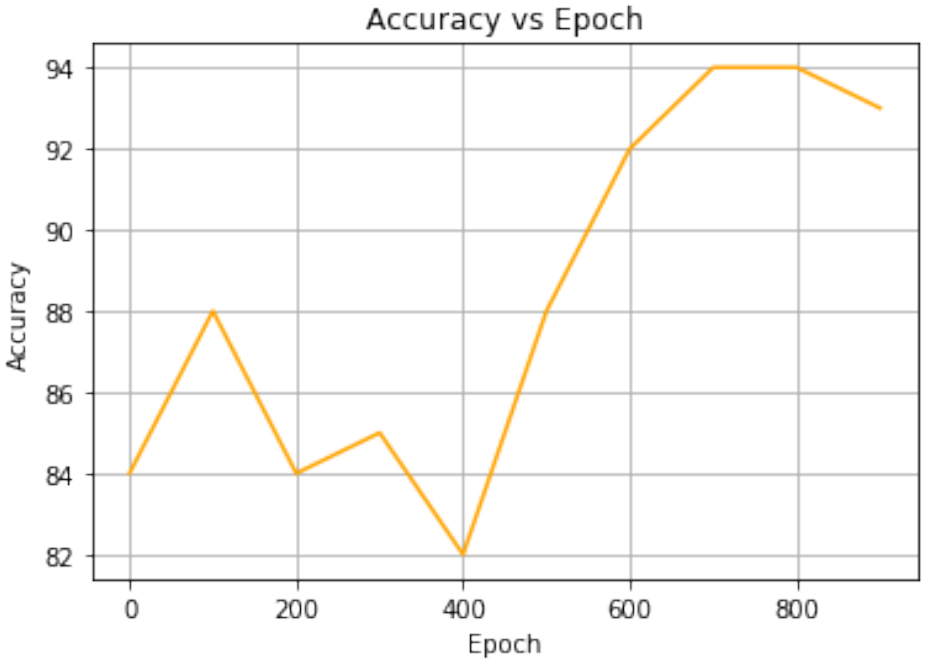


Figure 4: Experiment 1

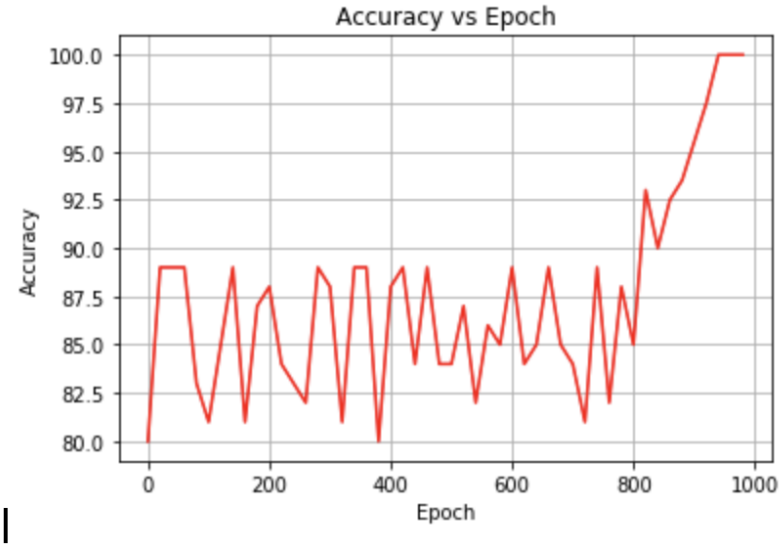


Figure 5: Experiment 2

- Two GANs(DCGAN) having the same architecture but different activation functions(GAN1 had sigmoid activation function, GAN2 had tanh activation function) were trained. It was found that the GAN2 started producing garbage images after epoch 820. A logistic regression classifier to classify the images from the two GANs. High accuracy(90+) was achieved. Results are shown in Fig 5
- Two GANs(DCGAN) having the same architecture were trained for 1000 epochs. A logistic regression classifier to classify the images from the two GANs. The accuracy first started increasing and then started decreasing and then again started increasing. It was found that the highest accuracy was achieved at 400 epoch(90%). Results are shown in Fig 6

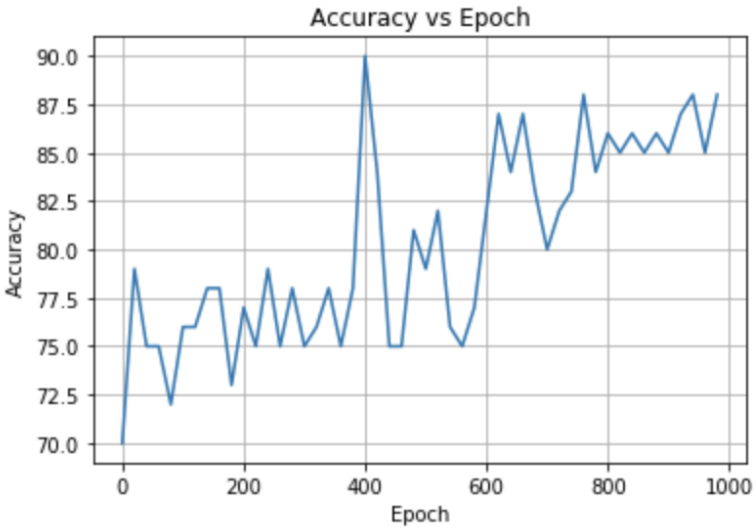


Figure 6: Experiment 3

3.2 Experiments with CIFAR-10 dataset

This dataset consists of colored images of 10 different classes. Some of the sample images of CIFAR dataset are shown in Fig. 7 Initially, CGAN and DCGAN architectures were used to generate synthetic images. However, due to mode collapse issue, these were replaced with the following architectures:

3.2.1 GAN 1

The following architecture was used for generator:

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_3 (Dense)              (None, 4096)              413696
_________________________________________________________________
leaky_re_lu_9 (LeakyReLU)    (None, 4096)              0
_________________________________________________________________
reshape_1 (Reshape)          (None, 4, 4, 256)         0
_________________________________________________________________
conv2d_transpose_1 (Conv2DTr (None, 8, 8, 128)         524416
_________________________________________________________________
leaky_re_lu_10 (LeakyReLU)   (None, 8, 8, 128)         0
_________________________________________________________________
conv2d_transpose_2 (Conv2DTr (None, 16, 16, 128)       262272
_________________________________________________________________
leaky_re_lu_11 (LeakyReLU)   (None, 16, 16, 128)       0
_________________________________________________________________
conv2d_transpose_3 (Conv2DTr (None, 32, 32, 128)       262272
_________________________________________________________________
leaky_re_lu_12 (LeakyReLU)   (None, 32, 32, 128)       0
_________________________________________________________________
conv2d_9 (Conv2D)            (None, 32, 32, 3)         3459
=================================================================
```

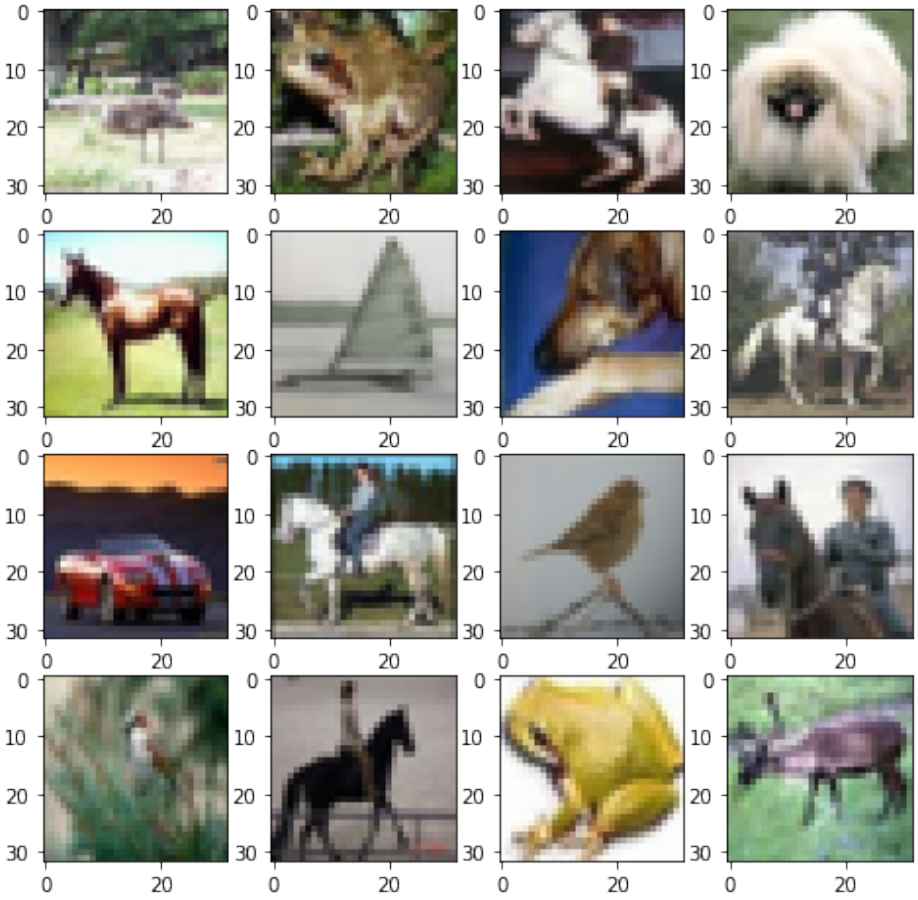


Figure 7: CIFAR 10 Sample images

```
Total params: 1,466,115
Trainable params: 1,466,115
Non-trainable params: 0
_________________________________________________________________
```

The following architecture was used for discriminator:

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_5 (Conv2D)            (None, 32, 32, 64)        1792
_________________________________________________________________
leaky_re_lu_5 (LeakyReLU)    (None, 32, 32, 64)        0
_________________________________________________________________
conv2d_6 (Conv2D)            (None, 16, 16, 128)       73856
_________________________________________________________________
leaky_re_lu_6 (LeakyReLU)    (None, 16, 16, 128)       0
_________________________________________________________________
conv2d_7 (Conv2D)            (None, 8, 8, 128)         147584
_________________________________________________________________
leaky_re_lu_7 (LeakyReLU)    (None, 8, 8, 128)         0
_________________________________________________________________
conv2d_8 (Conv2D)            (None, 4, 4, 256)         295168
```

```
_________________________________________________________________
leaky_re_lu_8 (LeakyReLU)    (None, 4, 4, 256)         0
_________________________________________________________________
flatten_2 (Flatten)          (None, 4096)              0
_________________________________________________________________
dropout_2 (Dropout)          (None, 4096)              0
_________________________________________________________________
dense_2 (Dense)              (None, 1)                 4097
=================================================================
Total params: 522,497
Trainable params: 522,497
Non-trainable params: 0
_________________________________________________________________
```

After training for 2000 epochs the images generated look as shown in Fig. 8.

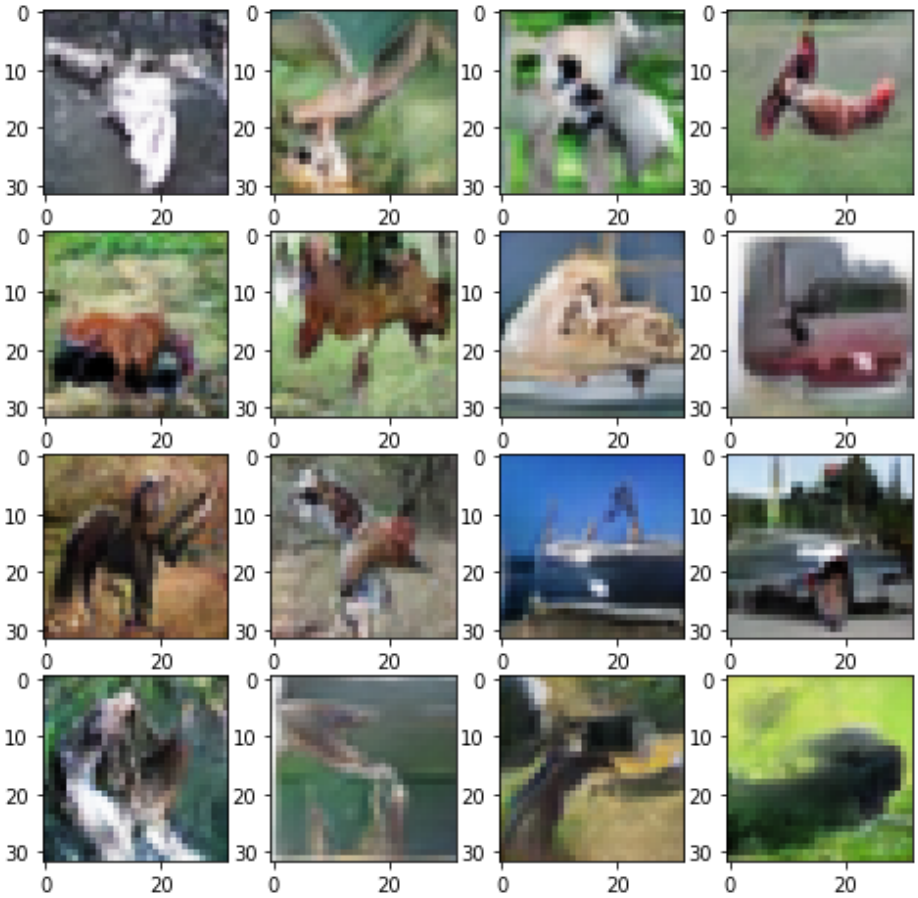


Figure 8: GAN 1 Generated images

3.2.2 GAN 2

Generator architecture:

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
```

```
dense_1 (Dense)              (None, 8192)              827392
_________________________________________________________________
reshape_1 (Reshape)          (None, 8, 8, 128)         0
_________________________________________________________________
up_sampling2d_1 (UpSampling2 (None, 16, 16, 128)       0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 16, 16, 128)       147584
_________________________________________________________________
batch_normalization_1 (Batch (None, 16, 16, 128)       512
_________________________________________________________________
activation_1 (Activation)    (None, 16, 16, 128)       0
_________________________________________________________________
up_sampling2d_2 (UpSampling2 (None, 32, 32, 128)       0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 32, 32, 64)        73792
_________________________________________________________________
batch_normalization_2 (Batch (None, 32, 32, 64)        256
_________________________________________________________________
activation_2 (Activation)    (None, 32, 32, 64)        0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 32, 32, 3)         1731
_________________________________________________________________
activation_3 (Activation)    (None, 32, 32, 3)         0
=================================================================
Total params: 1,051,267
Trainable params: 1,050,883
Non-trainable params: 384
_________________________________________________________________
```

Discriminator architecture:

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_12 (Conv2D)           (None, 16, 16, 32)        896
_________________________________________________________________
leaky_re_lu_9 (LeakyReLU)    (None, 16, 16, 32)        0
_________________________________________________________________
dropout_9 (Dropout)          (None, 16, 16, 32)        0
_________________________________________________________________
conv2d_13 (Conv2D)           (None, 8, 8, 64)          18496
_________________________________________________________________
zero_padding2d_3 (ZeroPaddin (None, 9, 9, 64)          0
_________________________________________________________________
batch_normalization_9 (Batch (None, 9, 9, 64)          256
_________________________________________________________________
leaky_re_lu_10 (LeakyReLU)   (None, 9, 9, 64)          0
_________________________________________________________________
dropout_10 (Dropout)         (None, 9, 9, 64)          0
_________________________________________________________________
conv2d_14 (Conv2D)           (None, 5, 5, 128)         73856
```

```
_________________________________________________________________
batch_normalization_10 (Batc (None, 5, 5, 128)         512
_________________________________________________________________
leaky_re_lu_11 (LeakyReLU)   (None, 5, 5, 128)         0
_________________________________________________________________
dropout_11 (Dropout)         (None, 5, 5, 128)         0
_________________________________________________________________
conv2d_15 (Conv2D)           (None, 5, 5, 256)         295168
_________________________________________________________________
batch_normalization_11 (Batc (None, 5, 5, 256)         1024
_________________________________________________________________
leaky_re_lu_12 (LeakyReLU)   (None, 5, 5, 256)         0
_________________________________________________________________
dropout_12 (Dropout)         (None, 5, 5, 256)         0
_________________________________________________________________
flatten_3 (Flatten)          (None, 6400)              0
_________________________________________________________________
dense_4 (Dense)              (None, 1)                 6401
=================================================================
Total params: 396,609
Trainable params: 395,713
Non-trainable params: 896
_________________________________________________________________
```

After training for 2000 epochs the images generated look like as shown in Fig. 9

3.2.3 GAN classifier Convolutional 2D network

This classifier was trained with 100 epochs. The architecture used was:

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 32, 32, 32)        896
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 32, 32, 32)        9248
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 16, 16, 32)        0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 16, 16, 64)        18496
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 16, 16, 64)        36928
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 8, 8, 64)          0
_________________________________________________________________
conv2d_5 (Conv2D)            (None, 8, 8, 128)         73856
_________________________________________________________________
conv2d_6 (Conv2D)            (None, 8, 8, 128)         147584
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 4, 4, 128)         0
_________________________________________________________________
conv2d_7 (Conv2D)            (None, 4, 4, 256)         295168
```

Figure 9: GAN 2 Generated images

```
_________________________________________________________________
conv2d_8 (Conv2D)            (None, 4, 4, 256)         590080
_________________________________________________________________
max_pooling2d_4 (MaxPooling2 (None, 2, 2, 256)         0
_________________________________________________________________
flatten_1 (Flatten)          (None, 1024)              0
_________________________________________________________________
dense_1 (Dense)              (None, 256)               262400
_________________________________________________________________
dropout_1 (Dropout)          (None, 256)               0
_________________________________________________________________
dense_2 (Dense)              (None, 256)               65792
_________________________________________________________________
dropout_2 (Dropout)          (None, 256)               0
_________________________________________________________________
dense_3 (Dense)              (None, 3)                 771
=================================================================
Total params: 1,501,219
Trainable params: 1,501,219
Non-trainable params: 0
_________________________________________________________________
```

This was a 3 class classifier used to classify between GAN-1 generated, GAN-2 generated and CIFAR10 images. After training for 100 epochs, the training accuracy obtained was 97.72% and the test accuracy was 88.44%.

Similarly, the binary classifier used to classify between GAN-1 and GAN-2 generated images. After training for 100 epochs, the training accuracy obtained was 89.69% and the test accuracy was 89.85%.

Another experiment was conducted to determine if the quality of images generated had any impact on the accuracy of the classifier. To do this, GAN models were saved after every 200 epochs and 10000 images were generated for each GAN at every step. So, at each step, the classifier was trained for 50 epochs and the validation accuracy was recorded. Then the plot of GAN epochs vs classifier accuracy was plotted as shown in Fig 10. The results remained inconclusive, however.

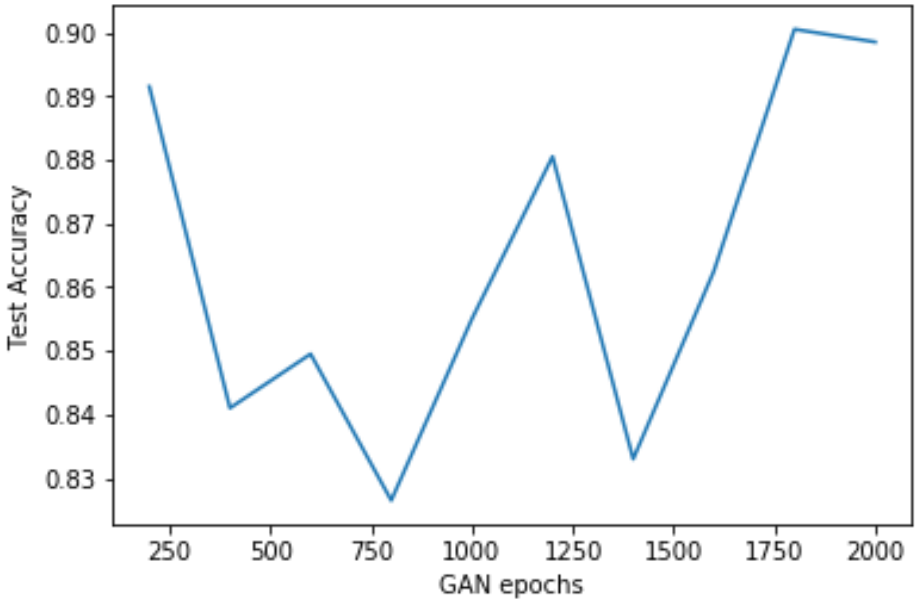


Figure 10: Validation Accuracy vs GAN Training Epochs

3.3 Model Inversion technique for GAN authorship

In this technique, to find the author model of the GAN generated image, an inverse model is trained that learns the latent space of a given GAN generator. The basic idea is shown in Fig. 11. In our case, the GAN1 generator is used to generate 20,000 images using random 100 dimensional vectors. This dataset generated is then used to train the inverse model with the following architecture:

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 32, 32, 32)        896
_________________________________________________________________
activation_1 (Activation)    (None, 32, 32, 32)        0
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 16, 16, 32)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 16, 16, 64)        18496
```

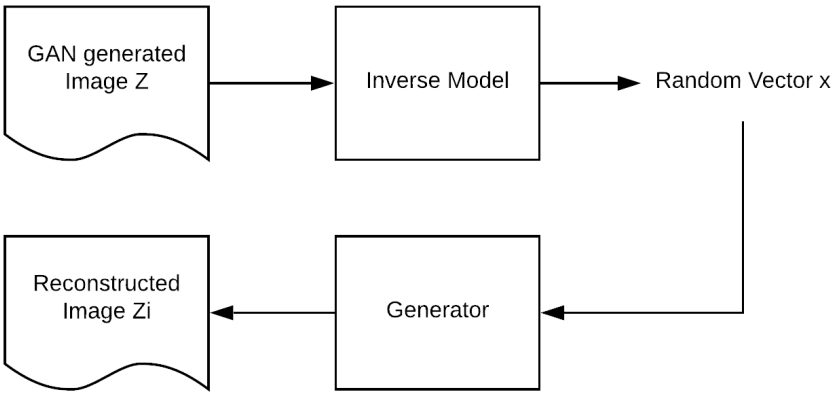


Figure 11: Inverse Model Technique

```
_________________________________________________________________
activation_2 (Activation)    (None, 16, 16, 64)        0
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 8, 8, 64)          0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 8, 8, 128)         73856
_________________________________________________________________
activation_3 (Activation)    (None, 8, 8, 128)         0
_________________________________________________________________
flatten_1 (Flatten)          (None, 8192)              0
_________________________________________________________________
dense_1 (Dense)              (None, 100)               819300
=================================================================
Total params: 912,548
Trainable params: 912,548
Non-trainable params: 0
_________________________________________________________________
```

After training for 100 epochs, the cosine proximity of 0.91 is obtained. The inverse model will have an input image and output as the corresponding random vector.

It is expected that the L2 norm difference between input image Z and reconstructed image Z_i will be minimum in case the images used are the same used to train the generator. Using this idea and calculating the distance between each image, we get the plot as shown in Fig. 12. Here, the red images represent the ones which were used to train the GAN generator. The blue ones are the GAN2 generated images. If a cutoff value of 19.12 for classifying the 2 set of images is used then an accuracy of 86.8% is obtained using this technique.

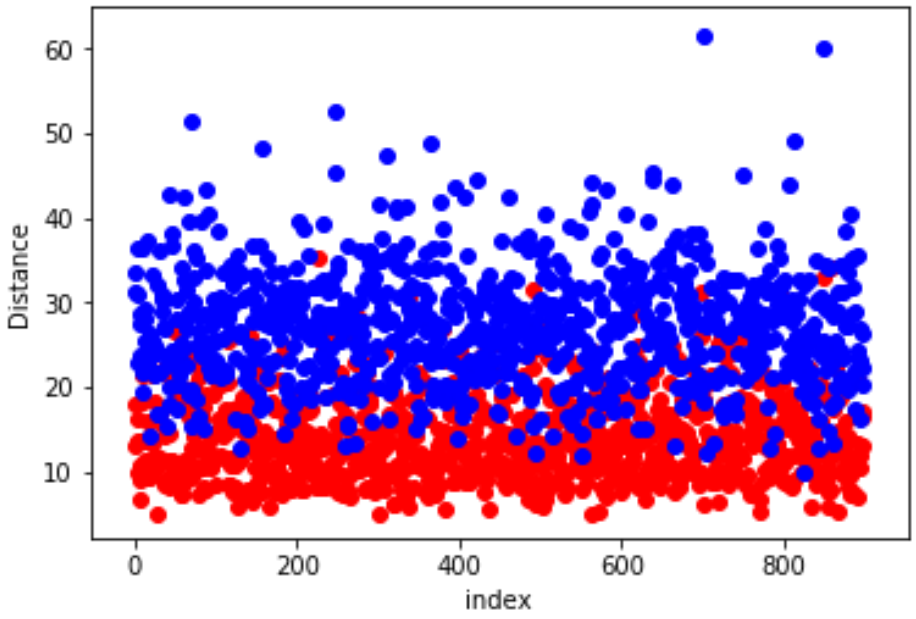


Figure 12: The reconstructed images from the same generator have relatively lesser distance.