

# Let's Generate Fake Images!

---

EE4025 - INDEPENDENT PROJECT

Hema Varshita - EE17BTECH11022  
Vamshika. K - EE17BTECH11045

# GENERATIVE ADVERSARIAL NETWORKS

---

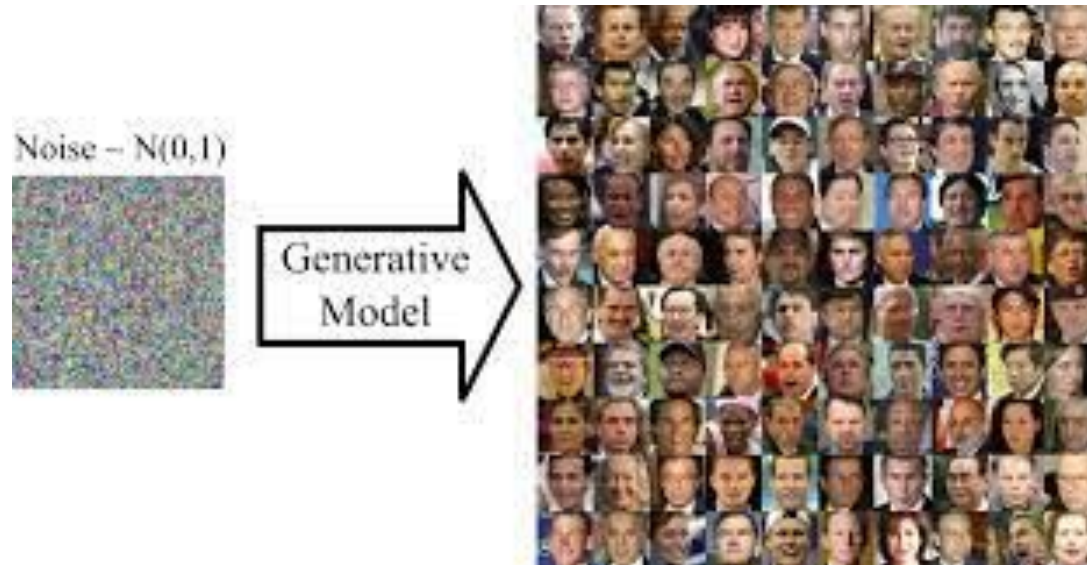
# What are GANs?

- Generative = Learn a generative model
- Adversarial = Trained in an adversarial setting
- Networks = Use Deep Neural Networks

# GENERATOR

- Consider  $y = f(x)$
- Generator is a network that is going to learn the function without explicitly knowing the function.
- This is an unsupervised form of learning because the generator is not shown labelled data beforehand.

# GENERATOR



Z ---> Image

Generator Takes random noise input and outputs new images that the network has never seen before

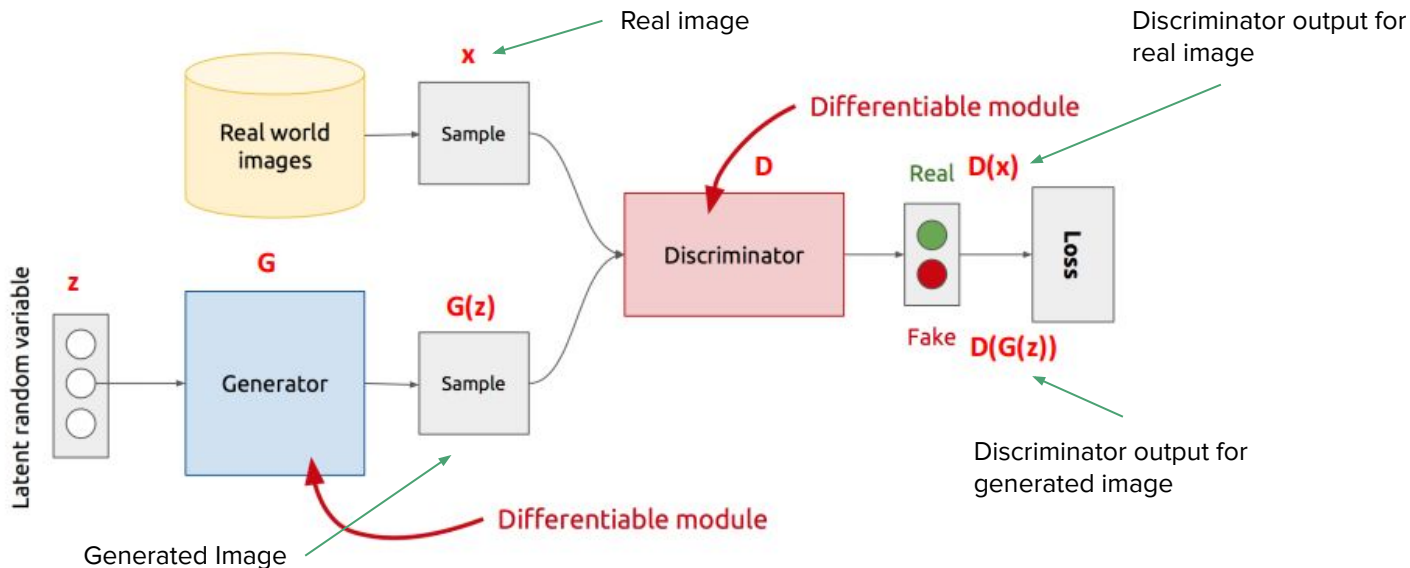
# DISCRIMINATOR



Discriminator takes the generated images and real images as input and tries to tell if the generated image input is real or fake.

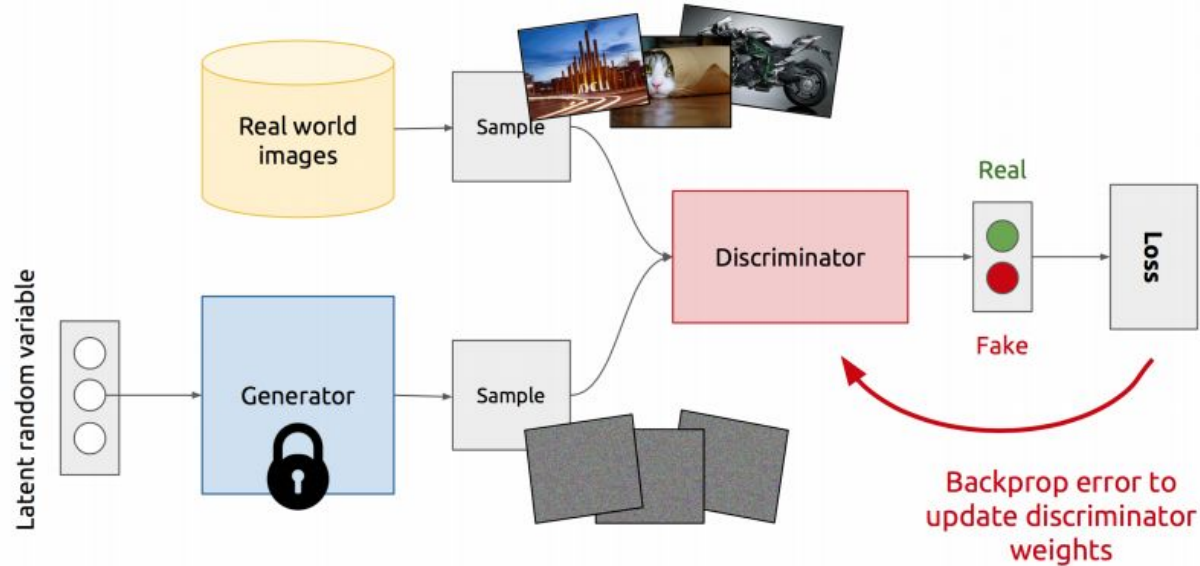
# Architecture

- $Z$  is some random Gaussian/Uniform noise which can be thought of as the latent representation of image.



# Training Of Discriminator

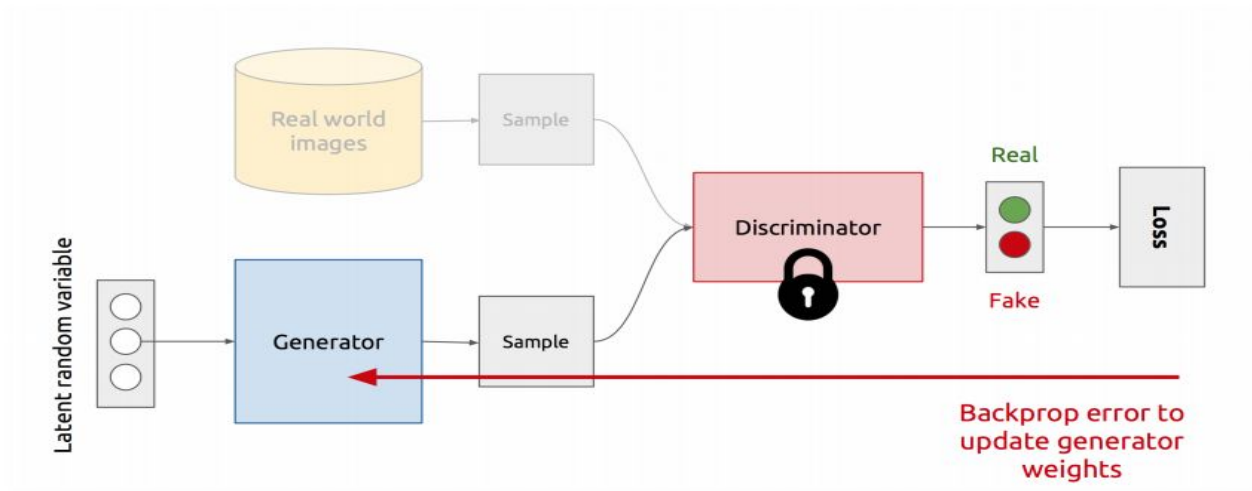
- We train D to maximize the probability of assigning the correct label to both training examples and samples from G.





# Training Of Generator

- We simultaneously train G to minimize  $\log(1 - D(G(z)))$  where  $z$  is gaussian/uniform distributed random noise.
- Both the generator and discriminator are trained against each other until we get better results.

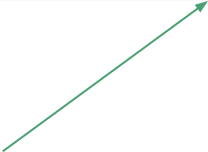


# Mathematical Formulation


- The problem is formulated as a minmax game with value function  $\mathbf{V(D,G)}$  as objective function.
- In this min-max game discriminator is trying to maximise the reward  $V$  while generator is trying to minimise  $V$  or maximising losses.

$$\min_G \max_D V_{GAN}(D, G) = \mathbb{E}_{x \sim p_{data}(x)} [\log D(x)] + \mathbb{E}_{z \sim p_z(z)} [\log(1 - D(G(z)))]$$

$D(x)$  = Discriminator output when input is a real image



$D(G(z))$  = Discriminator output when input is a generated image



# Minibatch stochastic gradient descent training of GANs

Computing the gradient of the loss function, and then update the parameters to min/max the loss function

---

**for** number of training iterations **do**

**for**  $k$  steps **do**

- Sample minibatch of  $m$  noise samples  $\{z^{(1)}, \dots, z^{(m)}\}$  from noise prior  $p_g(z)$ .
- Sample minibatch of  $m$  examples  $\{x^{(1)}, \dots, x^{(m)}\}$  from data generating distribution  $p_{\text{data}}(x)$ .
- Update the discriminator by ascending its stochastic gradient:

Gradient w.r.t the  
parameters of the  
Discriminator

$$\nabla_{\theta_d} \left[ \frac{1}{m} \sum_{i=1}^m \left[ \log D(x^{(i)}) + \log (1 - D(G(z^{(i)}))) \right] \right]$$

Loss function to  
maximize for the  
Discriminator

**end for**

- Sample minibatch of  $m$  noise samples  $\{z^{(1)}, \dots, z^{(m)}\}$  from noise prior  $p_g(z)$ .
- Update the generator by descending its stochastic gradient:

Gradient w.r.t the parameters of  
the Generator

$$\nabla_{\theta_g} \left[ \frac{1}{m} \sum_{i=1}^m \log (1 - D(G(z^{(i)}))) \right]$$

Loss function to minimize  
for the Generator

**end for**

The gradient-based updates can use any standard gradient-based learning rule. We used momentum in our experiments.

---

# Nash Equilibrium Of Minmax game

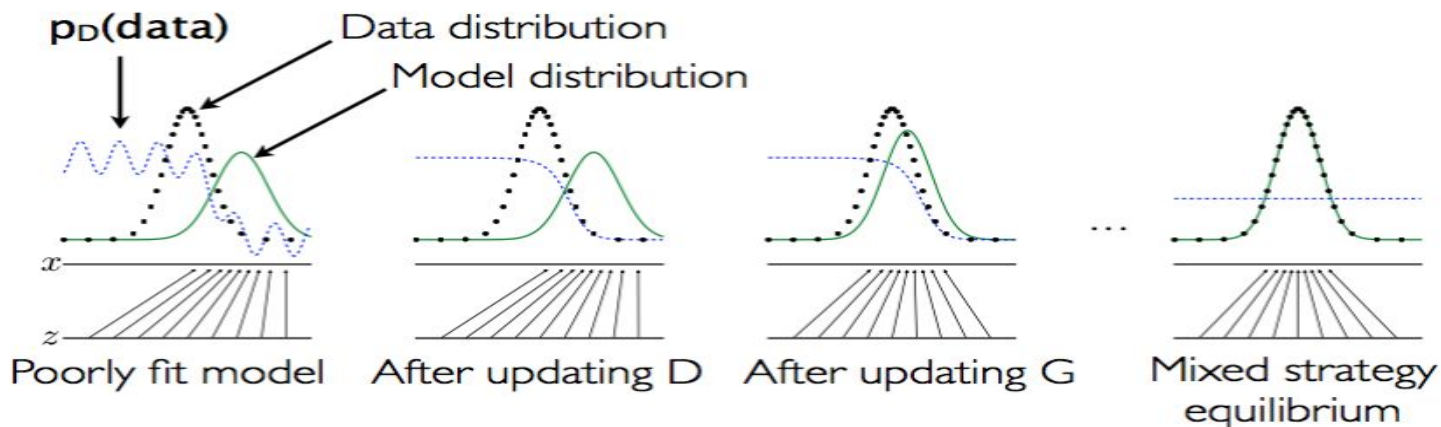
- The global optimum of the minmax game is called the Nash equilibrium.
- The following are the equilibrium points for our objective function where discriminator can't discriminate anymore.

$$\begin{aligned} P_{data}(x) &= P_{gen}(x) \quad \forall x \\ D(x) &= \frac{1}{2} \quad \forall x \end{aligned}$$

$P_{data}(x)$  = data  
distribution of real  
images

$P_{gen}(x)$  = data  
distribution of  
generated images

# Learning Process



**Observations:** It can be observed that the gradient of  $D$  has eventually guided  $G(x)$  to flow in directions where it can be classified as real data after every update. The final plot is after several iterations where  $p_g = p_{\text{data}}$  and  $D(x) = \frac{1}{2}$  where the discriminator is unable to discriminate anymore.

# Theoretical Results

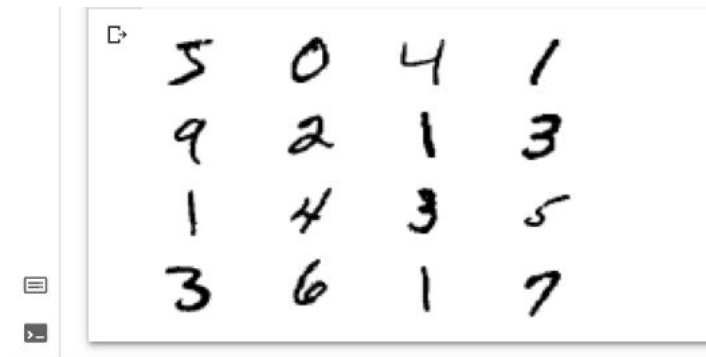
- The global optimum for the model is unique.
- Generator distribution corresponds to data distribution.
- At optimum, discriminators output 0.5 since it can't tell the difference between fake generated images and real images.
- For a fixed generator, the optimal discriminator is as follows

$$D_G^*(\mathbf{x}) = \frac{p_{data}(\mathbf{x})}{p_{data}(\mathbf{x}) + p_g(\mathbf{x})}$$

# IMPLEMENTATION

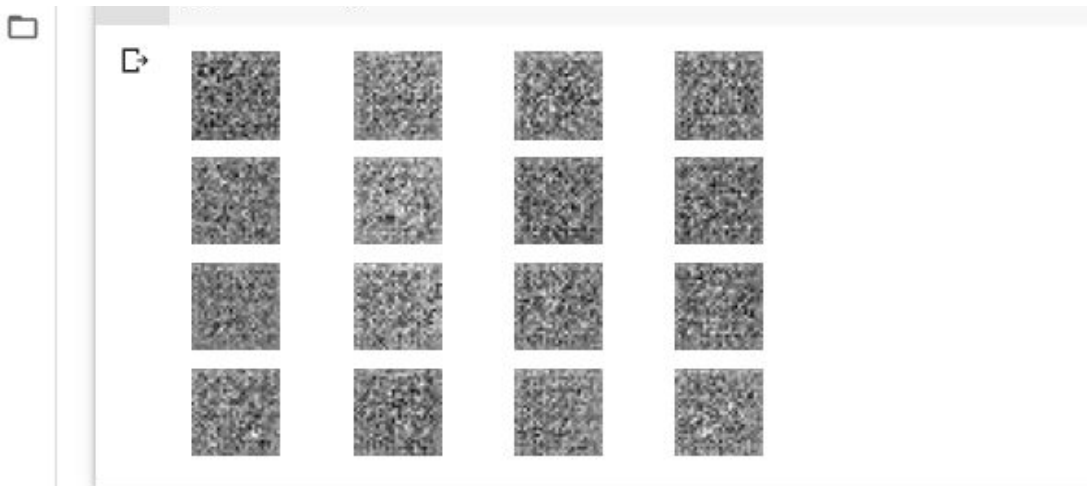
## DATASET DETAILS

- MNIST dataset consists of 70,000 28x28 greyscale images of handwritten digits from 0 to 9 among which 60,000 are training dataset and 10,000 for testing data.
- The pixel values of these images are unsigned integers between 0 and 255. The pixel value of 0 is black while the rest are in white depending on the values near to 255.



# IMPLEMENTATION

- Generated 2D 28x28 images from noise before training.





# IMPLEMENTATION

Model: "sequential\_3"

Layer (type)	Output Shape	Param #
conv2d_3 (Conv2D)	(None, 14, 14, 64)	640
leaky_re_lu_9 (LeakyReLU)	(None, 14, 14, 64)	0
dropout (Dropout)	(None, 14, 14, 64)	0
conv2d_4 (Conv2D)	(None, 7, 7, 64)	36928
leaky_re_lu_10 (LeakyReLU)	(None, 7, 7, 64)	0
dropout_1 (Dropout)	(None, 7, 7, 64)	0
flatten (Flatten)	(None, 3136)	0
dense_3 (Dense)	(None, 1)	3137

Total params: 40,705  
Trainable params: 40,705  
Non-trainable params: 0

DISCRIMINATOR

Model: "sequential\_2"

Layer (type)	Output Shape	Param #
dense_2 (Dense)	(None, 6272)	633472
leaky_re_lu_6 (LeakyReLU)	(None, 6272)	0
reshape_2 (Reshape)	(None, 7, 7, 128)	0
conv2d_transpose_4 (Conv2DTr	(None, 14, 14, 128)	262272
leaky_re_lu_7 (LeakyReLU)	(None, 14, 14, 128)	0
conv2d_transpose_5 (Conv2DTr	(None, 28, 28, 128)	262272
leaky_re_lu_8 (LeakyReLU)	(None, 28, 28, 128)	0
conv2d_2 (Conv2D)	(None, 28, 28, 1)	6273

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

GENERATOR

# Discriminator, generator losses during training and final accuracies achieved



```
>1, 1/234, d=0.691, g=0.721  
>1, 2/234, d=0.687, g=0.736  
>1, 3/234, d=0.671, g=0.760  
>1, 4/234, d=0.665, g=0.781  
>1, 5/234, d=0.662, g=0.797  
>1, 6/234, d=0.649, g=0.820  
>1, 7/234, d=0.647, g=0.833  
>1, 8/234, d=0.642, g=0.842  
>1, 9/234, d=0.640, g=0.840
```



```
>10, 227/234, d=0.680, g=0.673  
>10, 228/234, d=0.697, g=0.670  
>10, 229/234, d=0.692, g=0.673  
>10, 230/234, d=0.696, g=0.695  
>10, 231/234, d=0.684, g=0.716  
>10, 232/234, d=0.697, g=0.738  
>10, 233/234, d=0.690, g=0.737  
>10, 234/234, d=0.688, g=0.743  
>Accuracy real: 32%, fake: 83%
```

# Observations

- It can be observed that the discriminator and generator losses remain stable indicating that the generator model has not started generating rubbish examples that the discriminator can easily discriminate and we can continue to train.
- The accuracy given by discriminator of real images(32%) is low compared to fake images(83%) for generated images and so it can be said that the image quality might affect the accuracy of real images compared to fake images.

# IMPLEMENTATION RESULTS - Mnist



Final results after 10 Epochs

# DEEP CONVOLUTIONAL GENERATIVE ADVERSARIAL NETWORKS

---

# Generator and Discriminator

## GENERATOR

- Built on specific architectures described by the authors
- Goal is to fool discriminator such that  $D(G(z)) = 1$

## DISCRIMINATOR

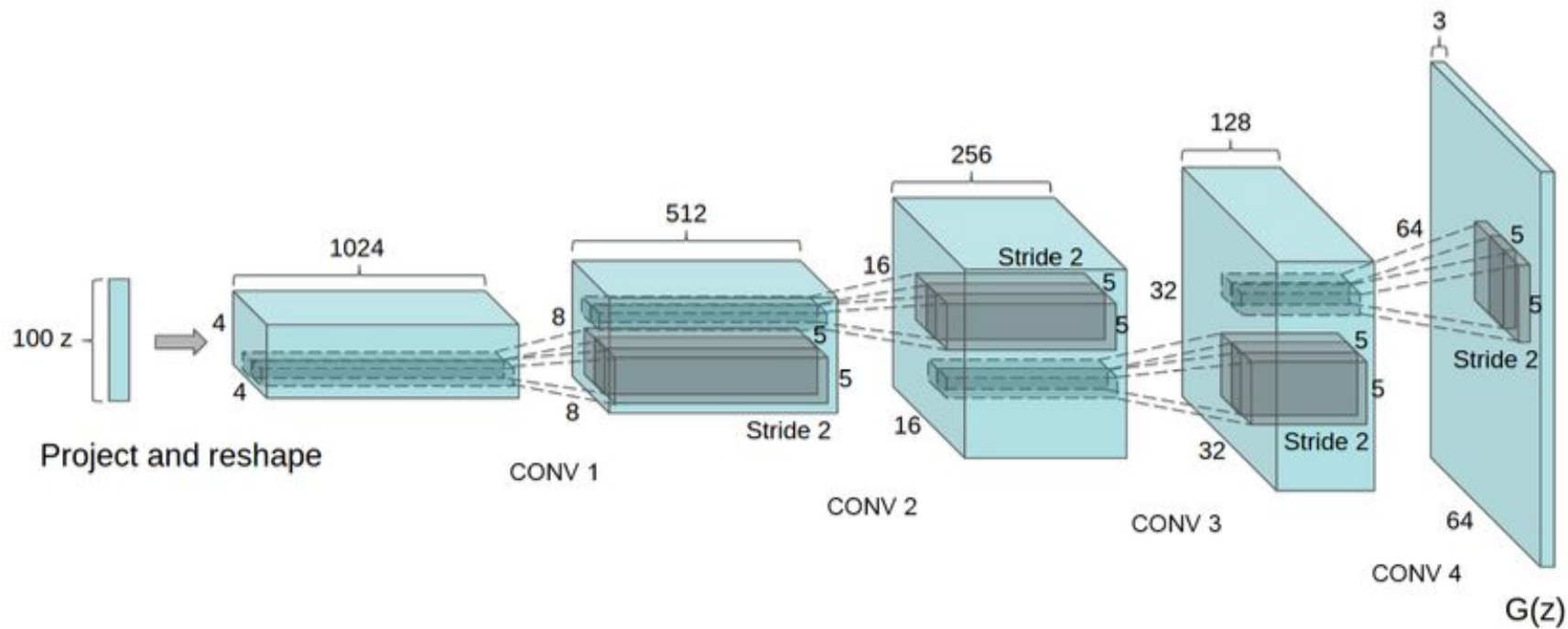
- Any CNN based classifier giving output of 1 class (fake or real) with batchnorm and LeakyReLU activation func can be used.
- If  $x$  is a real image and  $G(z)$  is a generated image then  $D(x) = 1$  and  $D(G(z)) = 0$

- These two conflicting unsupervised goals form the basis of adversarial learning.
- The result is called optimal when  $D(G(z)) = 0.5$ , i.e., when the model cannot determine if the sample is real or generated.

## **OBJECTIVE**

To use GANs with Convolutional Neural Networks and ensuring a stable model architecture with the help of slight modifications

# Generator Architecture





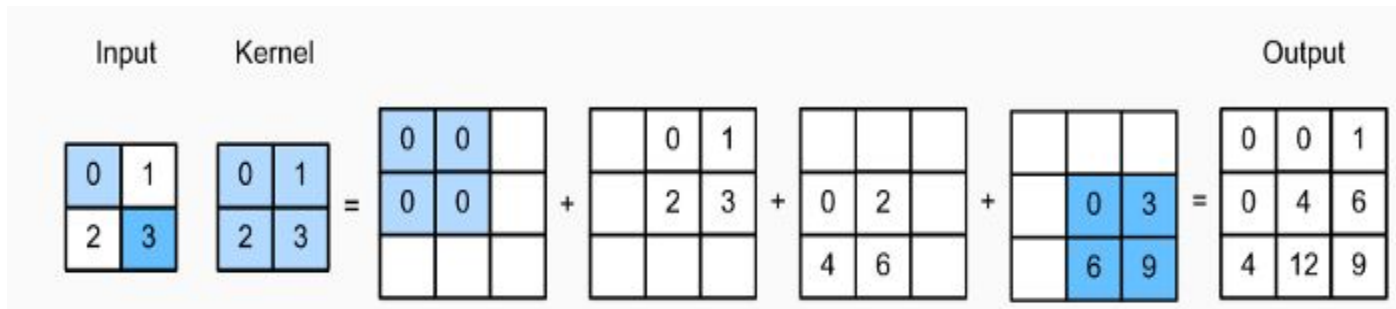
# Generator architecture

- The input to the generator ( $Z$ ) is a 100 dimensional random noise vector.
- The noise vector is transformed into a  $4 \times 4$  tensor with 1024 channels.
- In each subsequent step, the height and width of the tensor is upsampled by a factor of two, and the number of channels is downsampled by a factor of two.
- At the last step, an image output  $G(Z)$  of size  $64 \times 64$  comprising 3 channels, namely RGB is generated.

# Generator Architecture

- The upsampling of height and width and downsampling of the number of channels. But how does that happen?
- This is enabled using “TRANSPOSED CONVOLUTION”.
- It up samples the input feature map to a desired output feature map.

# Transposed Convolution



- The 2x2 input feature map is upsampled to a 3x3 output feature map.
- Each element of the input matrix is multiplied with the kernel and all such resultant matrices are added to form the final matrix.

# Architecture Guidelines for stable DCGANs

- Replace any pooling layers with strided convolutions (discriminator) and fractional-strided convolutions (generator).
- Use batchnorm in both the generator and the discriminator.
- Remove fully connected hidden layers for deeper architectures.
- Use ReLU activation in the generator for all layers except for the output, which uses Tanh.
- Use LeakyReLU activation in the discriminator for all layers.

These guidelines were summarized after multiple attempts by the authors to train a stable model.

Replace any pooling layers with strided convolutions (discriminator) and fractional-strided convolutions (generator).

Both pooling layers and strided convolutions, give us the same end results. I.e. a compressed feature map. For stability, authors recommend replacing pooling layers with strided convolutions in the discriminator.

However, for the generator, we must perform a fractional-strided convolution, which is roughly same as Transpose Convolution.

## Use batchnorm in both the generator and the discriminator.

Batchnorm would normalize input to be zero mean and unit variance ensuring equal importance to each sample. This in turn helps to overcome problems that occur due to poor initialization or weights exploding to nan etc.

## Remove Fully Connected Layers for deeper architectures

Connecting highest convolutional layer features to input of generator and output of discriminator gave better results.

- Use ReLU activation in the generator for all layers except for the output, which uses Tanh.
- Use LeakyReLU activation in the discriminator for all layers.

Both of these results were obtained after trial and error.

However, one important point to be noted is that in this model LeakyReLU is used in discriminator, but in the original GAN paper, maxout activation function was used.

# Is the model memorizing images?

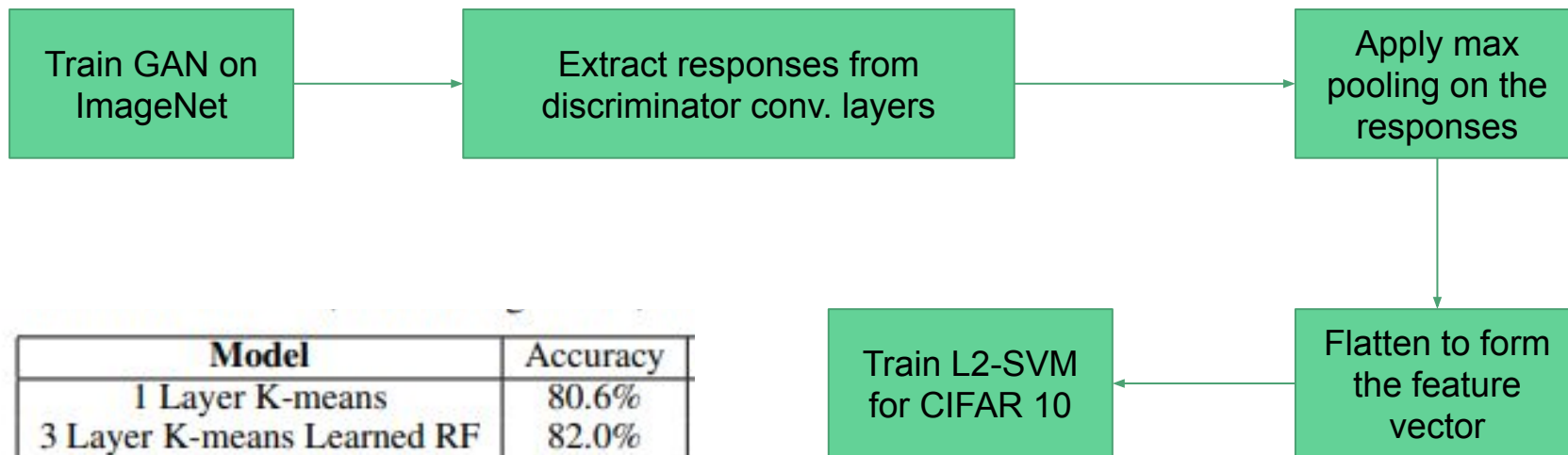
- To find this out, Intermediate samples from the trainings were examined.
- It was found that there were no changes like sudden addition or removal objects in the images, implying there were no “sharp transitions” in the latent space.
- This demonstrates that the model is not just producing samples by memorizing training examples.





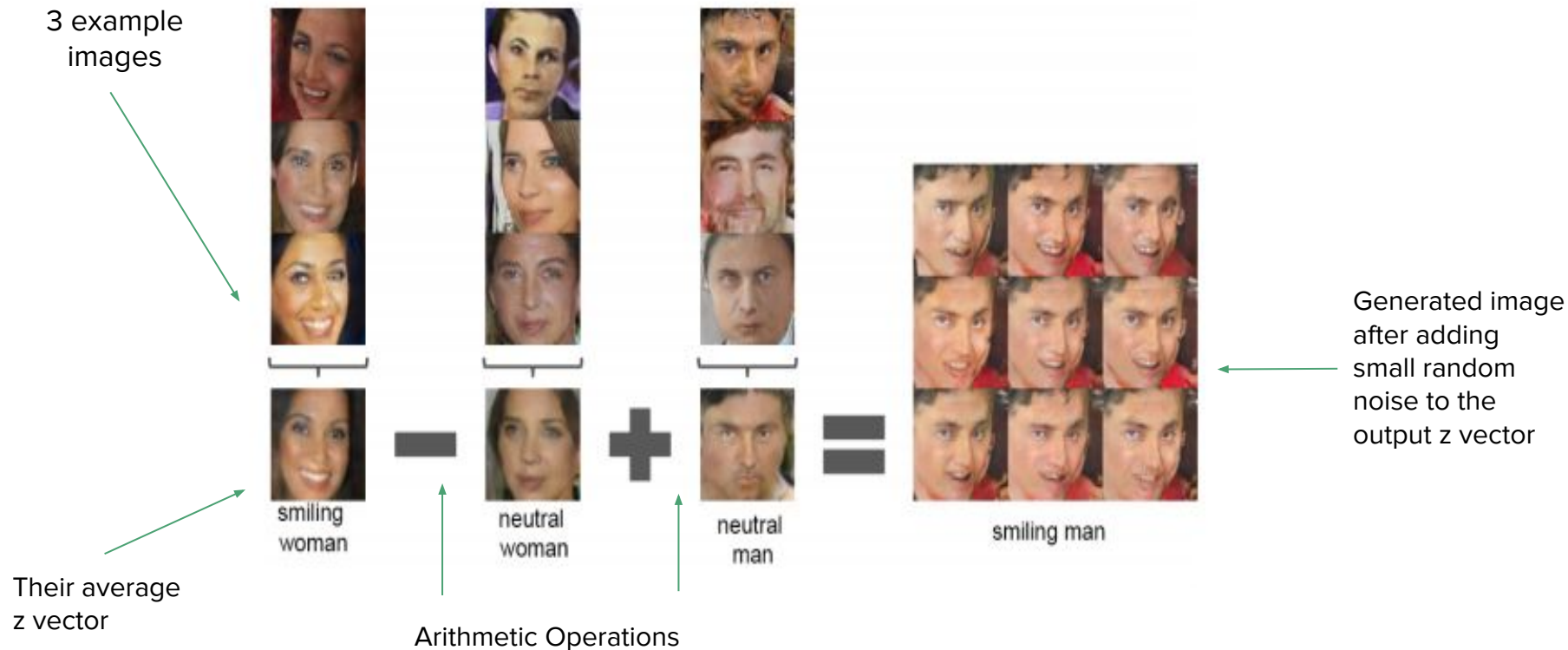
There are only smooth transitions in each row. Eg : In first row, the room is gradually transforming into a room with a window

# GANs as Feature Extractors



Model	Accuracy
1 Layer K-means	80.6%
3 Layer K-means Learned RF	82.0%
View Invariant K-means	81.9%
Exemplar CNN	84.3%
DCGAN (ours) + L2-SVM	82.8%

# Vector Arithmetic on Face Samples



# INSIGHTS

- DCGANs leverage Unsupervised mode of Learning.
- The network architecture comprises of CNNs with constraints.
- Stochastic gradient descent with Adam optimizer for both generator and discriminator is used.
- The objective of DCGAN is to learn hierarchy of representations from object parts.
- The performance metrics used were accuracy and error rate

# IMPLEMENTATION

- We've seen the brief overview of results from the paper and analysis of each step finalised by the authors.
- For the implementation part, following the guidelines, we build our generator and discriminator models. Due to limited computational resources, we only run 50 epochs, that gives just good enough result to prove the concept.
- However, exact results of the paper could not be replicated.

Link to the code : <https://github.com/Varshita0307/GANs>

# IMPLEMENTATION

## CODES USED

Model: "sequential"

Layer (type)	Output Shape	Param #
dense (Dense)	(None, 12544)	1254400
batch_normalization (Batch Normalization)	(None, 12544)	50176
leaky_re_lu (LeakyReLU)	(None, 12544)	0
reshape (Reshape)	(None, 7, 7, 256)	0
conv2d_transpose (Conv2DTranspose)	(None, 7, 7, 128)	819200
batch_normalization_1 (Batch Normalization)	(None, 7, 7, 128)	512
leaky_re_lu_1 (LeakyReLU)	(None, 7, 7, 128)	0
conv2d_transpose_1 (Conv2DTranspose)	(None, 14, 14, 64)	204800
batch_normalization_2 (Batch Normalization)	(None, 14, 14, 64)	256
leaky_re_lu_2 (LeakyReLU)	(None, 14, 14, 64)	0
conv2d_transpose_2 (Conv2DTranspose)	(None, 28, 28, 1)	1600
Total params: 2,330,944		
Trainable params: 2,305,472		
Non-trainable params: 25,472		

GENERATOR

Model: "sequential\_1"

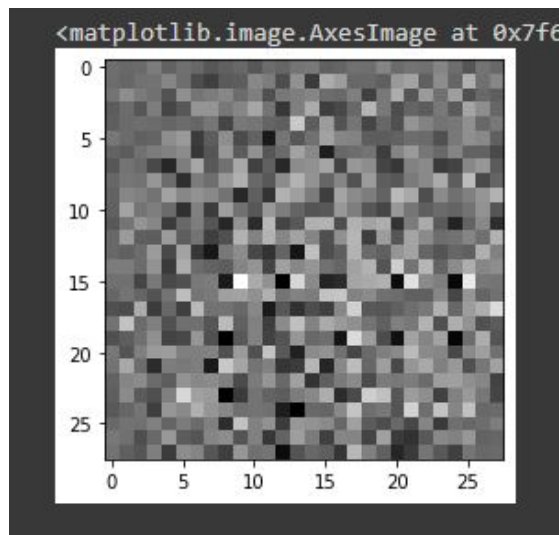
Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 14, 14, 64)	1664
leaky_re_lu_3 (LeakyReLU)	(None, 14, 14, 64)	0
dropout (Dropout)	(None, 14, 14, 64)	0
conv2d_1 (Conv2D)	(None, 7, 7, 128)	204928
leaky_re_lu_4 (LeakyReLU)	(None, 7, 7, 128)	0
dropout_1 (Dropout)	(None, 7, 7, 128)	0
flatten (Flatten)	(None, 6272)	0
dense_1 (Dense)	(None, 1)	6273
Total params: 212,865		
Trainable params: 212,865		
Non-trainable params: 0		

DISCRIMINATOR

# IMPLEMENTATION

- Both the datasets have 70,000 images sized 28x28.
- 60,000 samples are used in the training data
- A 100 dimensional dimensional vector is upsampled to form a resultant 28x28x1 generated image.
- This image is input into a CNN based discriminator that outputs a value between 0 and 1

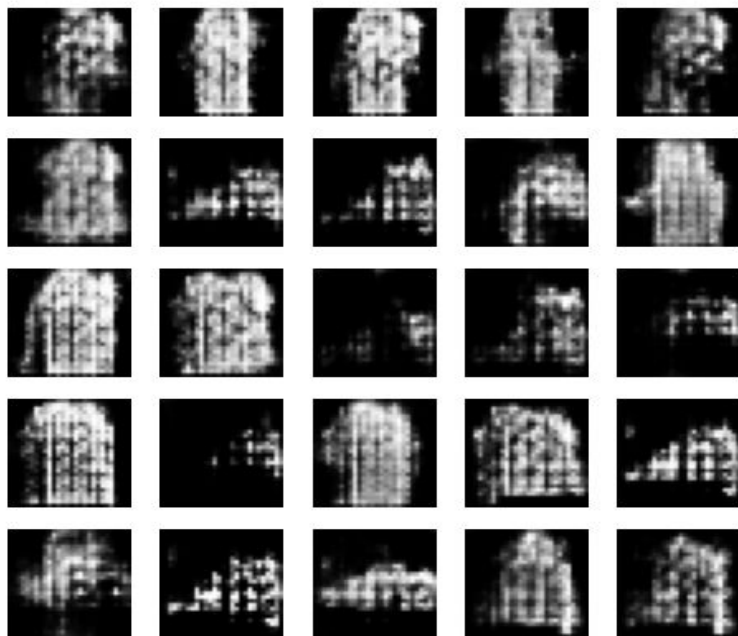
# Image and Decision from Untrained network



$$D(G(x)) = 0.49997$$



## RESULTS - Fashion MNIST

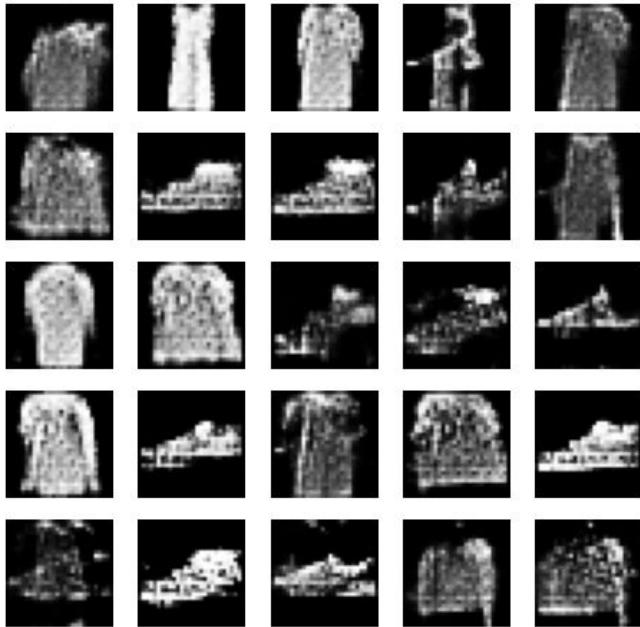


10 EPOCHS



20 EPOCHS

## RESULTS - Fashion MNIST

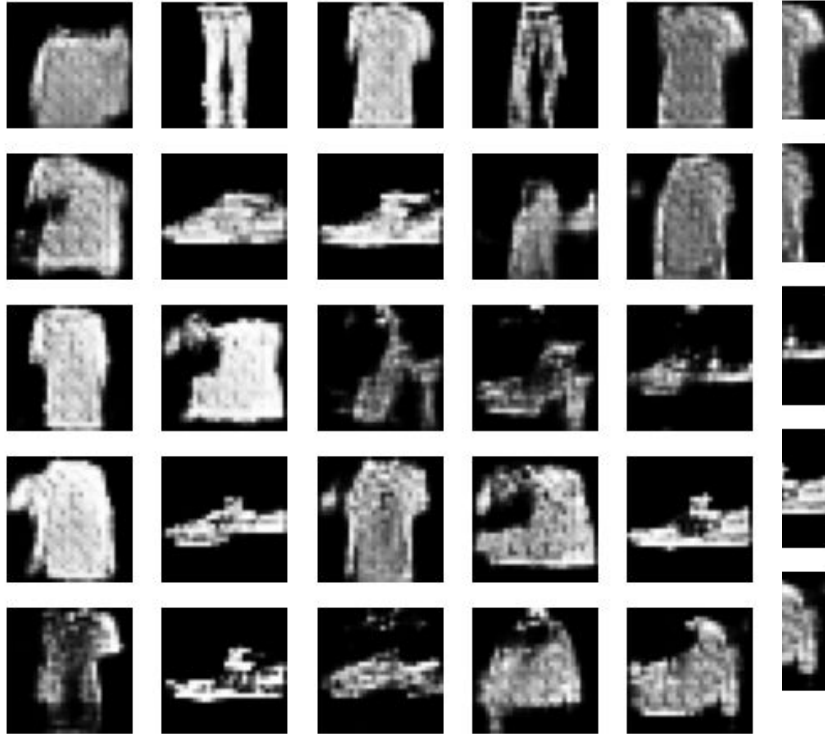


30 EPOCHS



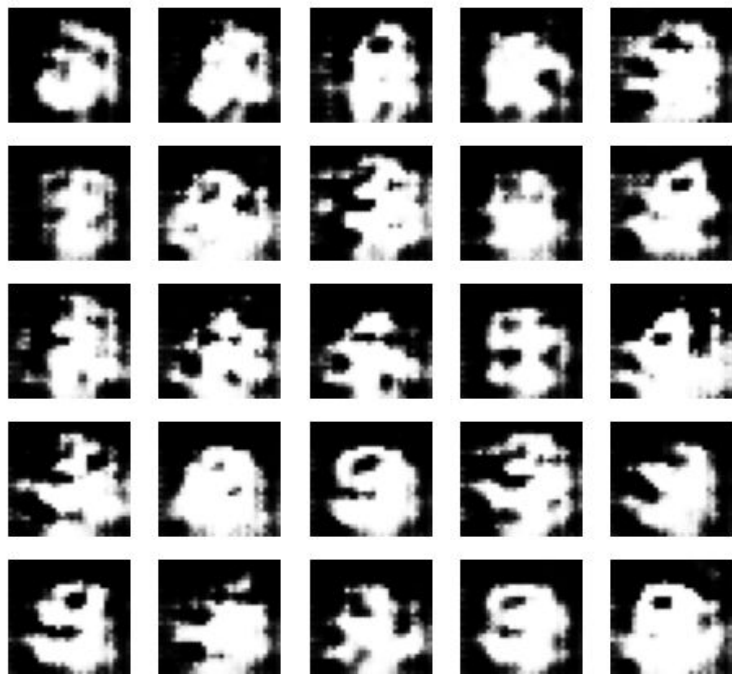
40 EPOCHS

## RESULTS - Fashion MNIST



Final Result after 50 EPOCHS

## RESULTS - MNIST



10 EPOCHS

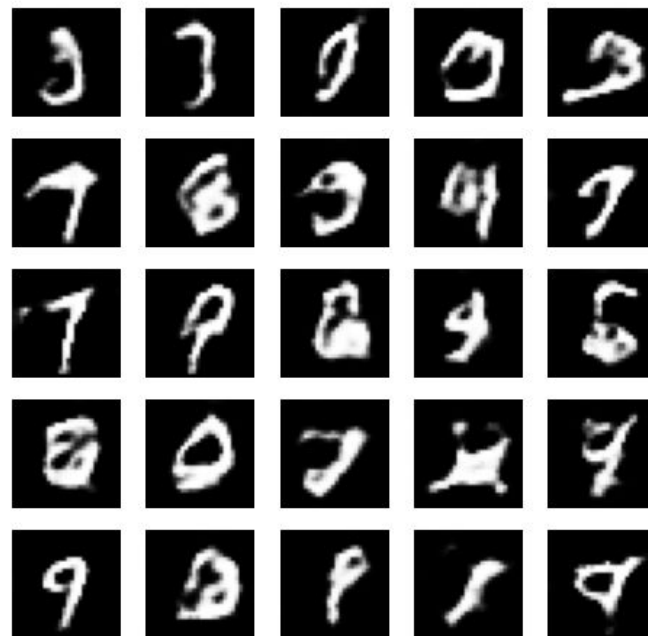


20 EPOCHS

## RESULTS - MNIST

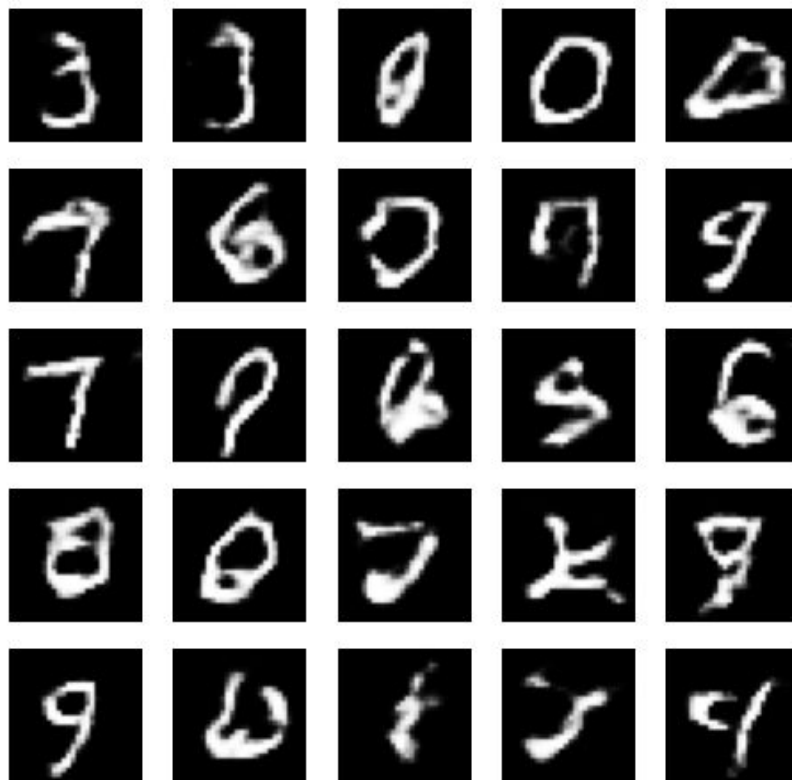


30 EPOCHS



40 EPOCHS

## RESULTS - MNIST



Final Result after 50 EPOCHS

# COMPARISON

1. Although the results are not sufficient to demonstrate, but theoretically, DCGANs must produce sharper results in lesser epochs in comparison to GANs.
2. DCGANs are specially used for image/video data whereas GANs have applications in various other domains
3. GANs, with its adversarial deep learning mechanism could be leveraged in tasks like NLP, image captioning etc.