Generative Adversarial Nets

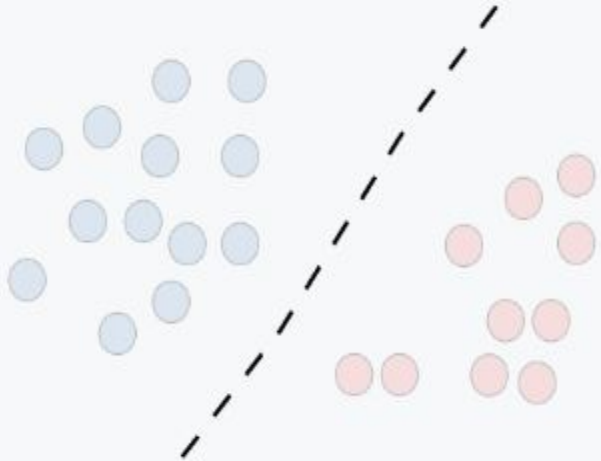
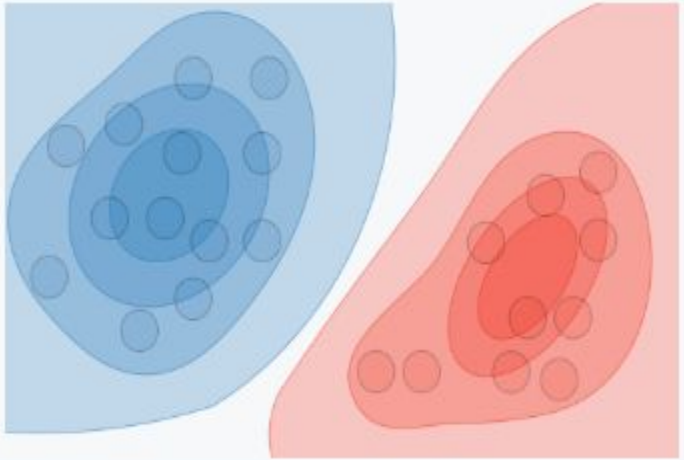
Team 24

(2020201068) Mahak Modani
(2020201026) Mansi Khamkar
(2020202019) Rani Tresa

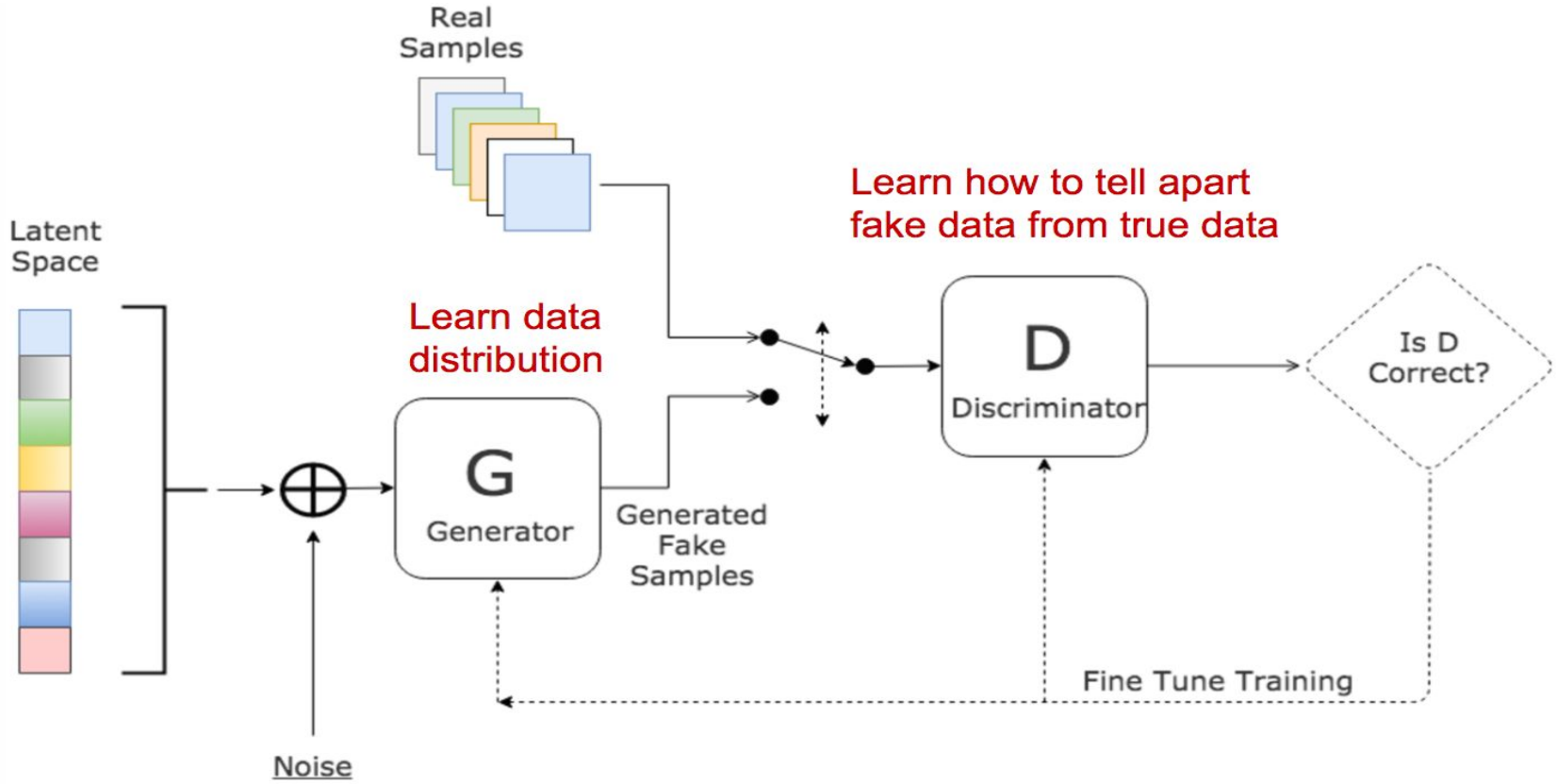
AIM

To understand and implement Generative Adversarial Networks (GANs) for unsupervised image representation learning in order to generate synthetic data that looks similar to the original data

Generative vs. Discriminative Algorithms

	Discriminative model	Generative model
Goal	Directly estimate $P(y\|x)$	Estimate $P(x\|y)$ to then deduce $P(y\|x)$
What's learned	Decision boundary	Probability distributions of the data
Illustration		

GAN Architecture



How GAN Works ?

Here are the steps a GAN takes:

- The generator takes in random numbers and returns an image.
- This generated image is fed into the discriminator alongside a stream of images taken from the actual, ground-truth dataset.
- The discriminator takes in both real and fake images and returns probabilities, a number between 0 and 1, with 1 representing a prediction of authenticity and 0 representing fake.

So you have a double feedback loop:

- The discriminator is in a feedback loop with the ground truth of the images, which we know
- The generator is in a feedback loop with the discriminator
- Generator is not updated directly with data samples, but only with gradients flowing through the Discriminator

GAN Algorithm

Algorithm 1 Minibatch stochastic gradient descent training of generative adversarial nets. The number of steps to apply to the discriminator, k, is a hyperparameter. We used $k = 1$, the least expensive option, in our experiments.

for number of training iterations **do**

  for k steps **do**

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

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

  end for

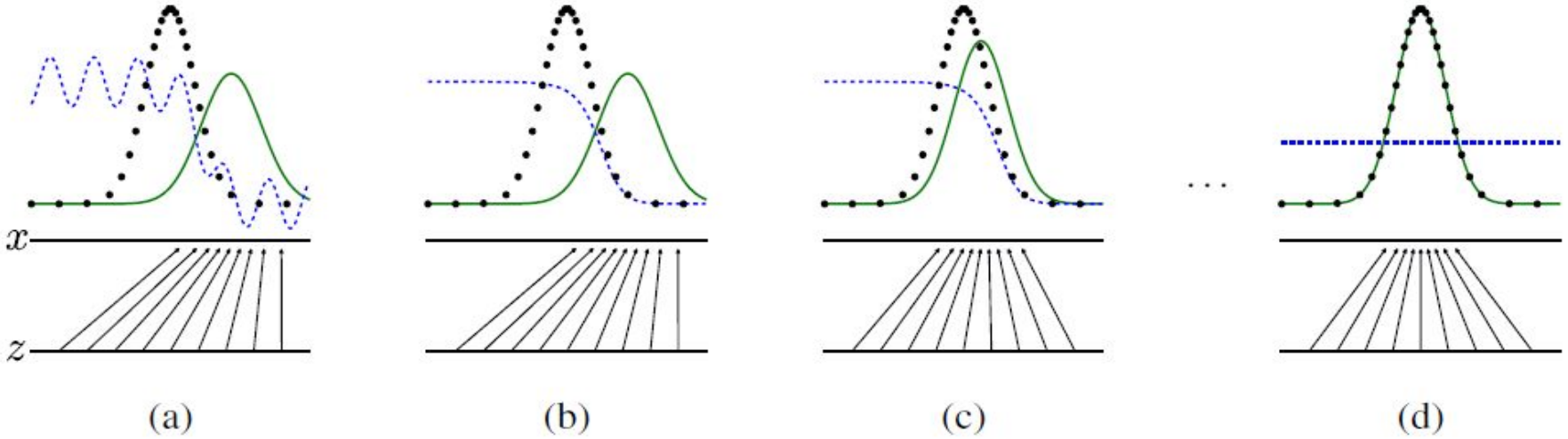
  - Sample minibatch of m noise samples $\{\boldsymbol{z}^{(1)}, \ldots, \boldsymbol{z}^{(m)}\}$ from noise prior $p_g(\boldsymbol{z})$.
  - Update the generator by descending its stochastic gradient:

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

end for

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

GAN Training



Original distribution

Generative distribution

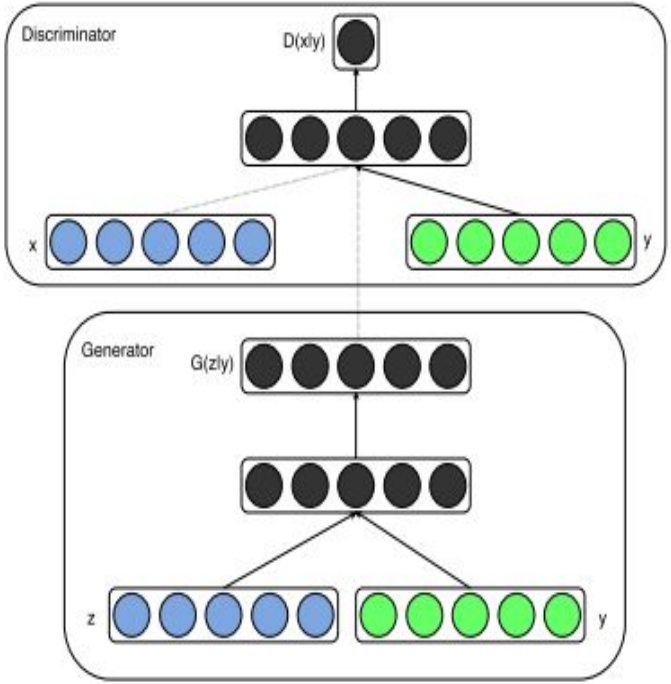
Discriminative distribution

$$D_G^*(\boldsymbol{x}) = \frac{p_{data}(\boldsymbol{x})}{p_{data}(\boldsymbol{x}) + p_g(\boldsymbol{x})} = \frac{1}{2}$$

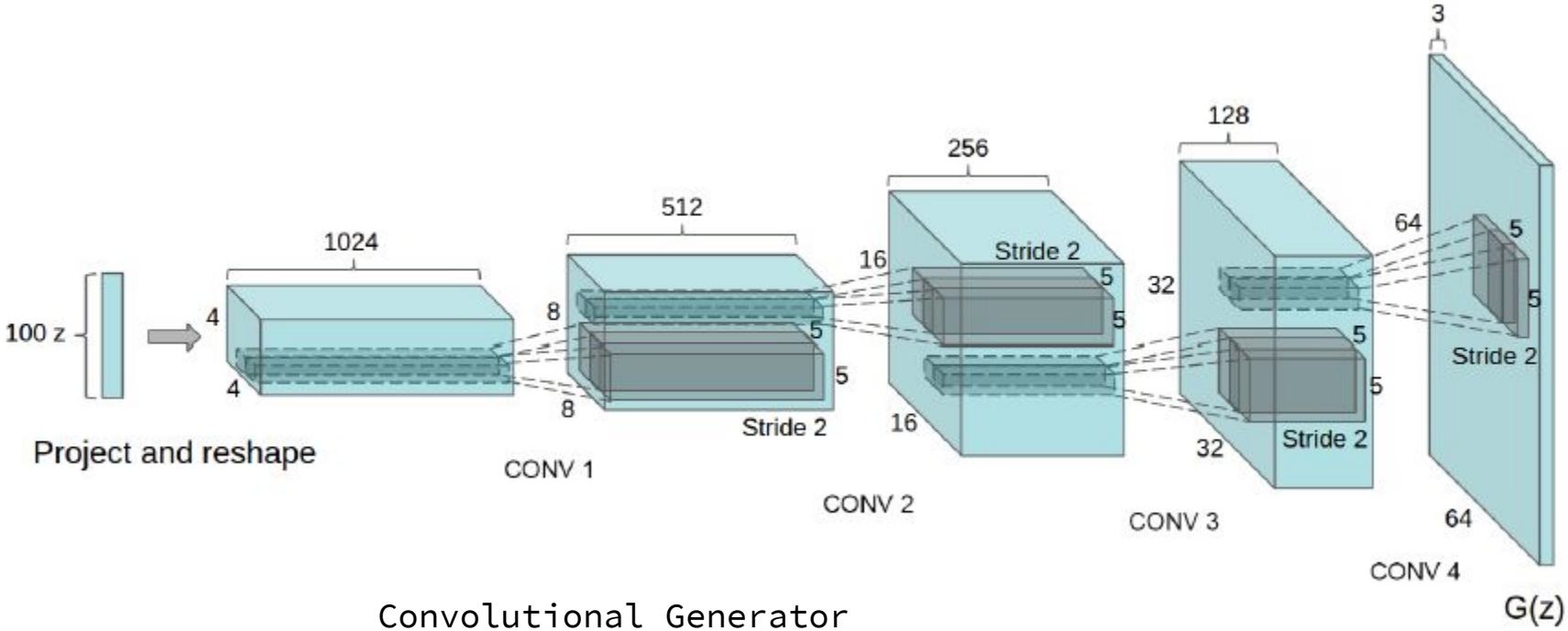
Optimal discriminator

Conditional GAN (CGAN)

- In an unconditioned generative model, there is no control on modes of the data being generated
- Both the generator and discriminator are conditioned on some extra information ‘y’
- ‘y’ can be class labels or data from other modality
- We feed ‘y’ into the both the discriminator and generator as additional input layer
- In the generator the prior input noise pz(z), and y are combined in joint hidden representation
- In the discriminator x and y are presented as inputs



Deep Convolutional GAN (DCGAN)



Convolutional Generator

Deep Convolutional GAN (DCGAN)

Architecture guidelines for stable Deep Convolutional GANs:

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

GAN

```
Generator(
  (model): Sequential(
    (0): Linear(in_features=100, out_features=128, bias=True)
    (1): LeakyReLU(negative_slope=0.2, inplace=True)
    (2): Linear(in_features=128, out_features=256, bias=True)
    (3): BatchNorm1d(256, eps=0.8, momentum=0.1, affine=True, track_running_stats=True)
    (4): LeakyReLU(negative_slope=0.2, inplace=True)
    (5): Linear(in_features=256, out_features=512, bias=True)
    (6): BatchNorm1d(512, eps=0.8, momentum=0.1, affine=True, track_running_stats=True)
    (7): LeakyReLU(negative_slope=0.2, inplace=True)
    (8): Linear(in_features=512, out_features=1024, bias=True)
    (9): BatchNorm1d(1024, eps=0.8, momentum=0.1, affine=True, track_running_stats=True)
    (10): LeakyReLU(negative_slope=0.2, inplace=True)
    (11): Linear(in_features=1024, out_features=784, bias=True)
    (12): Tanh()
  )
)
Discriminator(
  (model): Sequential(
    (0): Linear(in_features=784, out_features=512, bias=True)
    (1): LeakyReLU(negative_slope=0.2, inplace=True)
    (2): Linear(in_features=512, out_features=256, bias=True)
    (3): LeakyReLU(negative_slope=0.2, inplace=True)
    (4): Linear(in_features=256, out_features=1, bias=True)
    (5): Sigmoid()
  )
)
```

Conditional Gan

Discriminator

```
nn.Linear(opt.n_classes + int(np.prod(img_shape)), 512),
nn.LeakyReLU(0.2, inplace=True),
nn.Linear(512, 512),
nn.Dropout(0.4),
nn.LeakyReLU(0.2, inplace=True),
nn.Linear(512, 512),
nn.Dropout(0.4),
nn.LeakyReLU(0.2, inplace=True),
nn.Linear(512, 1)
```

Generator

```
nn.Linear(opt.latent_dim +
opt.n_classes, 128),
nn.LeakyReLU(0.2, inplace=True),
nn.Linear(128, 256),
nn.BatchNorm1d(256, 0.8),
nn.LeakyReLU(0.2, inplace=True),
nn.Linear(256, 512),
nn.BatchNorm1d(512, 0.8),
nn.LeakyReLU(0.2, inplace=True),
nn.Linear(512, 1024),
nn.BatchNorm1d(1024, 0.8),
nn.LeakyReLU(0.2, inplace=True),
nn.Linear(1024,
int(np.prod(img_shape))),
nn.Tanh()
```

DC GAN

Discriminator

```
nn.Conv2d(opt.channels, 16, 3, 2, 1),
nn.LeakyReLU(0.2, inplace=True),
nn.Dropout2d(0.25),
nn.Conv2d(16, 32, 3, 2, 1),
nn.LeakyReLU(0.2, inplace=True),
nn.Dropout2d(0.25),
nn.BatchNorm2d(32, 0.8),
nn.Conv2d(32, 64, 3, 2, 1),
nn.LeakyReLU(0.2, inplace=True),
nn.Dropout2d(0.25),
nn.BatchNorm2d(64, 0.8),
nn.Conv2d(64, 128, 3, 2, 1),
nn.LeakyReLU(0.2, inplace=True),
nn.Dropout2d(0.25),
nn.BatchNorm2d(128, 0.8),
out.view(out.shape[0], -1),
n=img_size/16,
nn.Sequential(nn.Linear(128 * n * n, 1),
nn.Sigmoid())
```

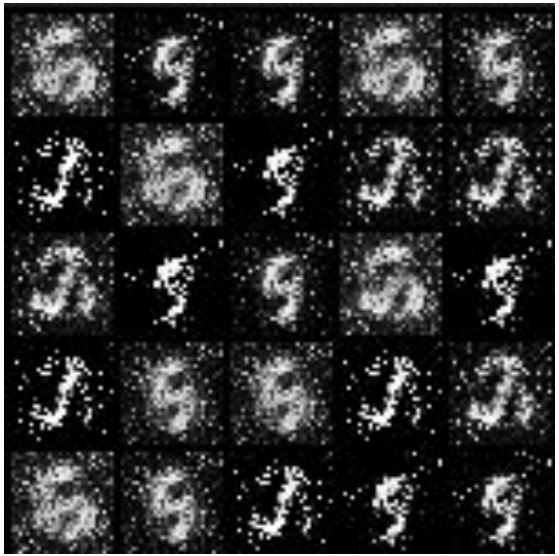
Generator

```
nn.Sequential(nn.Linear(opt.latent_dim, 128 * self.n * self.n))
n = img_size / 4
out.view(out.shape[0], 128, self.n, self.n)
nn.BatchNorm2d(128),
nn.Upsample(scale_factor=2),
nn.Conv2d(128, 128, 3, stride=1, padding=1),
nn.BatchNorm2d(128, 0.8),
nn.ReLU(True),
nn.Upsample(scale_factor=2),
nn.Conv2d(128, 64, 3, stride=1, padding=1),
nn.BatchNorm2d(64, 0.8),
nn.ReLU(True),
nn.Conv2d(64, opt.channels, 3, stride=1, padding=1),
nn.Tanh()
```

Experiments

- We implemented GAN using PyTorch framework
  - We trained GAN on: MNIST, E-MNIST, CelebA and CIFAR-10
- We further developed DC-GAN using PyTorch framework
  - We trained DCGAN on: MNIST, E-MNIST, CelebA and Fashion-MNIST
- We further developed Conditional-GAN using PyTorch framework
  - We trained the CGAN on: E-MNIST and Fashion-MNIST dataset

Results for GAN : MNIST



Epoch 0

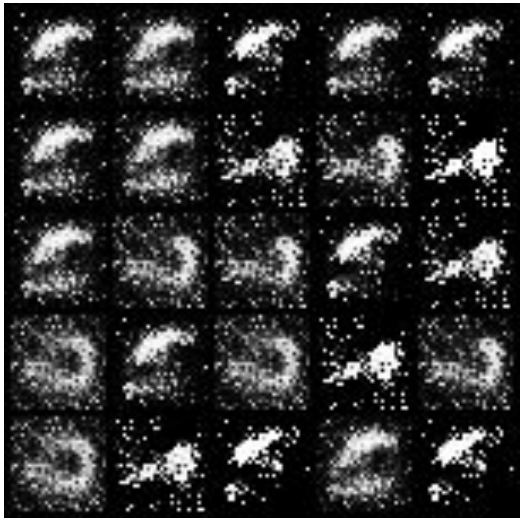
Epoch 400

Results for DCGAN : MNIST

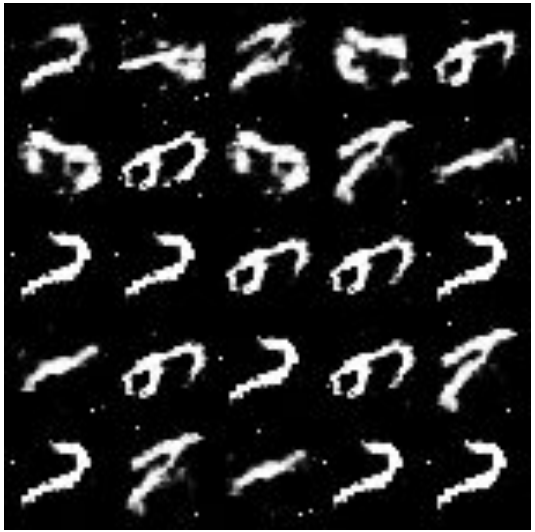
Epoch 0

Epoch 25

Results for GAN : EMNIST



Epoch 0



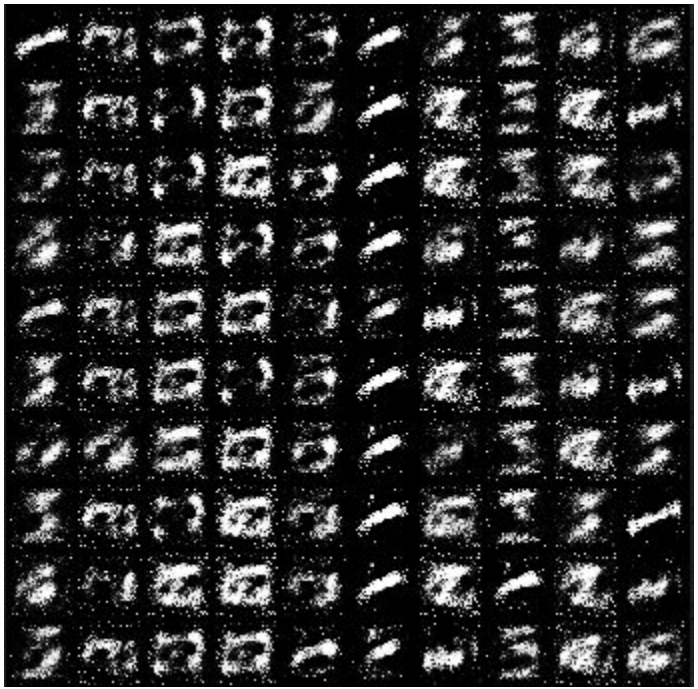
Epoch 600

Results for DCGAN : EMNIST

Epoch 0

Epoch 20

Results for CGAN : EMNIST



Epoch 0

Epoch 200

Results for GAN : CELEBA

Epoch 0

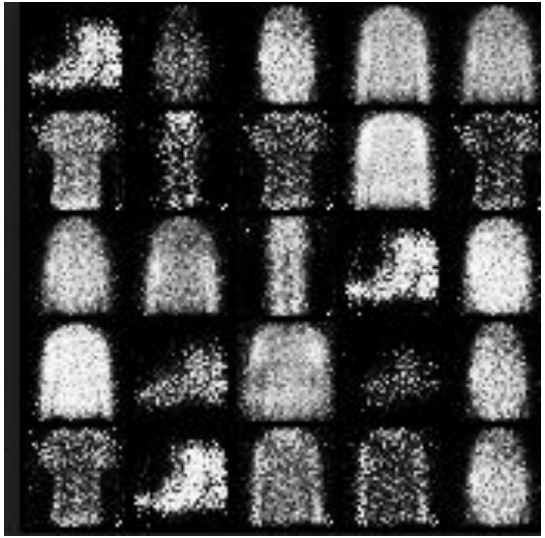
Epoch 500

Results for DCGAN : CELEBA

Epoch 0

Epoch 10

Results for CGAN : Fashion Mnist



Epoch 0

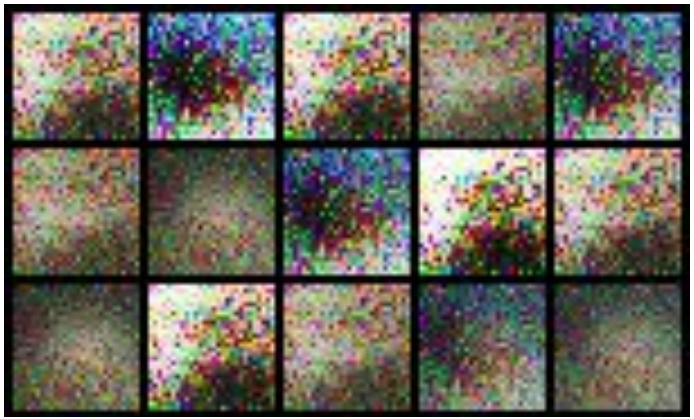
Epoch 200

Results for DCGAN : Fashion Mnist

Epoch 0

Epoch 10

Results for GAN: CIFAR10



Epoch 0

Epoch 2500

Contributions

- Mansi Khamkar (2020201026):
  - Defined the network architecture and algorithm of Gan and conditional Gan
  - Trained GAN on MNIST, DC-GAN on EMNIST and Conditional GAN on EMNIST
- Mahak Modani (2020201068):
  - Trained GAN on EMNIST , DC-GAN on Fashion-MNIST and Conditional GAN on Fashion-MNIST
  - Defined utility function like data loading, creating and loading checkpoints
- Rani Tresa (2020202019):
  - Defined the network architecture and algorithm of Deep Convolutional Gan
  - Trained GAN on CIFAR-10 and Celeb-A and DC-GAN on MNIST and Celeb-A

Conclusion

- **Pros of GANs:**
  - GANs provide better representation learning
  - Only backprop is used to obtain gradients, no inference is needed during learning
  - A wide variety of functions can be incorporated into the model

- **Cons of GANs:**
  - There is no explicit representation of Pg(x) (generator's distribution over data x)
  - Both networks should be synchronized well during training
  - Generator could learn to generate only a small subset of dataset to fool discriminator

THANKYOU!