EE 046746 - Technion - Computer Vision

Tal Daniel

Tutorial 14 - Generative Adversarial Networks (GANs)

- Image Source

Agenda

- What are Generative Adversarial Networks??)
  - Discriminative Vs. Generative
  - Adversarial Training
  - A Game Theory Perspective - Nash Equilibrium
- GANs Training Steps
  - Formulation
  - Algorithm
- 2D Demo
- GANs (Serious) Problems-Problems)

- Vanilla-GAN on MNIST with PyTorch
- Deep Convolutional GANs (DCGANs))
- The Latent Space
- Conditional GANs
- GANs Today
- Tips for Training GANs
- Applications
  - Image-to-Image Translation (Pix2Pix))
  - CycleGAN
  - Realistic Neural Talking Head Models

In [1]:
```
# imports for the tutorial
import time
import numpy as np
import matplotlib.pyplot as plt

# pytorch
import torch
import torch.nn.functional as F
from torchvision import datasets
from torchvision import transforms
import torch.nn as nn
from torch.utils.data import DataLoader

if torch.cuda.is_available():
    torch.backends.cudnn.deterministic = True
```

What Are Generative Adversarial Networks (GANs)?

- **Generative** - learn a generative model that can generate new data.
- **Adversarial** - trained in an *adversarial* setting (there is some competition during the model's training).
- **Networks** - the model is implemented using deep neural networks.

GANs were first introduced in Generative Adversarial Networks, NIPS 2014, by Goodfellow et al.

Discriminative vs. Generative

- So far, we have only seen *discriminative* models
    - Given an image X, predict a label Y
    - That is, we learn $P(Y \mid X)$
- The problem with discriminative models:
    - During training, labels are required, as it is a *supervised* setting.
    - Can't model $P(X)$, i.e., the probability of seeing a certain image.
    - As a result, can't *sample* from $P(X)$, i.e., **can't generate new images**.
- **Generative** models can overcome these limitations!
    - They can model $P(X)$, implicitly (e.g. GANs) or explicitly (e.g. Variational Autoencoders - VAEs).
    - Given a trained model, can generate new images (or data in general).

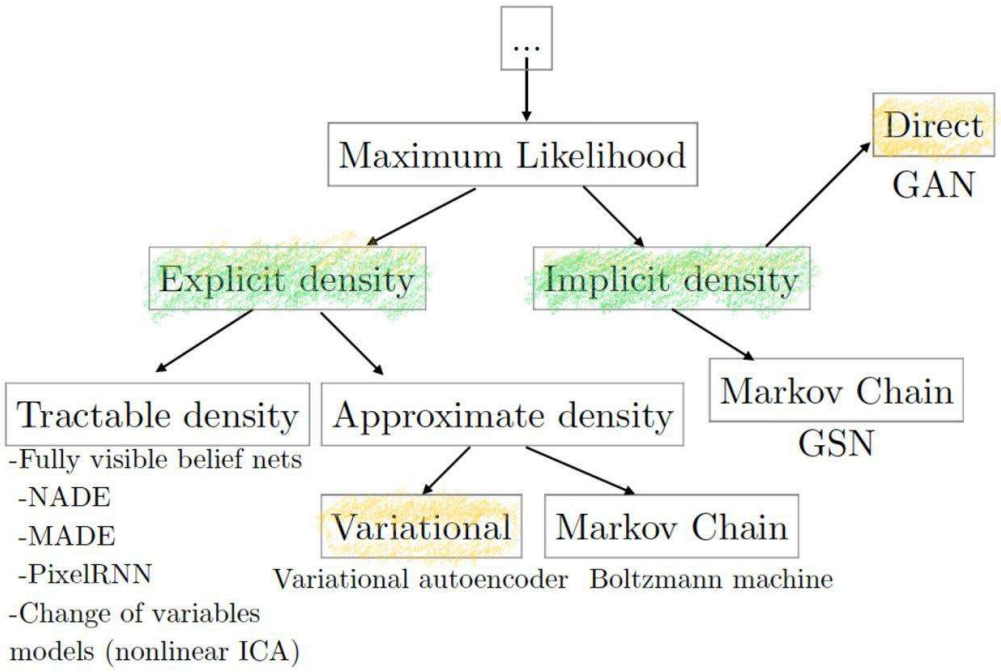
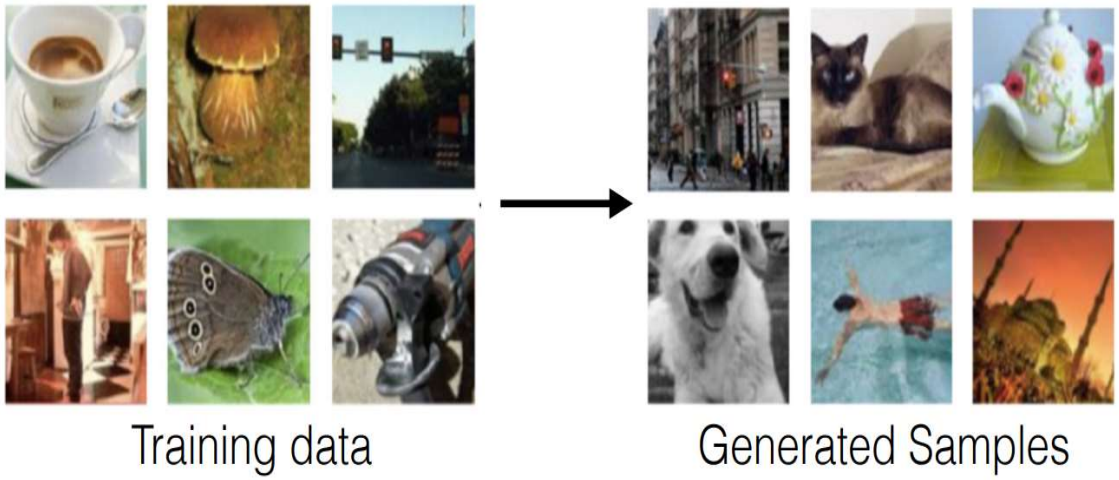


Image Source

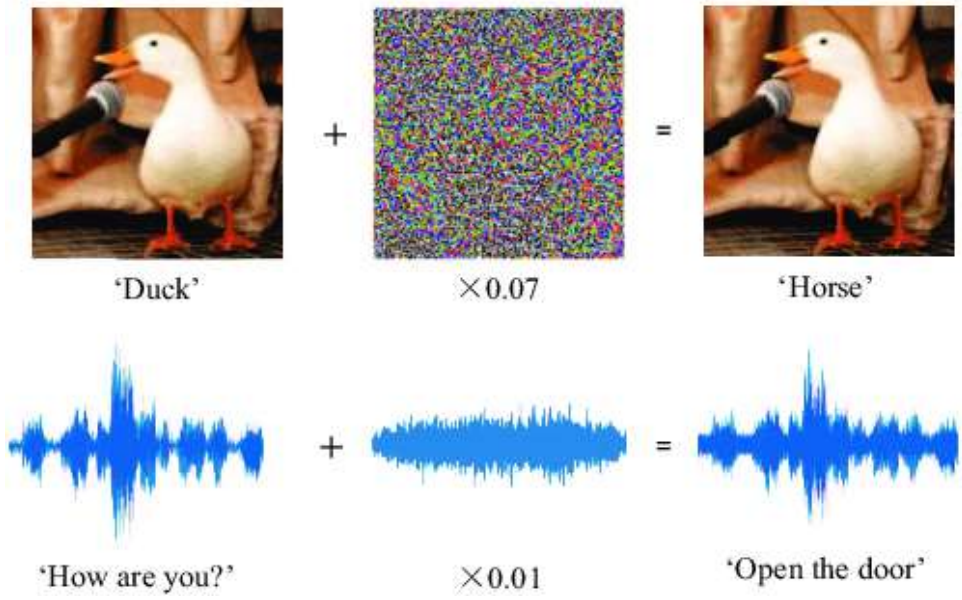
- **Explicit** density estimation: explicitly define and solve for $p_{model}(x)$.
- **Implicit** density estimation: learn a model that can sample from $p_{model}(x)$ without explicitly defining it.

Adversarial Training

- **Goal**: given training data, generate new samples from the same distribution.

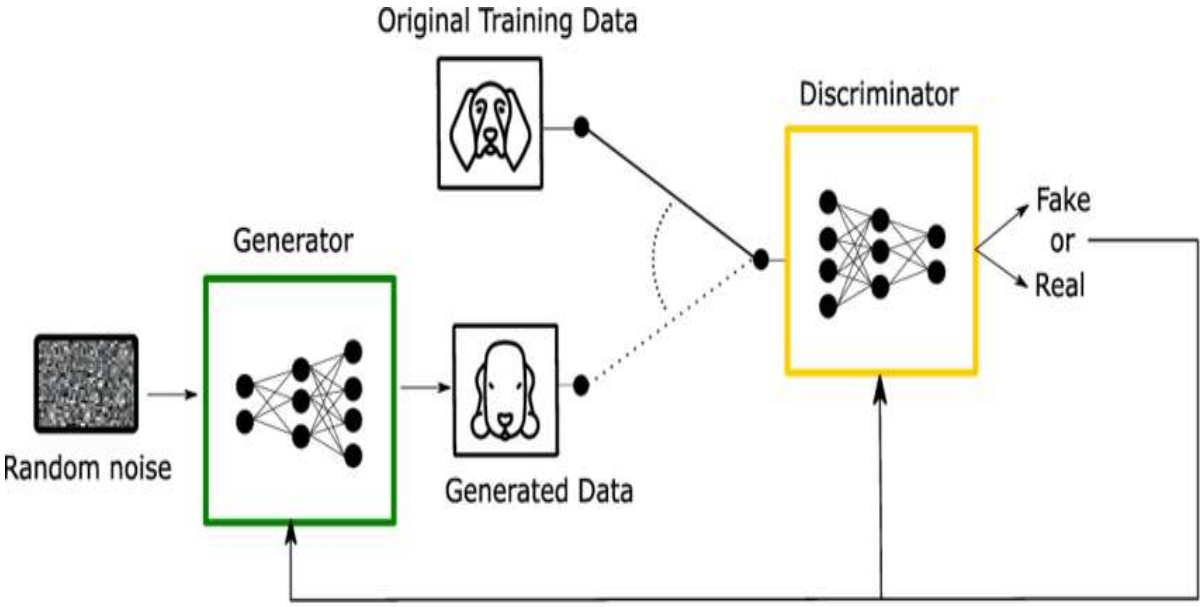


- In **general adversarial setting** (can also be discriminative):
  - We can generate adversarial samples to *fool* a dsicriminative model.
  - Using adversarial samples, we can make models more **robust.**
  - Doing this will require the adversarial samples to be of better quality over time.
    - This will require more effort in generating such quality samples!
  - Repeating this process will result in a better *discriminative* model.



- Image Source

- **GANs** extend this idea to *generative* models:
  - **Generator**: generate fake samples, tries to fool the *Discriminator*.
  - **Discriminator**: tries to dsitinguish between real and fake samples.
  - Train them **against** each other!
  - Repeat this and get a better *Generator* over time.



- Image Source

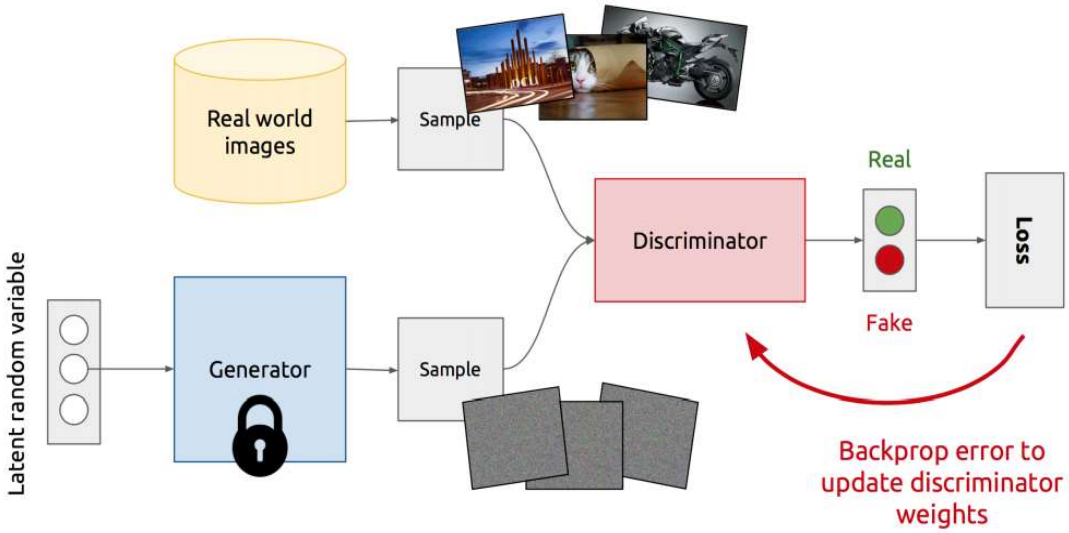
- Image Source

GANs - A Game Theory Persepective - Nash Equilibrium

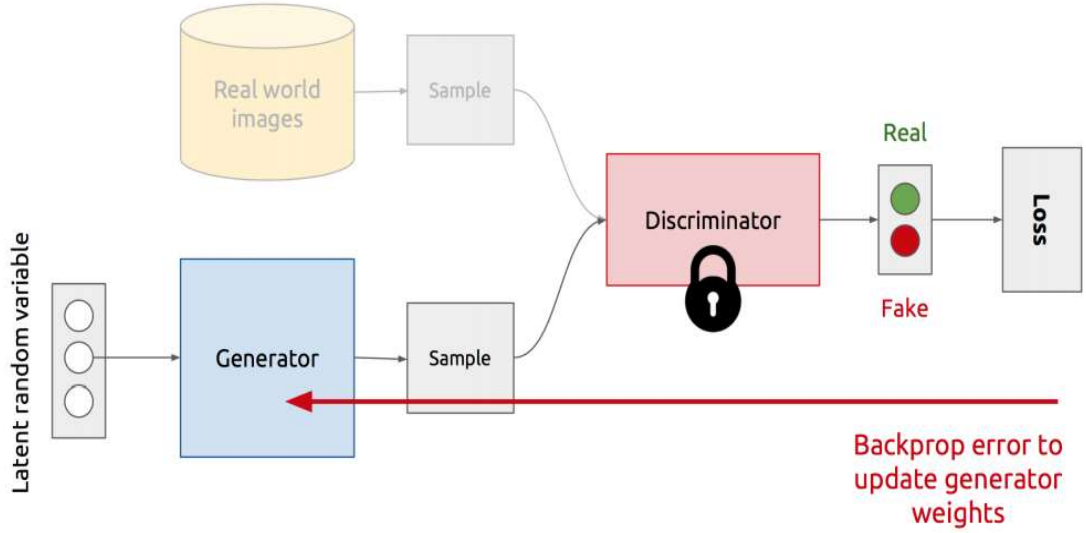
- GAN is based on a **zero-sum** cooperative game (minimax).
  - In short, if one wins the other loses.
- In game theory, the GAN model **converges** when the *discriminator* and the *generator* reach a **Nash equilibrium.**
- **Nash equilibrium** - as both sides want to beat the other, a Nash equilibrium happens when *one player will not change its action regardless of what the opponent may do.*
- **Cost functions may not converge using gradient descent in a minimax game.**

GANs Training Steps

- Training the **Discriminator**
  - **Freeze** the *Generator* and generate fake samples (that is, when backpropagating, don't update the generator weights)



- Training the **Generator**
  - **Freeze** the *Discriminator*, and update the Generator to get a higher score (like the real data) from the Discriminator



Formulation & Algorithm

- For a Discriminator (binary classifier) D, a Generator G and a reward function V, the GAN's objective function:

$$\min_G \max_D V(D, G)$$

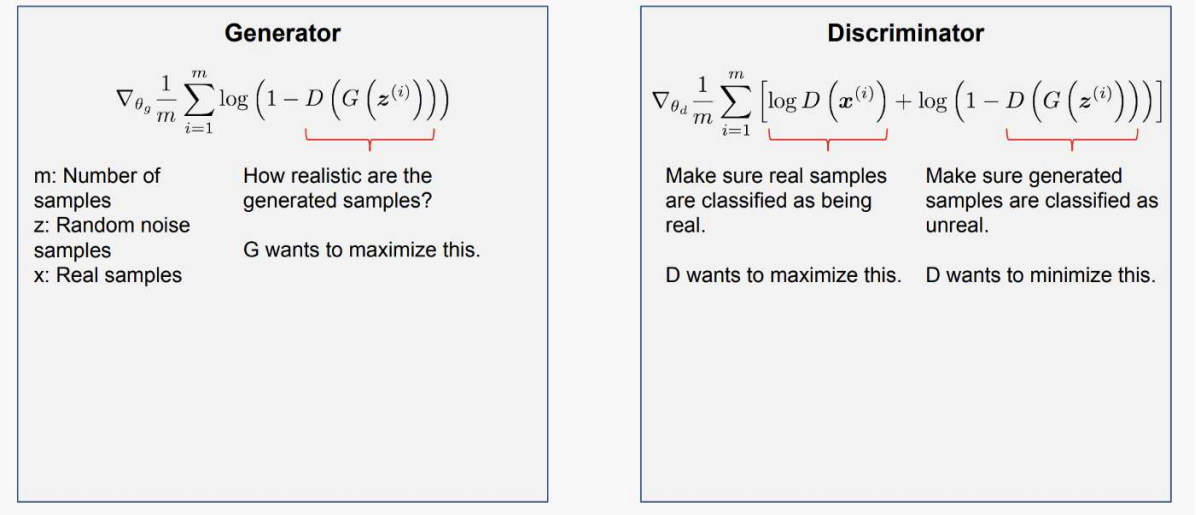
- It is formulated as a **minimax game**, where:
  - The **Discriminator** D is trying to *maximize* its reward $V(D, G)$
  - The **Generator** G is trying to *minimize* the Discriminator's reward (or maximize its loss)
    - Why? Because minimizing the Discriminator's reward means that the Discriminator can not tell the difference between real and fake samples, thus, the Generator is

"winning".

- In our case, the reward function V:

$$V(D,G) = \mathbb{E}_{x\sim p(x)}\left[\log D(x)\right] + \mathbb{E}_{z\sim q(z)}\left[\log(1 - D\left(G(z)\right))\right]$$

  - Recall that for binary classification (real or fake) we use the Binary Cross Entropy (BCE) loss function.
- The **Nash equilibrium** is reached when:
  - $P_{data}(x) = P_{gen}(x), \forall x$
  - $D(x) = \frac{1}{2}$ (completely random classifier).
- Proof of Nash Equilibrium in GANs - However out of the scope of this course, the mathematical proof is very nice and important. It is recommended to go over it if you are interested in working with GANs in the future.



- Image Source

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**

Discriminator updates

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

Generator updates

- 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:

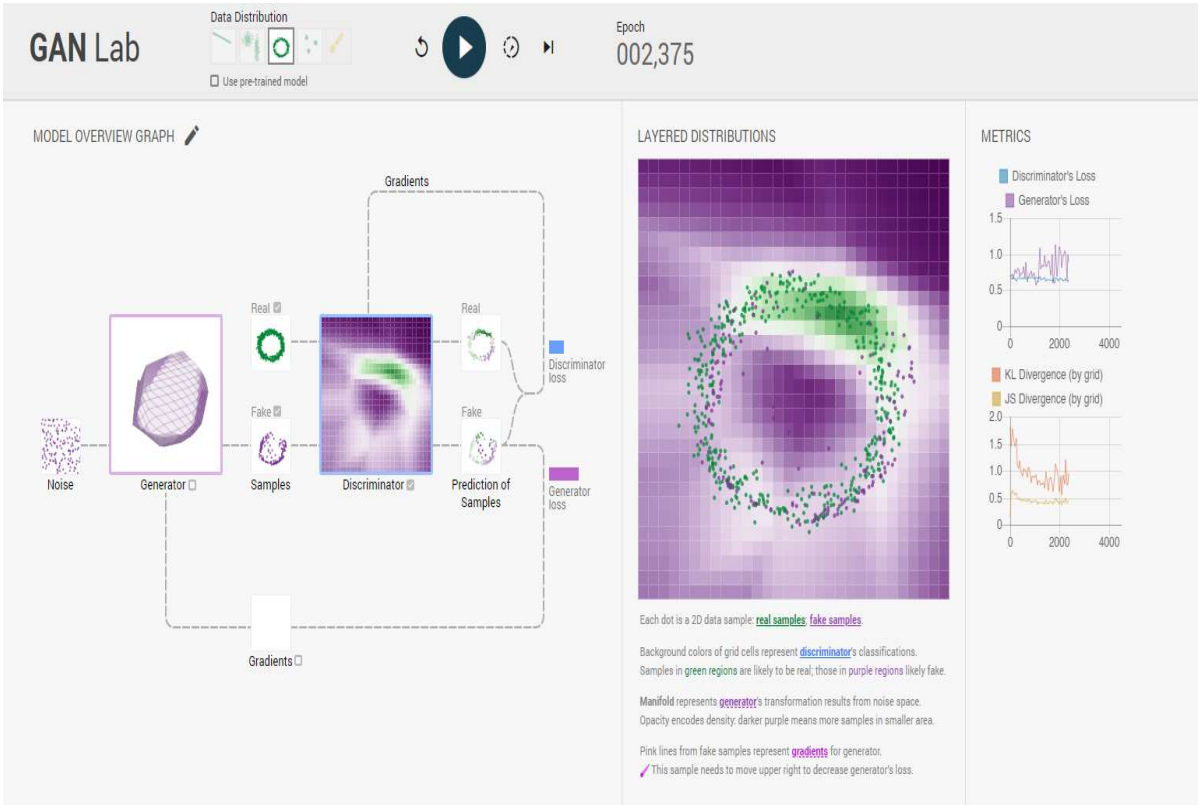
$$\nabla_{\theta_g} \frac{1}{m} \sum_{i=1}^{m} \log\left(1 - D\left(G\left(\boldsymbol{z}^{(i)}\right)\right)\right).$$

end for

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

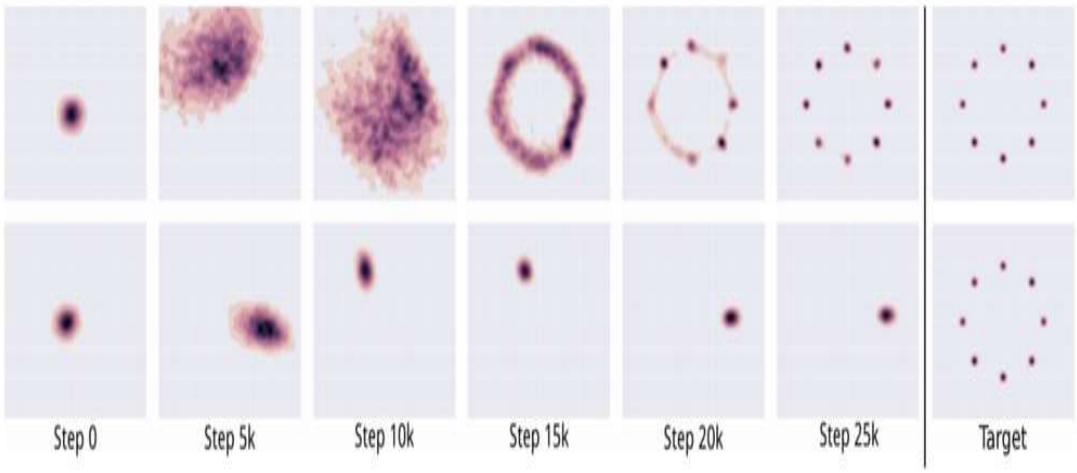
2D Demo

GAN Lab



GANs (Serious) Problems

- **Non-convergence**: the model parameters oscilate, destabilize and (almost) never converge.
- **Mode Collapse**: the *Generator* collapses, which produces limited varieties of samples.
  - For example, on a 2D eight-Gaussians dataset:



  - Image Source

- **Vanisihng/Diminishing Gradient**: the discriminator gets *too good* such that the generator gradient vanishes and learns nothing.
  - Proof: consider the second term in the objective function which is relavent only for the generator.
    - Recall that the output of binary classification is the output of the *sigmoid* function, σ.
    - $\nabla_{\theta_G} V(D, G) = \nabla_{\theta_G} \mathbb{E}_{z \sim q(z)} \left[\log(1 - D\left(G(z)\right))\right]$
    - $\nabla_a \log(1 - \sigma(a)) = \frac{-\nabla_a \sigma(a)}{1 - \sigma(a)} = \frac{-\sigma(a)(1-\sigma(a))}{1-\sigma(a)} = -\sigma(a) = -D\left(G(z)\right)$
    - So if D is confident (that the sample is fake), the gradient goes to 0, i.e. $D\left(G(z)\right) \to 0$
  - Possible remedy: replace the problematic term with

$$\mathbb{E}_{z \sim q(z)} \left[\log(1 - D\left(G(z)\right))\right] \to -\mathbb{E}_{z \sim q(z)} \left[\log D\left(G(z)\right)\right]$$

- **GANs are highly sensitive to hyper-parameters!**
  - Even the slightest change in hyper-parameters may lead to any of the above, e.g. even changing the learning rate from 0.0002 to 0.0001 may lead to instability.

Vanilla-GAN on MNIST with PyTorch

- Based on example by Sebastian Raschka

---------- CODE TIME ----------

In [2]:

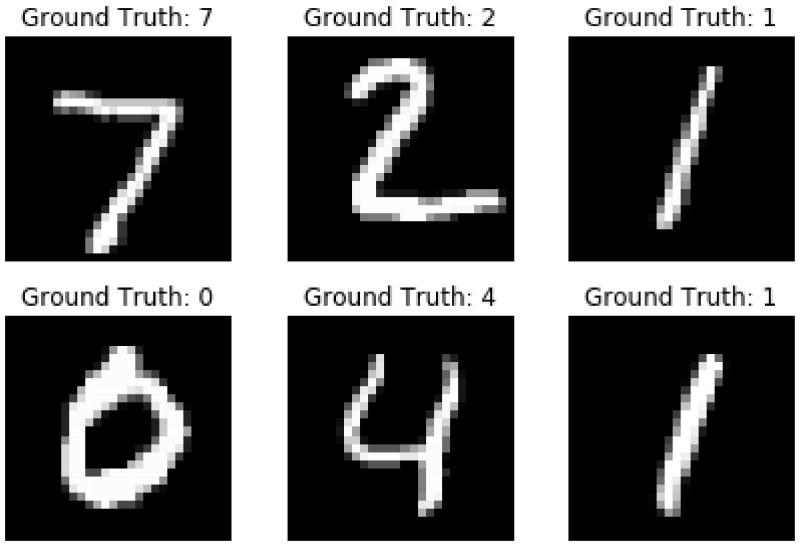
```
##########################
### SETTINGS
##########################

# Device
device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")

# Hyperparameters
# Remember that GANs are highly sensitive to hyper-parameters
random_seed = 123
generator_learning_rate = 0.001
discriminator_learning_rate = 0.001
NUM_EPOCHS = 100
BATCH_SIZE = 128
LATENT_DIM = 100  # latent vectors dimension [z]
IMG_SHAPE = (1, 28, 28)  # MNIST has 1 color channel, each image 28x8 pixels
IMG_SIZE = 1
for x in IMG_SHAPE:
    IMG_SIZE *= x
```

In [3]:

```
##########################
### MNIST DATASET
##########################

# Note transforms.ToTensor() scales input images
# to 0-1 range
train_dataset = datasets.MNIST(root='./datasets',
                               train=True,
                               transform=transforms.ToTensor(),
                               download=True)

test_dataset = datasets.MNIST(root='./datasets',
                              train=False,
                              transform=transforms.ToTensor())


train_loader = DataLoader(dataset=train_dataset,
                          batch_size=BATCH_SIZE,
                          shuffle=True)

test_loader = DataLoader(dataset=test_dataset,
                         batch_size=BATCH_SIZE,
                         shuffle=False)

# Checking the dataset
for images, labels in train_loader:
    print('Image batch dimensions:', images.shape)
    print('Image label dimensions:', labels.shape)
    break

# let's see some digits
examples = enumerate(test_loader)
batch_idx, (example_data, example_targets) = next(examples)
print("shape: \n", example_data.shape)
fig = plt.figure()
```

```
for i in range(6):
    ax = fig.add_subplot(2,3,i+1)
    ax.imshow(example_data[i][0], cmap='gray', interpolation='none')
    ax.set_title("Ground Truth: {}".format(example_targets[i]))
    ax.set_axis_off()
plt.tight_layout()
```

```
Image batch dimensions: torch.Size([128, 1, 28, 28])
Image label dimensions: torch.Size([128])
shape:
 torch.Size([128, 1, 28, 28])
```



In [4]:

```
##########################
### MODEL
##########################

class GAN(torch.nn.Module):

    def __init__(self):
        super(GAN, self).__init__()

        # generator: z [vector] -> image [matrix]
        self.generator = nn.Sequential(
            nn.Linear(LATENT_DIM, 128),
            nn.LeakyReLU(inplace=True),
            nn.Dropout(p=0.5),
            nn.Linear(128, IMG_SIZE),
            nn.Tanh()
        )

        # discriminator: image [matrix] -> label (0-fake, 1-real)
        self.discriminator = nn.Sequential(
            nn.Linear(IMG_SIZE, 128),
            nn.LeakyReLU(inplace=True),
            nn.Dropout(p=0.5),
            nn.Linear(128, 1),
            nn.Sigmoid()
        )


    def generator_forward(self, z):
        img = self.generator(z)
        return img

    def discriminator_forward(self, img):
```

```
        pred = model.discriminator(img)
```

In [5]:

```
# constant the seed
torch.manual_seed(random_seed)

# build the model, send it ti the device
model = GAN().to(device)

# optimizers: we have one for the generator and one for the discriminator
# that way, we can update only one of the modules, while the other one is "frozen"
optim_gener = torch.optim.Adam(model.generator.parameters(), lr=generator_learning_r
optim_discr = torch.optim.Adam(model.discriminator.parameters(), lr=discriminator_le
```

In [6]:

```
##########################
### Training
##########################

start_time = time.time()

discr_costs = []
gener_costs = []
for epoch in range(NUM_EPOCHS):
    model = model.train()
    for batch_idx, (features, targets) in enumerate(train_loader):

        features = (features - 0.5) * 2.0  # normalize between [-1, 1]
        features = features.view(-1, IMG_SIZE).to(device)
        targets = targets.to(device)

        # generate fake and real labels
        valid = torch.ones(targets.size(0)).float().to(device)
        fake = torch.zeros(targets.size(0)).float().to(device)

        ### FORWARD PASS AND BACKPROPAGATION

        # -------------------------
        # Train Generator
        # -------------------------

        # Make new images
        z = torch.zeros((targets.size(0), LATENT_DIM)).uniform_(-1.0, 1.0).to(device
        generated_features = model.generator_forward(z)

        # Loss for fooling the discriminator
        discr_pred = model.discriminator_forward(generated_features)

        # here we use the `valid` labels because we want the discriminator to "think
        # the generated samples are real
        gener_loss = F.binary_cross_entropy(discr_pred, valid)

        optim_gener.zero_grad()
        gener_loss.backward()
        optim_gener.step()

        # -------------------------
        # Train Discriminator
        # -------------------------

        discr_pred_real = model.discriminator_forward(features.view(-1, IMG_SIZE))
        real_loss = F.binary_cross_entropy(discr_pred_real, valid)

        # here we use the `fake` labels when training the discriminator
```

```
        discr_pred_fake = model.discriminator_forward(generated_features.detach())
        fake_loss = F.binary_cross_entropy(discr_pred_fake, fake)

        discr_loss = 0.5 * (real_loss + fake_loss)

        optim_discr.zero_grad()
        discr_loss.backward()
        optim_discr.step()

        discr_costs.append(discr_loss)
        gener_costs.append(gener_loss)


        ### LOGGING
        if not batch_idx % 100:
            print ('Epoch: %03d/%03d | Batch %03d/%03d | Gen/Dis Loss: %.4f/%.4f'
                   %(epoch+1, NUM_EPOCHS, batch_idx,
                     len(train_loader), gener_loss, discr_loss))

    print('Time elapsed: %.2f min' % ((time.time() - start_time)/60))

print('Total Training Time: %.2f min' % ((time.time() - start_time)/60))
```

```
Epoch: 001/100 | Batch 000/469 | Gen/Dis Loss: 0.6838/0.7138
Epoch: 001/100 | Batch 100/469 | Gen/Dis Loss: 4.3840/0.0432
Epoch: 001/100 | Batch 200/469 | Gen/Dis Loss: 1.9350/0.0932
Epoch: 001/100 | Batch 300/469 | Gen/Dis Loss: 1.2844/0.2061
Epoch: 001/100 | Batch 400/469 | Gen/Dis Loss: 1.9026/0.1210
Time elapsed: 0.21 min
Epoch: 002/100 | Batch 000/469 | Gen/Dis Loss: 3.0769/0.1038
Epoch: 002/100 | Batch 100/469 | Gen/Dis Loss: 2.2545/0.2297
Epoch: 002/100 | Batch 200/469 | Gen/Dis Loss: 1.6195/0.2978
Epoch: 002/100 | Batch 300/469 | Gen/Dis Loss: 1.3532/0.3820
Epoch: 002/100 | Batch 400/469 | Gen/Dis Loss: 0.8324/0.4919
Time elapsed: 0.42 min
Epoch: 003/100 | Batch 000/469 | Gen/Dis Loss: 0.9838/0.3945
Epoch: 003/100 | Batch 100/469 | Gen/Dis Loss: 1.1632/0.3346
Epoch: 003/100 | Batch 200/469 | Gen/Dis Loss: 1.5950/0.2493
Epoch: 003/100 | Batch 300/469 | Gen/Dis Loss: 0.9661/0.4662
Epoch: 003/100 | Batch 400/469 | Gen/Dis Loss: 1.4429/0.3401
Time elapsed: 0.60 min
Epoch: 004/100 | Batch 000/469 | Gen/Dis Loss: 1.0569/0.4390
Epoch: 004/100 | Batch 100/469 | Gen/Dis Loss: 1.0999/0.4675
Epoch: 004/100 | Batch 200/469 | Gen/Dis Loss: 1.0270/0.4134
Epoch: 004/100 | Batch 300/469 | Gen/Dis Loss: 1.5850/0.3808
Epoch: 004/100 | Batch 400/469 | Gen/Dis Loss: 0.8862/0.5399
Time elapsed: 0.82 min
Epoch: 005/100 | Batch 000/469 | Gen/Dis Loss: 1.3939/0.5117
Epoch: 005/100 | Batch 100/469 | Gen/Dis Loss: 0.7912/0.5171
Epoch: 005/100 | Batch 200/469 | Gen/Dis Loss: 1.2323/0.4814
Epoch: 005/100 | Batch 300/469 | Gen/Dis Loss: 1.3585/0.4899
Epoch: 005/100 | Batch 400/469 | Gen/Dis Loss: 1.3954/0.4305
Time elapsed: 1.03 min
Epoch: 006/100 | Batch 000/469 | Gen/Dis Loss: 0.9018/0.4902
Epoch: 006/100 | Batch 100/469 | Gen/Dis Loss: 1.2610/0.4479
Epoch: 006/100 | Batch 200/469 | Gen/Dis Loss: 1.3451/0.4238
Epoch: 006/100 | Batch 300/469 | Gen/Dis Loss: 1.0149/0.5674
Epoch: 006/100 | Batch 400/469 | Gen/Dis Loss: 0.8990/0.5092
Time elapsed: 1.24 min
Epoch: 007/100 | Batch 000/469 | Gen/Dis Loss: 1.0070/0.5217
Epoch: 007/100 | Batch 100/469 | Gen/Dis Loss: 0.9744/0.5619
Epoch: 007/100 | Batch 200/469 | Gen/Dis Loss: 0.9587/0.4871
Epoch: 007/100 | Batch 300/469 | Gen/Dis Loss: 1.2298/0.4278
Epoch: 007/100 | Batch 400/469 | Gen/Dis Loss: 1.2283/0.4623
Time elapsed: 1.42 min
Epoch: 008/100 | Batch 000/469 | Gen/Dis Loss: 2.6874/0.4093
Epoch: 008/100 | Batch 100/469 | Gen/Dis Loss: 1.2914/0.4006
```

```
Epoch: 008/100 | Batch 200/469 | Gen/Dis Loss: 1.0255/0.4877
Epoch: 008/100 | Batch 300/469 | Gen/Dis Loss: 1.0429/0.5381
Epoch: 008/100 | Batch 400/469 | Gen/Dis Loss: 0.9911/0.5373
Time elapsed: 1.62 min
Epoch: 009/100 | Batch 000/469 | Gen/Dis Loss: 1.3408/0.4490
Epoch: 009/100 | Batch 100/469 | Gen/Dis Loss: 0.8755/0.4988
Epoch: 009/100 | Batch 200/469 | Gen/Dis Loss: 0.9902/0.5171
Epoch: 009/100 | Batch 300/469 | Gen/Dis Loss: 1.1354/0.5539
Epoch: 009/100 | Batch 400/469 | Gen/Dis Loss: 1.2235/0.5087
Time elapsed: 1.82 min
Epoch: 010/100 | Batch 000/469 | Gen/Dis Loss: 1.0621/0.5484
Epoch: 010/100 | Batch 100/469 | Gen/Dis Loss: 1.0421/0.5424
Epoch: 010/100 | Batch 200/469 | Gen/Dis Loss: 1.1361/0.4913
Epoch: 010/100 | Batch 300/469 | Gen/Dis Loss: 1.3669/0.4614
Epoch: 010/100 | Batch 400/469 | Gen/Dis Loss: 1.7223/0.3940
Time elapsed: 2.01 min
Epoch: 011/100 | Batch 000/469 | Gen/Dis Loss: 1.8185/0.4393
Epoch: 011/100 | Batch 100/469 | Gen/Dis Loss: 1.4372/0.4684
Epoch: 011/100 | Batch 200/469 | Gen/Dis Loss: 1.2391/0.4745
Epoch: 011/100 | Batch 300/469 | Gen/Dis Loss: 1.8360/0.5076
Epoch: 011/100 | Batch 400/469 | Gen/Dis Loss: 1.0983/0.4962
Time elapsed: 2.22 min
Epoch: 012/100 | Batch 000/469 | Gen/Dis Loss: 1.0624/0.6154
Epoch: 012/100 | Batch 100/469 | Gen/Dis Loss: 1.4581/0.4141
Epoch: 012/100 | Batch 200/469 | Gen/Dis Loss: 1.5077/0.5387
Epoch: 012/100 | Batch 300/469 | Gen/Dis Loss: 0.9242/0.5931
Epoch: 012/100 | Batch 400/469 | Gen/Dis Loss: 1.6200/0.5159
Time elapsed: 2.42 min
Epoch: 013/100 | Batch 000/469 | Gen/Dis Loss: 1.1211/0.5278
Epoch: 013/100 | Batch 100/469 | Gen/Dis Loss: 1.3972/0.5559
Epoch: 013/100 | Batch 200/469 | Gen/Dis Loss: 1.0772/0.5088
Epoch: 013/100 | Batch 300/469 | Gen/Dis Loss: 1.0421/0.5161
Epoch: 013/100 | Batch 400/469 | Gen/Dis Loss: 1.0460/0.5283
Time elapsed: 2.63 min
Epoch: 014/100 | Batch 000/469 | Gen/Dis Loss: 1.0134/0.5324
Epoch: 014/100 | Batch 100/469 | Gen/Dis Loss: 1.0253/0.5432
Epoch: 014/100 | Batch 200/469 | Gen/Dis Loss: 1.0420/0.5056
Epoch: 014/100 | Batch 300/469 | Gen/Dis Loss: 0.9032/0.5923
Epoch: 014/100 | Batch 400/469 | Gen/Dis Loss: 1.5168/0.5644
Time elapsed: 2.83 min
Epoch: 015/100 | Batch 000/469 | Gen/Dis Loss: 0.9793/0.5231
Epoch: 015/100 | Batch 100/469 | Gen/Dis Loss: 1.1882/0.5807
Epoch: 015/100 | Batch 200/469 | Gen/Dis Loss: 1.5222/0.4828
Epoch: 015/100 | Batch 300/469 | Gen/Dis Loss: 1.0066/0.5354
Epoch: 015/100 | Batch 400/469 | Gen/Dis Loss: 1.1868/0.5143
Time elapsed: 3.03 min
Epoch: 016/100 | Batch 000/469 | Gen/Dis Loss: 1.2892/0.4966
Epoch: 016/100 | Batch 100/469 | Gen/Dis Loss: 1.2736/0.4939
Epoch: 016/100 | Batch 200/469 | Gen/Dis Loss: 1.2180/0.5542
Epoch: 016/100 | Batch 300/469 | Gen/Dis Loss: 1.1227/0.5524
Epoch: 016/100 | Batch 400/469 | Gen/Dis Loss: 1.3174/0.4623
Time elapsed: 3.23 min
Epoch: 017/100 | Batch 000/469 | Gen/Dis Loss: 1.1275/0.5069
Epoch: 017/100 | Batch 100/469 | Gen/Dis Loss: 1.2431/0.5452
Epoch: 017/100 | Batch 200/469 | Gen/Dis Loss: 1.2845/0.4954
Epoch: 017/100 | Batch 300/469 | Gen/Dis Loss: 1.1289/0.5558
Epoch: 017/100 | Batch 400/469 | Gen/Dis Loss: 1.8049/0.5299
Time elapsed: 3.46 min
Epoch: 018/100 | Batch 000/469 | Gen/Dis Loss: 1.0419/0.5352
Epoch: 018/100 | Batch 100/469 | Gen/Dis Loss: 0.9424/0.5597
Epoch: 018/100 | Batch 200/469 | Gen/Dis Loss: 1.1577/0.5832
Epoch: 018/100 | Batch 300/469 | Gen/Dis Loss: 0.9085/0.5702
Epoch: 018/100 | Batch 400/469 | Gen/Dis Loss: 1.0246/0.5918
Time elapsed: 3.67 min
Epoch: 019/100 | Batch 000/469 | Gen/Dis Loss: 1.2700/0.4862
Epoch: 019/100 | Batch 100/469 | Gen/Dis Loss: 1.4914/0.5635
Epoch: 019/100 | Batch 200/469 | Gen/Dis Loss: 0.9100/0.5929
Epoch: 019/100 | Batch 300/469 | Gen/Dis Loss: 1.7084/0.4677
Epoch: 019/100 | Batch 400/469 | Gen/Dis Loss: 0.9109/0.5568
```

```
Time elapsed: 3.88 min
Epoch: 020/100 | Batch 000/469 | Gen/Dis Loss: 1.0908/0.5502
Epoch: 020/100 | Batch 100/469 | Gen/Dis Loss: 1.6147/0.4600
Epoch: 020/100 | Batch 200/469 | Gen/Dis Loss: 1.1370/0.5883
Epoch: 020/100 | Batch 300/469 | Gen/Dis Loss: 1.1306/0.5267
Epoch: 020/100 | Batch 400/469 | Gen/Dis Loss: 1.4203/0.4824
Time elapsed: 4.09 min
Epoch: 021/100 | Batch 000/469 | Gen/Dis Loss: 1.1468/0.5482
Epoch: 021/100 | Batch 100/469 | Gen/Dis Loss: 0.9530/0.6101
Epoch: 021/100 | Batch 200/469 | Gen/Dis Loss: 1.1310/0.5294
Epoch: 021/100 | Batch 300/469 | Gen/Dis Loss: 0.9759/0.6017
Epoch: 021/100 | Batch 400/469 | Gen/Dis Loss: 1.0787/0.5905
Time elapsed: 4.31 min
Epoch: 022/100 | Batch 000/469 | Gen/Dis Loss: 1.0696/0.5426
Epoch: 022/100 | Batch 100/469 | Gen/Dis Loss: 0.9882/0.5554
Epoch: 022/100 | Batch 200/469 | Gen/Dis Loss: 0.9958/0.5572
Epoch: 022/100 | Batch 300/469 | Gen/Dis Loss: 1.1832/0.5385
Epoch: 022/100 | Batch 400/469 | Gen/Dis Loss: 1.2904/0.5589
Time elapsed: 4.55 min
Epoch: 023/100 | Batch 000/469 | Gen/Dis Loss: 0.9572/0.5809
Epoch: 023/100 | Batch 100/469 | Gen/Dis Loss: 0.9179/0.5986
Epoch: 023/100 | Batch 200/469 | Gen/Dis Loss: 0.9488/0.5900
Epoch: 023/100 | Batch 300/469 | Gen/Dis Loss: 1.0407/0.5560
Epoch: 023/100 | Batch 400/469 | Gen/Dis Loss: 1.3512/0.5442
Time elapsed: 4.79 min
Epoch: 024/100 | Batch 000/469 | Gen/Dis Loss: 1.1840/0.6484
Epoch: 024/100 | Batch 100/469 | Gen/Dis Loss: 1.1996/0.6011
Epoch: 024/100 | Batch 200/469 | Gen/Dis Loss: 1.1017/0.5112
Epoch: 024/100 | Batch 300/469 | Gen/Dis Loss: 1.0241/0.5999
Epoch: 024/100 | Batch 400/469 | Gen/Dis Loss: 1.1327/0.5325
Time elapsed: 5.04 min
Epoch: 025/100 | Batch 000/469 | Gen/Dis Loss: 1.0342/0.5672
Epoch: 025/100 | Batch 100/469 | Gen/Dis Loss: 0.8939/0.5617
Epoch: 025/100 | Batch 200/469 | Gen/Dis Loss: 0.9729/0.6117
Epoch: 025/100 | Batch 300/469 | Gen/Dis Loss: 0.9589/0.6447
Epoch: 025/100 | Batch 400/469 | Gen/Dis Loss: 1.4610/0.5302
Time elapsed: 5.25 min
Epoch: 026/100 | Batch 000/469 | Gen/Dis Loss: 1.0038/0.5160
Epoch: 026/100 | Batch 100/469 | Gen/Dis Loss: 1.1733/0.5434
Epoch: 026/100 | Batch 200/469 | Gen/Dis Loss: 1.3599/0.5809
Epoch: 026/100 | Batch 300/469 | Gen/Dis Loss: 0.8480/0.6155
Epoch: 026/100 | Batch 400/469 | Gen/Dis Loss: 1.4864/0.4897
Time elapsed: 5.48 min
Epoch: 027/100 | Batch 000/469 | Gen/Dis Loss: 0.9030/0.5860
Epoch: 027/100 | Batch 100/469 | Gen/Dis Loss: 1.1043/0.5277
Epoch: 027/100 | Batch 200/469 | Gen/Dis Loss: 0.9574/0.5939
Epoch: 027/100 | Batch 300/469 | Gen/Dis Loss: 1.0379/0.5460
Epoch: 027/100 | Batch 400/469 | Gen/Dis Loss: 0.8172/0.5898
Time elapsed: 5.69 min
Epoch: 028/100 | Batch 000/469 | Gen/Dis Loss: 1.1094/0.5256
Epoch: 028/100 | Batch 100/469 | Gen/Dis Loss: 1.0559/0.5904
Epoch: 028/100 | Batch 200/469 | Gen/Dis Loss: 0.9051/0.6414
Epoch: 028/100 | Batch 300/469 | Gen/Dis Loss: 1.1645/0.5475
Epoch: 028/100 | Batch 400/469 | Gen/Dis Loss: 1.0180/0.5622
Time elapsed: 5.91 min
Epoch: 029/100 | Batch 000/469 | Gen/Dis Loss: 1.0982/0.5997
Epoch: 029/100 | Batch 100/469 | Gen/Dis Loss: 0.9629/0.5879
Epoch: 029/100 | Batch 200/469 | Gen/Dis Loss: 0.9076/0.5769
Epoch: 029/100 | Batch 300/469 | Gen/Dis Loss: 1.2277/0.5598
Epoch: 029/100 | Batch 400/469 | Gen/Dis Loss: 1.1624/0.5365
Time elapsed: 6.11 min
Epoch: 030/100 | Batch 000/469 | Gen/Dis Loss: 1.2315/0.6021
Epoch: 030/100 | Batch 100/469 | Gen/Dis Loss: 0.9686/0.6194
Epoch: 030/100 | Batch 200/469 | Gen/Dis Loss: 1.0561/0.5846
Epoch: 030/100 | Batch 300/469 | Gen/Dis Loss: 1.0981/0.5803
Epoch: 030/100 | Batch 400/469 | Gen/Dis Loss: 0.8582/0.5956
Time elapsed: 6.34 min
Epoch: 031/100 | Batch 000/469 | Gen/Dis Loss: 1.3425/0.5229
Epoch: 031/100 | Batch 100/469 | Gen/Dis Loss: 0.9575/0.5593
```

```
Epoch: 031/100 | Batch 200/469 | Gen/Dis Loss: 0.9260/0.5978
Epoch: 031/100 | Batch 300/469 | Gen/Dis Loss: 0.9323/0.6066
Epoch: 031/100 | Batch 400/469 | Gen/Dis Loss: 1.1442/0.5835
Time elapsed: 6.54 min
Epoch: 032/100 | Batch 000/469 | Gen/Dis Loss: 1.3801/0.5657
Epoch: 032/100 | Batch 100/469 | Gen/Dis Loss: 0.8169/0.6057
Epoch: 032/100 | Batch 200/469 | Gen/Dis Loss: 0.9113/0.6228
Epoch: 032/100 | Batch 300/469 | Gen/Dis Loss: 0.9893/0.5602
Epoch: 032/100 | Batch 400/469 | Gen/Dis Loss: 1.0111/0.5802
Time elapsed: 6.77 min
Epoch: 033/100 | Batch 000/469 | Gen/Dis Loss: 0.8720/0.5803
Epoch: 033/100 | Batch 100/469 | Gen/Dis Loss: 1.0094/0.5656
Epoch: 033/100 | Batch 200/469 | Gen/Dis Loss: 0.9539/0.5879
Epoch: 033/100 | Batch 300/469 | Gen/Dis Loss: 0.9599/0.5688
Epoch: 033/100 | Batch 400/469 | Gen/Dis Loss: 1.3850/0.5857
Time elapsed: 7.02 min
Epoch: 034/100 | Batch 000/469 | Gen/Dis Loss: 0.8792/0.5796
Epoch: 034/100 | Batch 100/469 | Gen/Dis Loss: 0.8232/0.6460
Epoch: 034/100 | Batch 200/469 | Gen/Dis Loss: 1.1379/0.5846
Epoch: 034/100 | Batch 300/469 | Gen/Dis Loss: 0.9232/0.6094
Epoch: 034/100 | Batch 400/469 | Gen/Dis Loss: 1.0959/0.6208
Time elapsed: 7.25 min
Epoch: 035/100 | Batch 000/469 | Gen/Dis Loss: 1.1373/0.5709
Epoch: 035/100 | Batch 100/469 | Gen/Dis Loss: 0.8874/0.6248
Epoch: 035/100 | Batch 200/469 | Gen/Dis Loss: 0.9056/0.5765
Epoch: 035/100 | Batch 300/469 | Gen/Dis Loss: 1.1130/0.6055
Epoch: 035/100 | Batch 400/469 | Gen/Dis Loss: 0.9627/0.6587
Time elapsed: 7.49 min
Epoch: 036/100 | Batch 000/469 | Gen/Dis Loss: 0.8178/0.5952
Epoch: 036/100 | Batch 100/469 | Gen/Dis Loss: 0.8416/0.6053
Epoch: 036/100 | Batch 200/469 | Gen/Dis Loss: 0.8820/0.6133
Epoch: 036/100 | Batch 300/469 | Gen/Dis Loss: 0.8580/0.6191
Epoch: 036/100 | Batch 400/469 | Gen/Dis Loss: 1.0142/0.6009
Time elapsed: 7.69 min
Epoch: 037/100 | Batch 000/469 | Gen/Dis Loss: 0.7809/0.6244
Epoch: 037/100 | Batch 100/469 | Gen/Dis Loss: 0.9743/0.5590
Epoch: 037/100 | Batch 200/469 | Gen/Dis Loss: 0.8327/0.5932
Epoch: 037/100 | Batch 300/469 | Gen/Dis Loss: 0.8744/0.6076
Epoch: 037/100 | Batch 400/469 | Gen/Dis Loss: 0.9382/0.6378
Time elapsed: 7.93 min
Epoch: 038/100 | Batch 000/469 | Gen/Dis Loss: 0.8697/0.6495
Epoch: 038/100 | Batch 100/469 | Gen/Dis Loss: 1.0503/0.6030
Epoch: 038/100 | Batch 200/469 | Gen/Dis Loss: 0.8235/0.6585
Epoch: 038/100 | Batch 300/469 | Gen/Dis Loss: 0.9239/0.6315
Epoch: 038/100 | Batch 400/469 | Gen/Dis Loss: 0.7960/0.6314
Time elapsed: 8.16 min
Epoch: 039/100 | Batch 000/469 | Gen/Dis Loss: 0.8326/0.6530
Epoch: 039/100 | Batch 100/469 | Gen/Dis Loss: 0.7643/0.6429
Epoch: 039/100 | Batch 200/469 | Gen/Dis Loss: 0.7270/0.6328
Epoch: 039/100 | Batch 300/469 | Gen/Dis Loss: 1.3712/0.5792
Epoch: 039/100 | Batch 400/469 | Gen/Dis Loss: 0.8928/0.6070
Time elapsed: 8.42 min
Epoch: 040/100 | Batch 000/469 | Gen/Dis Loss: 0.9620/0.5787
Epoch: 040/100 | Batch 100/469 | Gen/Dis Loss: 1.0916/0.5769
Epoch: 040/100 | Batch 200/469 | Gen/Dis Loss: 1.1232/0.6285
Epoch: 040/100 | Batch 300/469 | Gen/Dis Loss: 0.8525/0.6524
Epoch: 040/100 | Batch 400/469 | Gen/Dis Loss: 0.9783/0.6252
Time elapsed: 8.64 min
Epoch: 041/100 | Batch 000/469 | Gen/Dis Loss: 0.9488/0.5574
Epoch: 041/100 | Batch 100/469 | Gen/Dis Loss: 0.9242/0.6050
Epoch: 041/100 | Batch 200/469 | Gen/Dis Loss: 0.9447/0.6402
Epoch: 041/100 | Batch 300/469 | Gen/Dis Loss: 0.8728/0.6267
Epoch: 041/100 | Batch 400/469 | Gen/Dis Loss: 0.9273/0.5865
Time elapsed: 8.85 min
Epoch: 042/100 | Batch 000/469 | Gen/Dis Loss: 0.7490/0.6812
Epoch: 042/100 | Batch 100/469 | Gen/Dis Loss: 0.8127/0.6084
Epoch: 042/100 | Batch 200/469 | Gen/Dis Loss: 0.8308/0.6367
Epoch: 042/100 | Batch 300/469 | Gen/Dis Loss: 0.7962/0.6566
Epoch: 042/100 | Batch 400/469 | Gen/Dis Loss: 1.4659/0.5730
```

```
Time elapsed: 9.08 min
Epoch: 043/100 | Batch 000/469 | Gen/Dis Loss: 1.1201/0.6311
Epoch: 043/100 | Batch 100/469 | Gen/Dis Loss: 0.8128/0.6460
Epoch: 043/100 | Batch 200/469 | Gen/Dis Loss: 0.8567/0.5978
Epoch: 043/100 | Batch 300/469 | Gen/Dis Loss: 0.7823/0.6573
Epoch: 043/100 | Batch 400/469 | Gen/Dis Loss: 0.7419/0.6539
Time elapsed: 9.33 min
Epoch: 044/100 | Batch 000/469 | Gen/Dis Loss: 0.7437/0.6641
Epoch: 044/100 | Batch 100/469 | Gen/Dis Loss: 1.0232/0.5778
Epoch: 044/100 | Batch 200/469 | Gen/Dis Loss: 0.8897/0.5788
Epoch: 044/100 | Batch 300/469 | Gen/Dis Loss: 1.1323/0.5828
Epoch: 044/100 | Batch 400/469 | Gen/Dis Loss: 0.8746/0.6311
Time elapsed: 9.56 min
Epoch: 045/100 | Batch 000/469 | Gen/Dis Loss: 0.8020/0.6247
Epoch: 045/100 | Batch 100/469 | Gen/Dis Loss: 0.9846/0.5807
Epoch: 045/100 | Batch 200/469 | Gen/Dis Loss: 0.8492/0.6349
Epoch: 045/100 | Batch 300/469 | Gen/Dis Loss: 0.9224/0.6573
Epoch: 045/100 | Batch 400/469 | Gen/Dis Loss: 0.9209/0.5996
Time elapsed: 9.80 min
Epoch: 046/100 | Batch 000/469 | Gen/Dis Loss: 0.8332/0.6598
Epoch: 046/100 | Batch 100/469 | Gen/Dis Loss: 0.8846/0.6273
Epoch: 046/100 | Batch 200/469 | Gen/Dis Loss: 0.8176/0.6147
Epoch: 046/100 | Batch 300/469 | Gen/Dis Loss: 0.8477/0.6145
Epoch: 046/100 | Batch 400/469 | Gen/Dis Loss: 0.9532/0.5765
Time elapsed: 10.03 min
Epoch: 047/100 | Batch 000/469 | Gen/Dis Loss: 0.7871/0.6487
Epoch: 047/100 | Batch 100/469 | Gen/Dis Loss: 0.8932/0.6317
Epoch: 047/100 | Batch 200/469 | Gen/Dis Loss: 0.7984/0.6379
Epoch: 047/100 | Batch 300/469 | Gen/Dis Loss: 0.8198/0.6323
Epoch: 047/100 | Batch 400/469 | Gen/Dis Loss: 1.0208/0.5895
Time elapsed: 10.27 min
Epoch: 048/100 | Batch 000/469 | Gen/Dis Loss: 0.8552/0.6434
Epoch: 048/100 | Batch 100/469 | Gen/Dis Loss: 0.8886/0.6436
Epoch: 048/100 | Batch 200/469 | Gen/Dis Loss: 0.9072/0.6061
Epoch: 048/100 | Batch 300/469 | Gen/Dis Loss: 0.9447/0.6791
Epoch: 048/100 | Batch 400/469 | Gen/Dis Loss: 0.7883/0.6480
Time elapsed: 10.48 min
Epoch: 049/100 | Batch 000/469 | Gen/Dis Loss: 0.8265/0.6202
Epoch: 049/100 | Batch 100/469 | Gen/Dis Loss: 0.8176/0.6464
Epoch: 049/100 | Batch 200/469 | Gen/Dis Loss: 0.7235/0.6404
Epoch: 049/100 | Batch 300/469 | Gen/Dis Loss: 0.8794/0.6152
Epoch: 049/100 | Batch 400/469 | Gen/Dis Loss: 0.7694/0.6427
Time elapsed: 10.74 min
Epoch: 050/100 | Batch 000/469 | Gen/Dis Loss: 0.7295/0.6389
Epoch: 050/100 | Batch 100/469 | Gen/Dis Loss: 0.9543/0.6071
Epoch: 050/100 | Batch 200/469 | Gen/Dis Loss: 0.8134/0.6282
Epoch: 050/100 | Batch 300/469 | Gen/Dis Loss: 0.8284/0.6722
Epoch: 050/100 | Batch 400/469 | Gen/Dis Loss: 0.8386/0.6264
Time elapsed: 10.96 min
Epoch: 051/100 | Batch 000/469 | Gen/Dis Loss: 0.7738/0.6598
Epoch: 051/100 | Batch 100/469 | Gen/Dis Loss: 0.7897/0.6525
Epoch: 051/100 | Batch 200/469 | Gen/Dis Loss: 1.0262/0.6191
Epoch: 051/100 | Batch 300/469 | Gen/Dis Loss: 0.8058/0.6210
Epoch: 051/100 | Batch 400/469 | Gen/Dis Loss: 0.8964/0.6628
Time elapsed: 11.20 min
Epoch: 052/100 | Batch 000/469 | Gen/Dis Loss: 0.7546/0.6437
Epoch: 052/100 | Batch 100/469 | Gen/Dis Loss: 0.8014/0.6632
Epoch: 052/100 | Batch 200/469 | Gen/Dis Loss: 0.8867/0.6125
Epoch: 052/100 | Batch 300/469 | Gen/Dis Loss: 0.8593/0.6246
Epoch: 052/100 | Batch 400/469 | Gen/Dis Loss: 1.0143/0.6210
Time elapsed: 11.46 min
Epoch: 053/100 | Batch 000/469 | Gen/Dis Loss: 0.8539/0.6260
Epoch: 053/100 | Batch 100/469 | Gen/Dis Loss: 0.7258/0.6664
Epoch: 053/100 | Batch 200/469 | Gen/Dis Loss: 0.8724/0.6416
Epoch: 053/100 | Batch 300/469 | Gen/Dis Loss: 0.8666/0.6325
Epoch: 053/100 | Batch 400/469 | Gen/Dis Loss: 1.1027/0.6079
Time elapsed: 11.75 min
Epoch: 054/100 | Batch 000/469 | Gen/Dis Loss: 0.8014/0.6610
Epoch: 054/100 | Batch 100/469 | Gen/Dis Loss: 0.7267/0.6432
```

```
Epoch: 054/100 | Batch 200/469 | Gen/Dis Loss: 0.9097/0.6240
Epoch: 054/100 | Batch 300/469 | Gen/Dis Loss: 0.7457/0.6150
Epoch: 054/100 | Batch 400/469 | Gen/Dis Loss: 0.9077/0.6315
Time elapsed: 11.98 min
Epoch: 055/100 | Batch 000/469 | Gen/Dis Loss: 0.8604/0.6533
Epoch: 055/100 | Batch 100/469 | Gen/Dis Loss: 0.8062/0.6420
Epoch: 055/100 | Batch 200/469 | Gen/Dis Loss: 0.8209/0.6615
Epoch: 055/100 | Batch 300/469 | Gen/Dis Loss: 0.8918/0.6377
Epoch: 055/100 | Batch 400/469 | Gen/Dis Loss: 0.9119/0.6437
Time elapsed: 12.23 min
Epoch: 056/100 | Batch 000/469 | Gen/Dis Loss: 0.7633/0.6261
Epoch: 056/100 | Batch 100/469 | Gen/Dis Loss: 0.6655/0.6866
Epoch: 056/100 | Batch 200/469 | Gen/Dis Loss: 0.7852/0.6559
Epoch: 056/100 | Batch 300/469 | Gen/Dis Loss: 0.8242/0.6614
Epoch: 056/100 | Batch 400/469 | Gen/Dis Loss: 0.8658/0.6213
Time elapsed: 12.48 min
Epoch: 057/100 | Batch 000/469 | Gen/Dis Loss: 0.8262/0.6221
Epoch: 057/100 | Batch 100/469 | Gen/Dis Loss: 1.0824/0.6364
Epoch: 057/100 | Batch 200/469 | Gen/Dis Loss: 0.9147/0.6439
Epoch: 057/100 | Batch 300/469 | Gen/Dis Loss: 0.8846/0.6504
Epoch: 057/100 | Batch 400/469 | Gen/Dis Loss: 0.8016/0.6304
Time elapsed: 12.73 min
Epoch: 058/100 | Batch 000/469 | Gen/Dis Loss: 0.8156/0.6550
Epoch: 058/100 | Batch 100/469 | Gen/Dis Loss: 0.8336/0.6293
Epoch: 058/100 | Batch 200/469 | Gen/Dis Loss: 0.8739/0.6302
Epoch: 058/100 | Batch 300/469 | Gen/Dis Loss: 0.7949/0.6394
Epoch: 058/100 | Batch 400/469 | Gen/Dis Loss: 1.0678/0.6345
Time elapsed: 12.96 min
Epoch: 059/100 | Batch 000/469 | Gen/Dis Loss: 0.9281/0.6529
Epoch: 059/100 | Batch 100/469 | Gen/Dis Loss: 1.0390/0.5976
Epoch: 059/100 | Batch 200/469 | Gen/Dis Loss: 0.8520/0.6345
Epoch: 059/100 | Batch 300/469 | Gen/Dis Loss: 0.7449/0.6629
Epoch: 059/100 | Batch 400/469 | Gen/Dis Loss: 0.9793/0.6175
Time elapsed: 13.21 min
Epoch: 060/100 | Batch 000/469 | Gen/Dis Loss: 0.8872/0.6448
Epoch: 060/100 | Batch 100/469 | Gen/Dis Loss: 0.8577/0.6347
Epoch: 060/100 | Batch 200/469 | Gen/Dis Loss: 0.7144/0.6646
Epoch: 060/100 | Batch 300/469 | Gen/Dis Loss: 0.8494/0.6390
Epoch: 060/100 | Batch 400/469 | Gen/Dis Loss: 0.8207/0.6161
Time elapsed: 13.45 min
Epoch: 061/100 | Batch 000/469 | Gen/Dis Loss: 0.8034/0.6211
Epoch: 061/100 | Batch 100/469 | Gen/Dis Loss: 0.9030/0.6405
Epoch: 061/100 | Batch 200/469 | Gen/Dis Loss: 1.1420/0.6369
Epoch: 061/100 | Batch 300/469 | Gen/Dis Loss: 0.7998/0.6654
Epoch: 061/100 | Batch 400/469 | Gen/Dis Loss: 0.8823/0.6318
Time elapsed: 13.69 min
Epoch: 062/100 | Batch 000/469 | Gen/Dis Loss: 0.8928/0.6329
Epoch: 062/100 | Batch 100/469 | Gen/Dis Loss: 1.0672/0.5925
Epoch: 062/100 | Batch 200/469 | Gen/Dis Loss: 0.9199/0.6389
Epoch: 062/100 | Batch 300/469 | Gen/Dis Loss: 0.9460/0.6272
Epoch: 062/100 | Batch 400/469 | Gen/Dis Loss: 0.7696/0.6317
Time elapsed: 13.92 min
Epoch: 063/100 | Batch 000/469 | Gen/Dis Loss: 0.8032/0.6188
Epoch: 063/100 | Batch 100/469 | Gen/Dis Loss: 0.8523/0.6255
Epoch: 063/100 | Batch 200/469 | Gen/Dis Loss: 0.8331/0.6165
Epoch: 063/100 | Batch 300/469 | Gen/Dis Loss: 0.9053/0.6025
Epoch: 063/100 | Batch 400/469 | Gen/Dis Loss: 0.7981/0.6805
Time elapsed: 14.16 min
Epoch: 064/100 | Batch 000/469 | Gen/Dis Loss: 0.9514/0.6040
Epoch: 064/100 | Batch 100/469 | Gen/Dis Loss: 0.8829/0.6090
Epoch: 064/100 | Batch 200/469 | Gen/Dis Loss: 0.9718/0.6152
Epoch: 064/100 | Batch 300/469 | Gen/Dis Loss: 0.8870/0.6212
Epoch: 064/100 | Batch 400/469 | Gen/Dis Loss: 0.8058/0.6274
Time elapsed: 14.37 min
Epoch: 065/100 | Batch 000/469 | Gen/Dis Loss: 1.0966/0.6199
Epoch: 065/100 | Batch 100/469 | Gen/Dis Loss: 0.9423/0.6327
Epoch: 065/100 | Batch 200/469 | Gen/Dis Loss: 0.9604/0.6319
Epoch: 065/100 | Batch 300/469 | Gen/Dis Loss: 0.9357/0.7044
Epoch: 065/100 | Batch 400/469 | Gen/Dis Loss: 0.7764/0.6162
```

```
Time elapsed: 14.61 min
Epoch: 066/100 | Batch 000/469 | Gen/Dis Loss: 0.8263/0.6414
Epoch: 066/100 | Batch 100/469 | Gen/Dis Loss: 0.8937/0.6359
Epoch: 066/100 | Batch 200/469 | Gen/Dis Loss: 0.9016/0.5980
Epoch: 066/100 | Batch 300/469 | Gen/Dis Loss: 1.0026/0.6003
Epoch: 066/100 | Batch 400/469 | Gen/Dis Loss: 1.0411/0.5995
Time elapsed: 14.83 min
Epoch: 067/100 | Batch 000/469 | Gen/Dis Loss: 0.8075/0.6471
Epoch: 067/100 | Batch 100/469 | Gen/Dis Loss: 0.7602/0.6657
Epoch: 067/100 | Batch 200/469 | Gen/Dis Loss: 0.8805/0.6219
Epoch: 067/100 | Batch 300/469 | Gen/Dis Loss: 0.8166/0.6294
Epoch: 067/100 | Batch 400/469 | Gen/Dis Loss: 0.8649/0.6536
Time elapsed: 15.06 min
Epoch: 068/100 | Batch 000/469 | Gen/Dis Loss: 0.8024/0.6385
Epoch: 068/100 | Batch 100/469 | Gen/Dis Loss: 0.7368/0.6687
Epoch: 068/100 | Batch 200/469 | Gen/Dis Loss: 0.7788/0.6443
Epoch: 068/100 | Batch 300/469 | Gen/Dis Loss: 0.8080/0.6618
Epoch: 068/100 | Batch 400/469 | Gen/Dis Loss: 0.9293/0.6562
Time elapsed: 15.35 min
Epoch: 069/100 | Batch 000/469 | Gen/Dis Loss: 0.8477/0.6086
Epoch: 069/100 | Batch 100/469 | Gen/Dis Loss: 0.9098/0.6121
Epoch: 069/100 | Batch 200/469 | Gen/Dis Loss: 0.7648/0.6389
Epoch: 069/100 | Batch 300/469 | Gen/Dis Loss: 0.9560/0.6190
Epoch: 069/100 | Batch 400/469 | Gen/Dis Loss: 0.8296/0.5983
Time elapsed: 15.55 min
Epoch: 070/100 | Batch 000/469 | Gen/Dis Loss: 0.7523/0.6666
Epoch: 070/100 | Batch 100/469 | Gen/Dis Loss: 0.7846/0.6529
Epoch: 070/100 | Batch 200/469 | Gen/Dis Loss: 0.9757/0.6385
Epoch: 070/100 | Batch 300/469 | Gen/Dis Loss: 1.1968/0.6099
Epoch: 070/100 | Batch 400/469 | Gen/Dis Loss: 0.7996/0.6532
Time elapsed: 15.79 min
Epoch: 071/100 | Batch 000/469 | Gen/Dis Loss: 0.8963/0.6730
Epoch: 071/100 | Batch 100/469 | Gen/Dis Loss: 1.0414/0.6217
Epoch: 071/100 | Batch 200/469 | Gen/Dis Loss: 0.6961/0.6473
Epoch: 071/100 | Batch 300/469 | Gen/Dis Loss: 0.8585/0.6967
Epoch: 071/100 | Batch 400/469 | Gen/Dis Loss: 0.9822/0.6616
Time elapsed: 16.00 min
Epoch: 072/100 | Batch 000/469 | Gen/Dis Loss: 0.8398/0.6352
Epoch: 072/100 | Batch 100/469 | Gen/Dis Loss: 0.9294/0.6267
Epoch: 072/100 | Batch 200/469 | Gen/Dis Loss: 0.8173/0.6460
Epoch: 072/100 | Batch 300/469 | Gen/Dis Loss: 0.8092/0.5969
Epoch: 072/100 | Batch 400/469 | Gen/Dis Loss: 0.8368/0.6070
Time elapsed: 16.26 min
Epoch: 073/100 | Batch 000/469 | Gen/Dis Loss: 0.8562/0.6346
Epoch: 073/100 | Batch 100/469 | Gen/Dis Loss: 0.8194/0.6662
Epoch: 073/100 | Batch 200/469 | Gen/Dis Loss: 0.8147/0.6533
Epoch: 073/100 | Batch 300/469 | Gen/Dis Loss: 0.8100/0.6370
Epoch: 073/100 | Batch 400/469 | Gen/Dis Loss: 0.7502/0.6730
Time elapsed: 16.50 min
Epoch: 074/100 | Batch 000/469 | Gen/Dis Loss: 0.9910/0.6028
Epoch: 074/100 | Batch 100/469 | Gen/Dis Loss: 0.8630/0.6215
Epoch: 074/100 | Batch 200/469 | Gen/Dis Loss: 0.7791/0.6552
Epoch: 074/100 | Batch 300/469 | Gen/Dis Loss: 0.7393/0.6499
Epoch: 074/100 | Batch 400/469 | Gen/Dis Loss: 0.7767/0.6400
Time elapsed: 16.76 min
Epoch: 075/100 | Batch 000/469 | Gen/Dis Loss: 0.8337/0.6043
Epoch: 075/100 | Batch 100/469 | Gen/Dis Loss: 0.7942/0.6237
Epoch: 075/100 | Batch 200/469 | Gen/Dis Loss: 0.7966/0.6378
Epoch: 075/100 | Batch 300/469 | Gen/Dis Loss: 0.8452/0.6111
Epoch: 075/100 | Batch 400/469 | Gen/Dis Loss: 0.8375/0.6285
Time elapsed: 17.00 min
Epoch: 076/100 | Batch 000/469 | Gen/Dis Loss: 0.9601/0.6112
Epoch: 076/100 | Batch 100/469 | Gen/Dis Loss: 0.8372/0.6297
Epoch: 076/100 | Batch 200/469 | Gen/Dis Loss: 0.8599/0.6622
Epoch: 076/100 | Batch 300/469 | Gen/Dis Loss: 0.8358/0.6348
Epoch: 076/100 | Batch 400/469 | Gen/Dis Loss: 0.8049/0.6253
Time elapsed: 17.22 min
Epoch: 077/100 | Batch 000/469 | Gen/Dis Loss: 0.8975/0.6311
Epoch: 077/100 | Batch 100/469 | Gen/Dis Loss: 0.8132/0.6238
```

```
Epoch: 077/100 | Batch 200/469 | Gen/Dis Loss: 0.7812/0.6634
Epoch: 077/100 | Batch 300/469 | Gen/Dis Loss: 0.9771/0.6255
Epoch: 077/100 | Batch 400/469 | Gen/Dis Loss: 0.8582/0.6255
Time elapsed: 17.48 min
Epoch: 078/100 | Batch 000/469 | Gen/Dis Loss: 0.8319/0.6560
Epoch: 078/100 | Batch 100/469 | Gen/Dis Loss: 0.9870/0.5862
Epoch: 078/100 | Batch 200/469 | Gen/Dis Loss: 1.1849/0.5715
Epoch: 078/100 | Batch 300/469 | Gen/Dis Loss: 1.1314/0.6836
Epoch: 078/100 | Batch 400/469 | Gen/Dis Loss: 0.7721/0.6525
Time elapsed: 17.72 min
Epoch: 079/100 | Batch 000/469 | Gen/Dis Loss: 0.7220/0.6426
Epoch: 079/100 | Batch 100/469 | Gen/Dis Loss: 0.7230/0.6273
Epoch: 079/100 | Batch 200/469 | Gen/Dis Loss: 1.1738/0.6662
Epoch: 079/100 | Batch 300/469 | Gen/Dis Loss: 0.9005/0.6252
Epoch: 079/100 | Batch 400/469 | Gen/Dis Loss: 0.7750/0.6464
Time elapsed: 18.00 min
Epoch: 080/100 | Batch 000/469 | Gen/Dis Loss: 0.8295/0.6295
Epoch: 080/100 | Batch 100/469 | Gen/Dis Loss: 0.8663/0.6554
Epoch: 080/100 | Batch 200/469 | Gen/Dis Loss: 0.8364/0.6609
Epoch: 080/100 | Batch 300/469 | Gen/Dis Loss: 0.7880/0.6376
Epoch: 080/100 | Batch 400/469 | Gen/Dis Loss: 0.8634/0.6717
Time elapsed: 18.42 min
Epoch: 081/100 | Batch 000/469 | Gen/Dis Loss: 0.7117/0.6424
Epoch: 081/100 | Batch 100/469 | Gen/Dis Loss: 0.8929/0.6514
Epoch: 081/100 | Batch 200/469 | Gen/Dis Loss: 0.8996/0.6247
Epoch: 081/100 | Batch 300/469 | Gen/Dis Loss: 0.8245/0.6668
Epoch: 081/100 | Batch 400/469 | Gen/Dis Loss: 0.7557/0.6659
Time elapsed: 18.63 min
Epoch: 082/100 | Batch 000/469 | Gen/Dis Loss: 1.2211/0.6186
Epoch: 082/100 | Batch 100/469 | Gen/Dis Loss: 0.8101/0.6184
Epoch: 082/100 | Batch 200/469 | Gen/Dis Loss: 0.7458/0.6828
Epoch: 082/100 | Batch 300/469 | Gen/Dis Loss: 0.9857/0.6592
Epoch: 082/100 | Batch 400/469 | Gen/Dis Loss: 1.1430/0.6307
Time elapsed: 18.84 min
Epoch: 083/100 | Batch 000/469 | Gen/Dis Loss: 0.8097/0.6529
Epoch: 083/100 | Batch 100/469 | Gen/Dis Loss: 0.8299/0.6105
Epoch: 083/100 | Batch 200/469 | Gen/Dis Loss: 0.8595/0.6670
Epoch: 083/100 | Batch 300/469 | Gen/Dis Loss: 0.8077/0.6430
Epoch: 083/100 | Batch 400/469 | Gen/Dis Loss: 1.0253/0.6076
Time elapsed: 19.05 min
Epoch: 084/100 | Batch 000/469 | Gen/Dis Loss: 0.7838/0.6275
Epoch: 084/100 | Batch 100/469 | Gen/Dis Loss: 0.8083/0.6147
Epoch: 084/100 | Batch 200/469 | Gen/Dis Loss: 0.7782/0.6396
Epoch: 084/100 | Batch 300/469 | Gen/Dis Loss: 0.8773/0.6287
Epoch: 084/100 | Batch 400/469 | Gen/Dis Loss: 0.9187/0.6337
Time elapsed: 19.26 min
Epoch: 085/100 | Batch 000/469 | Gen/Dis Loss: 0.9819/0.6251
Epoch: 085/100 | Batch 100/469 | Gen/Dis Loss: 0.7306/0.6554
Epoch: 085/100 | Batch 200/469 | Gen/Dis Loss: 0.8541/0.6376
Epoch: 085/100 | Batch 300/469 | Gen/Dis Loss: 0.7771/0.6319
Epoch: 085/100 | Batch 400/469 | Gen/Dis Loss: 0.7148/0.6602
Time elapsed: 19.48 min
Epoch: 086/100 | Batch 000/469 | Gen/Dis Loss: 1.0560/0.5908
Epoch: 086/100 | Batch 100/469 | Gen/Dis Loss: 0.8637/0.6590
Epoch: 086/100 | Batch 200/469 | Gen/Dis Loss: 0.7408/0.6439
Epoch: 086/100 | Batch 300/469 | Gen/Dis Loss: 0.7907/0.6256
Epoch: 086/100 | Batch 400/469 | Gen/Dis Loss: 0.7978/0.6680
Time elapsed: 19.69 min
Epoch: 087/100 | Batch 000/469 | Gen/Dis Loss: 1.1564/0.6111
Epoch: 087/100 | Batch 100/469 | Gen/Dis Loss: 0.7237/0.6324
Epoch: 087/100 | Batch 200/469 | Gen/Dis Loss: 0.7677/0.6681
Epoch: 087/100 | Batch 300/469 | Gen/Dis Loss: 0.8510/0.6370
Epoch: 087/100 | Batch 400/469 | Gen/Dis Loss: 0.8028/0.6308
Time elapsed: 19.90 min
Epoch: 088/100 | Batch 000/469 | Gen/Dis Loss: 0.7685/0.6658
Epoch: 088/100 | Batch 100/469 | Gen/Dis Loss: 0.8040/0.6579
Epoch: 088/100 | Batch 200/469 | Gen/Dis Loss: 0.8205/0.6542
Epoch: 088/100 | Batch 300/469 | Gen/Dis Loss: 0.8986/0.6516
Epoch: 088/100 | Batch 400/469 | Gen/Dis Loss: 0.7769/0.6236
```

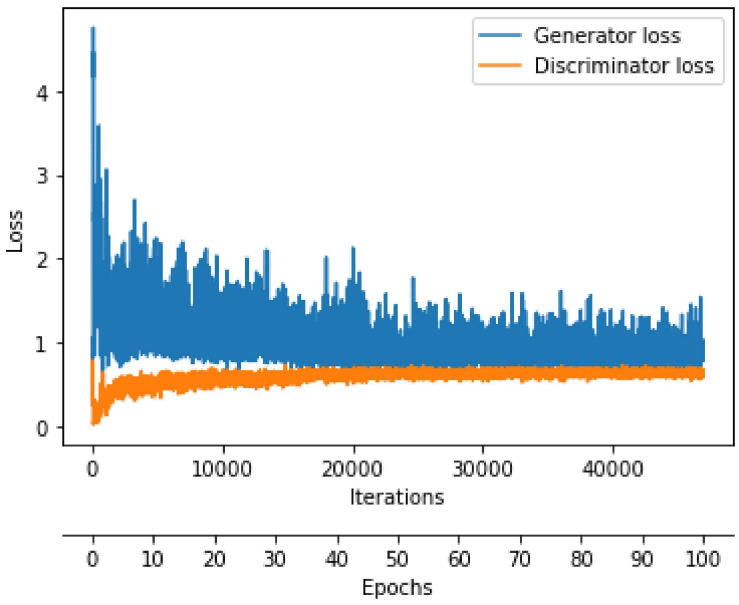
```
Time elapsed: 20.12 min
Epoch: 089/100 | Batch 000/469 | Gen/Dis Loss: 0.8783/0.5702
Epoch: 089/100 | Batch 100/469 | Gen/Dis Loss: 0.8150/0.6399
Epoch: 089/100 | Batch 200/469 | Gen/Dis Loss: 0.9573/0.6198
Epoch: 089/100 | Batch 300/469 | Gen/Dis Loss: 0.8865/0.6421
Epoch: 089/100 | Batch 400/469 | Gen/Dis Loss: 0.9520/0.6323
Time elapsed: 20.33 min
Epoch: 090/100 | Batch 000/469 | Gen/Dis Loss: 0.7949/0.6935
Epoch: 090/100 | Batch 100/469 | Gen/Dis Loss: 0.8458/0.6471
Epoch: 090/100 | Batch 200/469 | Gen/Dis Loss: 0.8740/0.6185
Epoch: 090/100 | Batch 300/469 | Gen/Dis Loss: 0.8003/0.6347
Epoch: 090/100 | Batch 400/469 | Gen/Dis Loss: 0.7020/0.6729
Time elapsed: 20.55 min
Epoch: 091/100 | Batch 000/469 | Gen/Dis Loss: 0.6975/0.6452
Epoch: 091/100 | Batch 100/469 | Gen/Dis Loss: 0.8379/0.6660
Epoch: 091/100 | Batch 200/469 | Gen/Dis Loss: 0.7840/0.6381
Epoch: 091/100 | Batch 300/469 | Gen/Dis Loss: 0.7874/0.6217
Epoch: 091/100 | Batch 400/469 | Gen/Dis Loss: 0.9426/0.6304
Time elapsed: 20.77 min
Epoch: 092/100 | Batch 000/469 | Gen/Dis Loss: 0.8998/0.6600
Epoch: 092/100 | Batch 100/469 | Gen/Dis Loss: 0.9038/0.6546
Epoch: 092/100 | Batch 200/469 | Gen/Dis Loss: 0.9421/0.6394
Epoch: 092/100 | Batch 300/469 | Gen/Dis Loss: 0.8497/0.6253
Epoch: 092/100 | Batch 400/469 | Gen/Dis Loss: 0.8662/0.6154
Time elapsed: 20.99 min
Epoch: 093/100 | Batch 000/469 | Gen/Dis Loss: 0.8021/0.6363
Epoch: 093/100 | Batch 100/469 | Gen/Dis Loss: 0.8512/0.6426
Epoch: 093/100 | Batch 200/469 | Gen/Dis Loss: 0.9419/0.6552
Epoch: 093/100 | Batch 300/469 | Gen/Dis Loss: 0.8228/0.6503
Epoch: 093/100 | Batch 400/469 | Gen/Dis Loss: 0.7939/0.6657
Time elapsed: 21.21 min
Epoch: 094/100 | Batch 000/469 | Gen/Dis Loss: 0.9474/0.6177
Epoch: 094/100 | Batch 100/469 | Gen/Dis Loss: 0.8332/0.6538
Epoch: 094/100 | Batch 200/469 | Gen/Dis Loss: 0.8632/0.6420
Epoch: 094/100 | Batch 300/469 | Gen/Dis Loss: 0.8193/0.6417
Epoch: 094/100 | Batch 400/469 | Gen/Dis Loss: 0.8336/0.6086
Time elapsed: 21.43 min
Epoch: 095/100 | Batch 000/469 | Gen/Dis Loss: 0.8405/0.5922
Epoch: 095/100 | Batch 100/469 | Gen/Dis Loss: 0.7253/0.7023
Epoch: 095/100 | Batch 200/469 | Gen/Dis Loss: 1.0668/0.6835
Epoch: 095/100 | Batch 300/469 | Gen/Dis Loss: 0.7982/0.6697
Epoch: 095/100 | Batch 400/469 | Gen/Dis Loss: 0.8610/0.6149
Time elapsed: 21.65 min
Epoch: 096/100 | Batch 000/469 | Gen/Dis Loss: 0.8360/0.6345
Epoch: 096/100 | Batch 100/469 | Gen/Dis Loss: 0.8215/0.6148
Epoch: 096/100 | Batch 200/469 | Gen/Dis Loss: 0.8496/0.6280
Epoch: 096/100 | Batch 300/469 | Gen/Dis Loss: 0.8134/0.6545
Epoch: 096/100 | Batch 400/469 | Gen/Dis Loss: 0.8732/0.6463
Time elapsed: 21.87 min
Epoch: 097/100 | Batch 000/469 | Gen/Dis Loss: 0.8341/0.6405
Epoch: 097/100 | Batch 100/469 | Gen/Dis Loss: 0.8049/0.6499
Epoch: 097/100 | Batch 200/469 | Gen/Dis Loss: 0.6979/0.6609
Epoch: 097/100 | Batch 300/469 | Gen/Dis Loss: 0.8193/0.6505
Epoch: 097/100 | Batch 400/469 | Gen/Dis Loss: 0.8594/0.6040
Time elapsed: 22.15 min
Epoch: 098/100 | Batch 000/469 | Gen/Dis Loss: 0.8453/0.6326
Epoch: 098/100 | Batch 100/469 | Gen/Dis Loss: 0.8277/0.6393
Epoch: 098/100 | Batch 200/469 | Gen/Dis Loss: 0.7473/0.6400
Epoch: 098/100 | Batch 300/469 | Gen/Dis Loss: 0.8403/0.6336
Epoch: 098/100 | Batch 400/469 | Gen/Dis Loss: 0.8817/0.6329
Time elapsed: 22.41 min
Epoch: 099/100 | Batch 000/469 | Gen/Dis Loss: 0.8962/0.6441
Epoch: 099/100 | Batch 100/469 | Gen/Dis Loss: 0.8487/0.6575
Epoch: 099/100 | Batch 200/469 | Gen/Dis Loss: 0.7715/0.6383
Epoch: 099/100 | Batch 300/469 | Gen/Dis Loss: 0.7481/0.6080
Epoch: 099/100 | Batch 400/469 | Gen/Dis Loss: 0.8113/0.6272
Time elapsed: 22.62 min
Epoch: 100/100 | Batch 000/469 | Gen/Dis Loss: 0.9436/0.6348
Epoch: 100/100 | Batch 100/469 | Gen/Dis Loss: 0.8433/0.6371
```

```
Epoch: 100/100 | Batch 200/469 | Gen/Dis Loss: 0.8108/0.6150
Epoch: 100/100 | Batch 300/469 | Gen/Dis Loss: 0.8266/0.6527
Epoch: 100/100 | Batch 400/469 | Gen/Dis Loss: 0.8515/0.6409
Time elapsed: 22.84 min
Total Training Time: 22.84 min
```

In [7]:

```
##########################
### Evaluation
##########################

ax1 = plt.subplot(1, 1, 1)
ax1.plot(range(len(gener_costs)), gener_costs, label='Generator loss')
ax1.plot(range(len(discr_costs)), discr_costs, label='Discriminator loss')
ax1.set_xlabel('Iterations')
ax1.set_ylabel('Loss')
ax1.legend()

# Set scond x-axis
ax2 = ax1.twiny()
newlabel = list(range(NUM_EPOCHS+1))
iter_per_epoch = len(train_loader)
newpos = [e*iter_per_epoch for e in newlabel]

ax2.set_xticklabels(newlabel[::10])
ax2.set_xticks(newpos[::10])

ax2.xaxis.set_ticks_position('bottom')
ax2.xaxis.set_label_position('bottom')
ax2.spines['bottom'].set_position(('outward', 45))
ax2.set_xlabel('Epochs')
ax2.set_xlim(ax1.get_xlim())
```

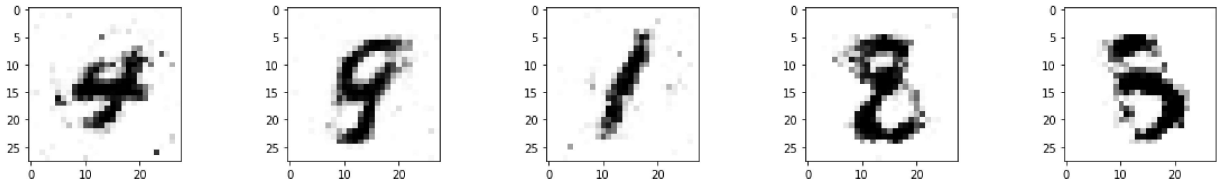
Out[7]: (-2344.9500000000003, 49243.95)



In [11]:

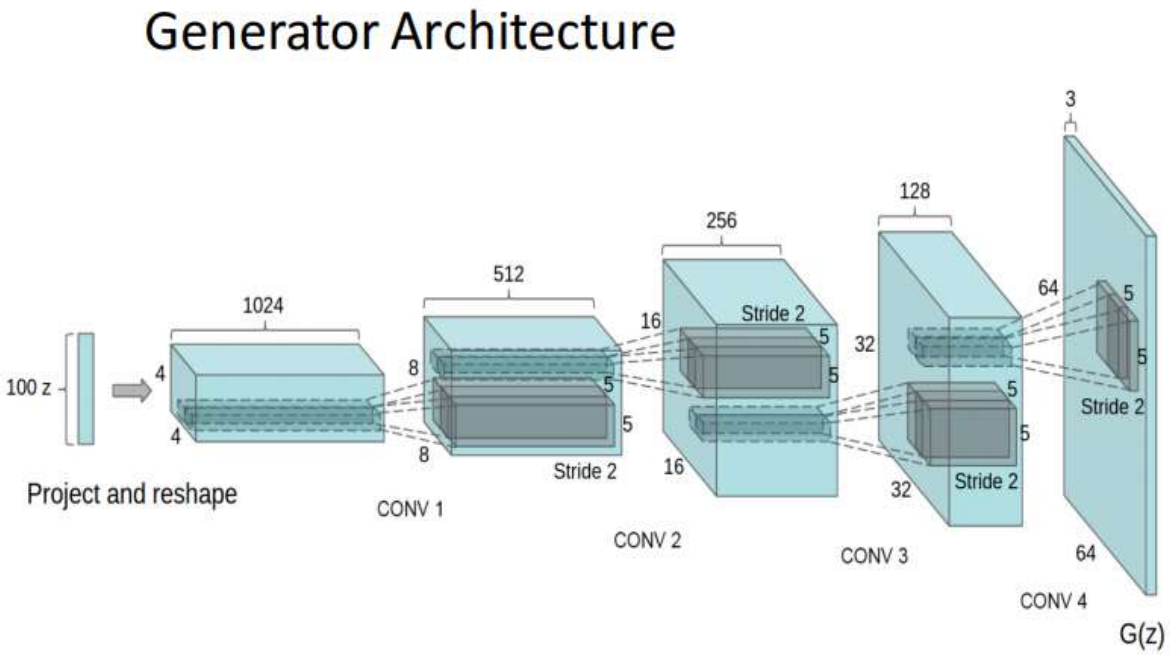
```
##########################
### VISUALIZATION
##########################

model.eval()
# Make new images
z = torch.zeros((5, LATENT_DIM)).uniform_(-1.0, 1.0).to(device)
generated_features = model.generator_forward(z)
imgs = generated_features.view(-1, 28, 28)
```

```
fig, axes = plt.subplots(nrows=1, ncols=5, figsize=(20, 2.5))


for i, ax in enumerate(axes):
    axes[i].imshow(imgs[i].to(torch.device('cpu')).detach(), cmap='binary')
```



Deep Convolutional GANs (DCGANs)

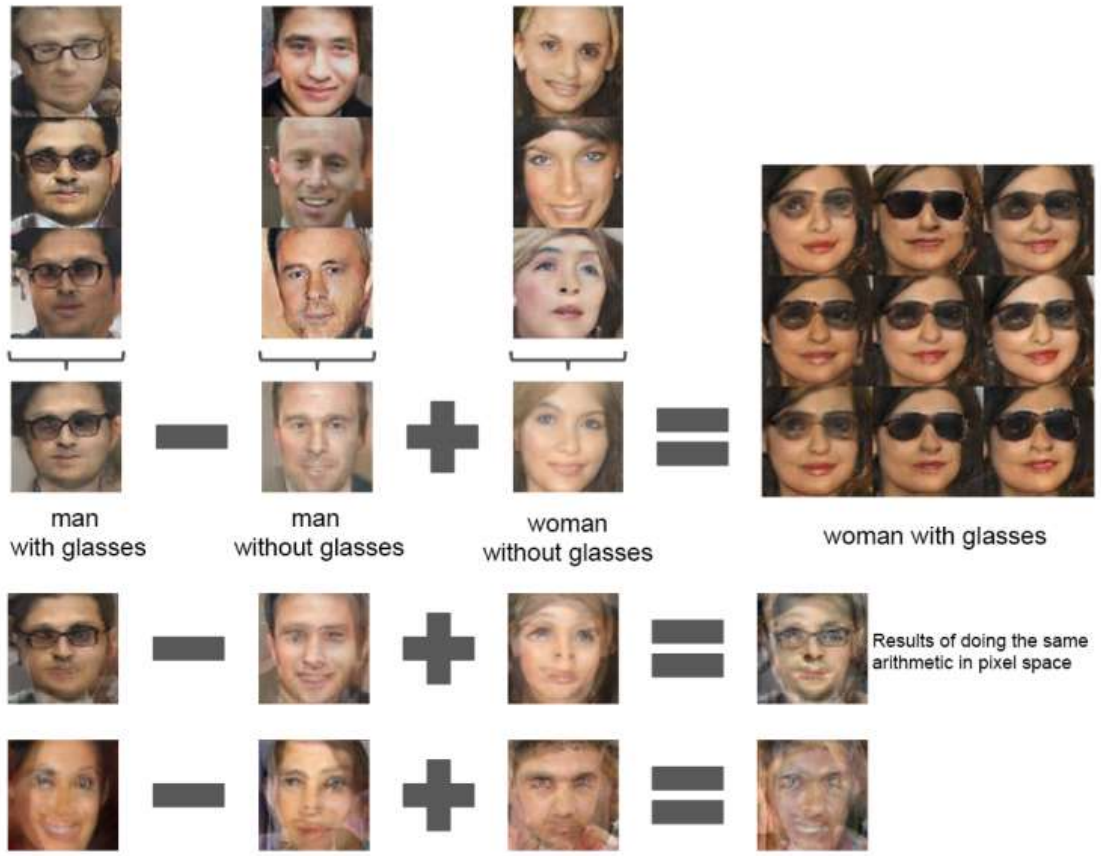
- Key ideas:
  - Replace fully-connected (FC) hidden layers with convolutions.
    - Use fractional/dialated convolutions (to perform the up-convolution from vectors to images).
  - Use *Batch Normalization* after each layer.
  - Activations:
    - Hidden layers are activated with ReLUs.
    - Output layer is activated with Tanh (i.e, the pixel values are normalized between $[-1, 1]$).



The Latent Space

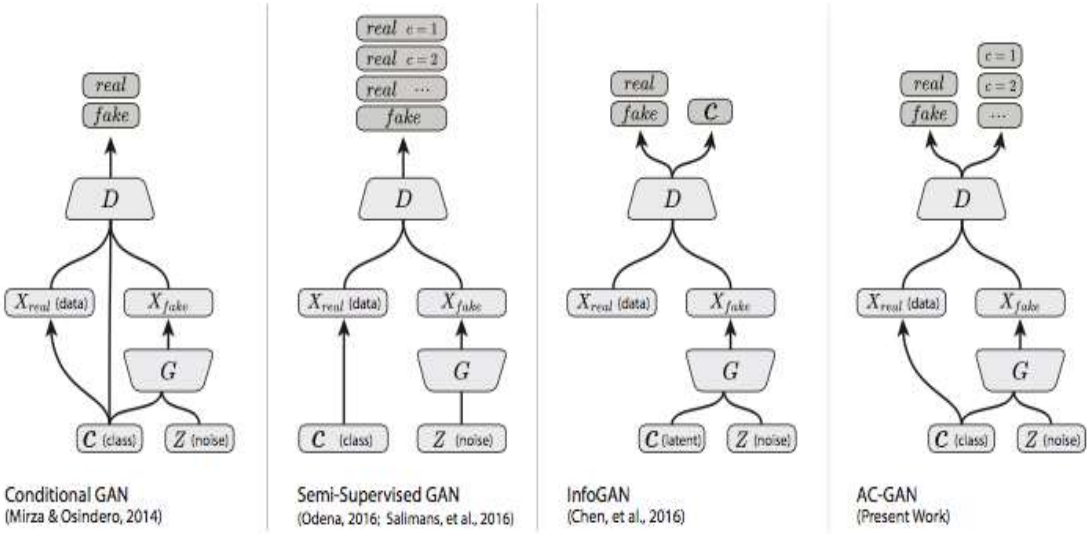
- As we learn how to transform a latent vector, z, to images, we actullay learn a latent continuous space.

- This continuous spcae allows us to perform interpolation and arithmetics.
- As this space is continuous, unlike the original data (images), it was found that some operations (like summing) perform really well when done on the latent space.
- As you can see below, those operations were demonstrated in the paper Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks, Alec Radford, Luke Metz, Soumith Chintala, ICLR 2016



Conditional GANs

- As you probably have noticed, we don't too much control over the latent space, e.g., with vanilla-GAN trained on MNIST we can't control what digit we are generating.
- **Conditional-GANs** - a simple modification to the original GAN framework that *conditions* the model on additional information for better multi-modal learning.
- In practice, we usually use the labels of the datasets to perform the conditioning.
  - For example, on MNIST we will use the one-hot vector representation of the digit ($1 \rightarrow [0, 1, 0, 0, 0, 0, 0, 0, 0, 0]$) along with the images from that class.
- Leads to many practical applications of GANs when we have *explicit supervision available.*
- There is more than one way to perform conditioning, some approaches are presented below.



- Conditional Generative Adversarial Nets, Mehdi Mirza, Simon Osindero
- Conditional GANs

GANs Today

- GANs are **HARD to train** and many researche studies try to improve training stability.
- **WGAN** - Wasserstein GANs use the Wasserstein (Earth Movers) distance as the loss function. Training is more stabilized than vanilla-GAN.
  - **WGAN-GP** - improves upon the original WGAN by using *Gradient Penalty* in the loss function (instead of *value clipping*)
  - WGAN Paper, PyTorch Code
  - WGAN-GP Paper, PyTorch Code
- **EBGAN** - Energy-Based GANs use *autoencoders* in their architecture (with the autoencoder loss).
  - EBGAN Paper, PyTorch Code
- **BEGAN** - Boundary Equilibrium GANs combines *autoencoders* and Wassertein distance to balance the generator and discriminator during training.
  - BEGAN Paper, PyTorch Code
- **Mimicry** - a lightweight PyTorch library aimed towards the reproducibility of GAN research - GitHub

Tips for Training GANs

All tips are here: Tips for Training GANs

- Normalize the inputs - usually between $[-1, 1]$. Use TanH for the Generator output.
- Use the modified loss function to avoid the vanishing gradients.
- Use a spherical Z - sample from a Gaussian distribution instead of uniform distribution.
- BatchNorm (when batchnorm is not an option use instance normalization).

- Avoid Sparse Gradients: ReLU, MaxPool - the stability of the GAN game suffers if you have sparse gradients.
  - LeakyReLU is good (in both G and D)
  - For Downsampling, use: Average Pooling, Conv2d + stride
  - For Upsampling, use: PixelShuffle, ConvTranspose2d + stride

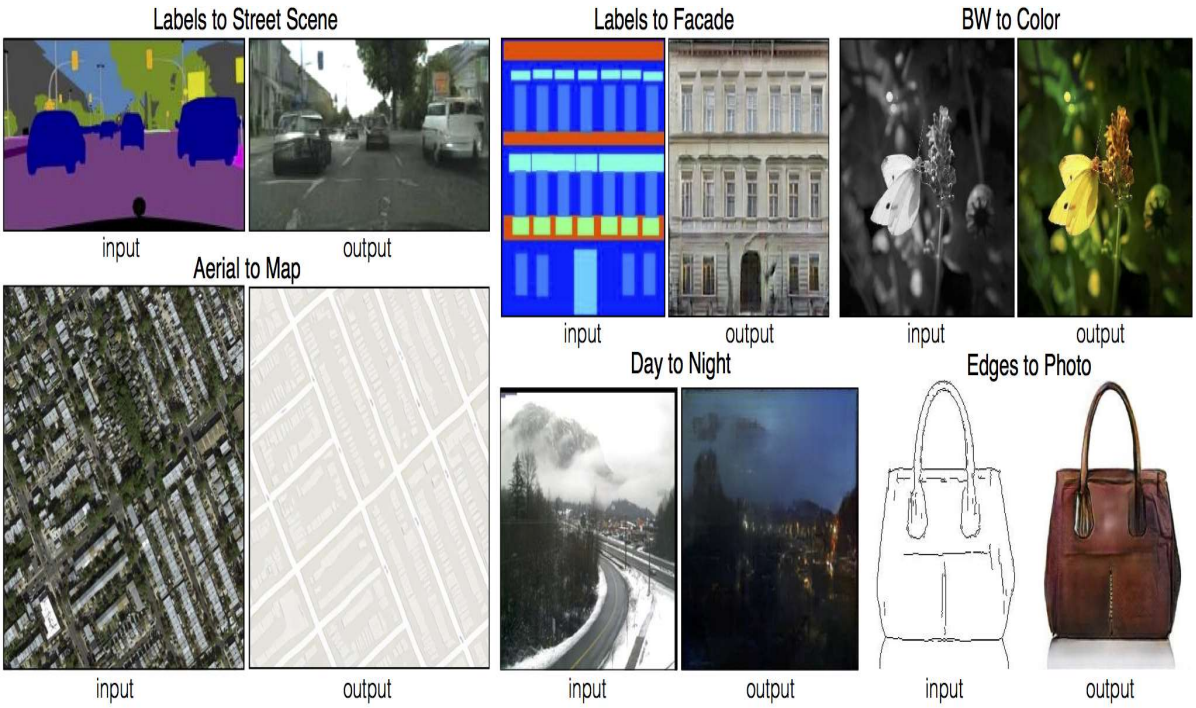
- Use Soft and Noisy Labels
  - Label Smoothing, i.e. if you have two target labels: Real=1 and Fake=0, then for each incoming sample, if it is real, then replace the label with a random number between 0.7 and 1.2, and if it is a fake sample, replace it with 0.0 and 0.3.
  - Make the labels the noisy for the discriminator: occasionally flip the labels when training the discriminator
- Track failures early:
  - D loss goes to 0 -- failure mode.
  - Check norms of gradients: if they are over 100 things are not good...
  - When things are working, D loss has low variance and goes down over time vs. having huge variance and spiking.
- **Don't** balance loss via statistics (unless you have a good reason to)
  - For example, don't do that: `while lossD > A: train D` or `while lossG > B: train G`

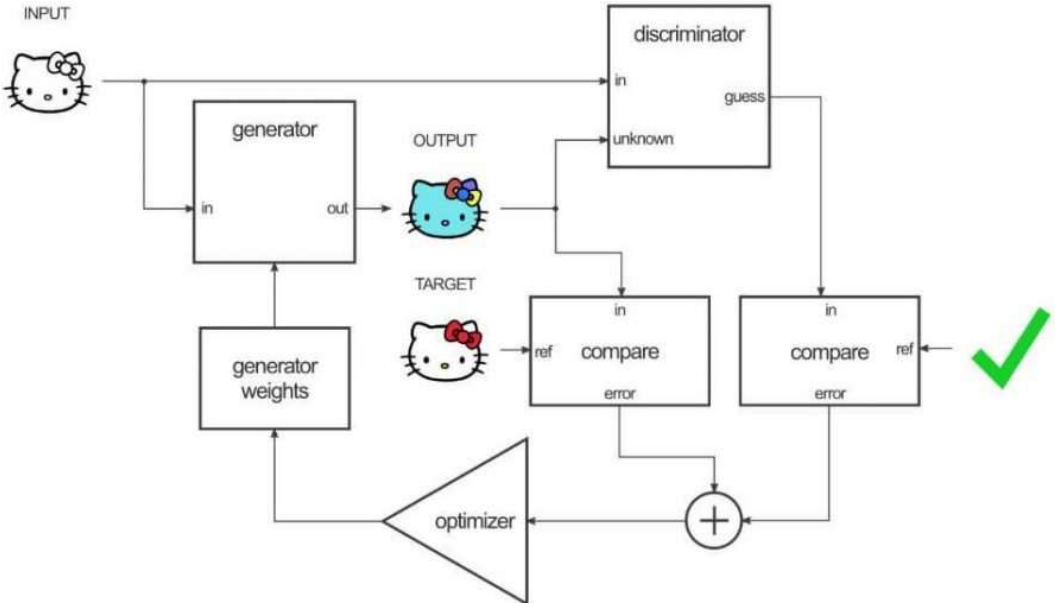
Applications

- There has been great progress in GANs, and everyday there is "a new GAN".
- The current quality of the generations is almost indistinguishable from real images.
  - StyleGAN V2 - ThisPersonDoesNotExist.com
  - SyleGAN V2 - PyTorch Code
- There are many appliactions which we do not cover, but we provide links to projects at the end of this tutorial. We encourage you to explore areas that you find interesting and integrate them in your homework and projects.

Image-to-Image Translation (Pix2Pix)

- Training is conditioned on the images from the source domain.
- Conditional GANs provide an effective way to handle many complex domains without worrying about designing structured loss functions explicitly.
  - These networks not only learn the mapping from input image to output image, but also learn a loss function to train this mapping.
- Project Page
  - PyTorch Code on GitHub
- Edges-to-Cats Demo





- Image Source

- Pix2pix uses a conditional generative adversarial network (cGAN) to learn a function to map from an input image to an output image.
- The **Generator** transforms the input image to get the output image.
- The **Discriminator** measures the similarity of the input image to an unknown image (either a target image from the dataset or an output image from the generator) and tries to guess if it real or fake.

CycleGAN

- For many tasks, paired training data will not be available (like in Pix2Pix).
- **CycleGAN** - an approach for learning to translate an image from a source domain X to a target domain Y in the absence of paired examples.

- The goal is to learn a mapping $G : X \rightarrow Y$ such that the distribution of images from $G(X)$ is indistinguishable from the distribution Y using an adversarial loss.
- Because this mapping is highly under-constrained, it is coupled with an inverse mapping $F : Y \rightarrow X$ and introduce a cycle consistency loss to push $F(G(X)) \approx X$ (and vice versa).

$$\text{Loss}_{cyc}(G, F, X, Y) = \frac{1}{m} \sum_{i=1}^{m} [|| F(G(x_i)) - x_i ||_1] + [|| G(F(y_i)) - y_i ||_1]$$

- The complete loss:

$$\text{Loss}_{full} = \text{Loss}_{adv} + \lambda \text{Loss}_{cyc}$$

  - $\lambda = 10$ in the original implementation.

- Project Page
  - PyTorch Code on GitHub

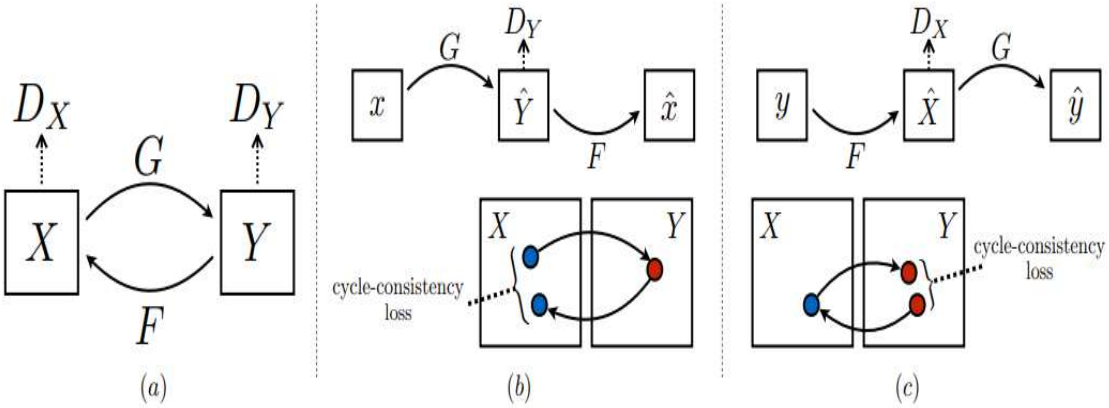
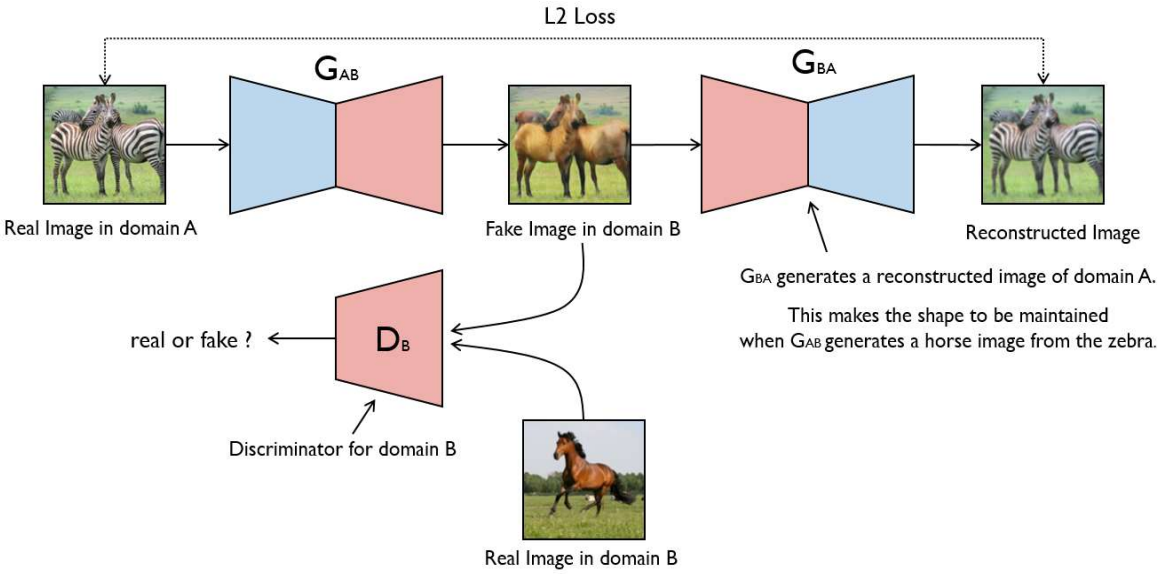
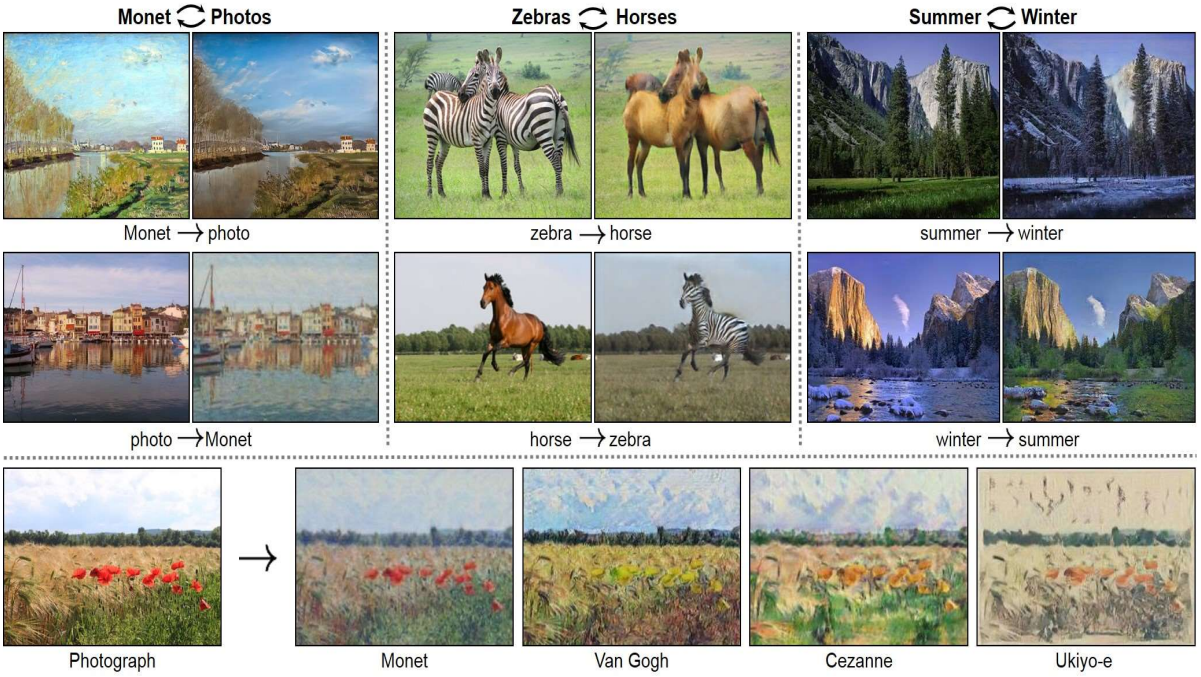


Figure 3: (a) Our model contains two mapping functions $G : X \rightarrow Y$ and $F : Y \rightarrow X$, and associated adversarial discriminators D_Y and D_X. D_Y encourages G to translate X into outputs indistinguishable from domain Y, and vice versa for D_X and F. To further regularize the mappings, we introduce two *cycle consistency losses* that capture the intuition that if we translate from one domain to the other and back again we should arrive at where we started: (b) forward cycle-consistency loss: $x \rightarrow G(x) \rightarrow F(G(x)) \approx x$, and (c) backward cycle-consistency loss: $y \rightarrow F(y) \rightarrow G(F(y)) \approx y$



- Image Source



- Video: [CycleGAN] Rendering Cityscapes in GTA Style

Realistic Neural Talking Head Models

- In order to create a personalized talking head model, it usually requires training on a large dataset of images of a single person.
- However, in many practical scenarios, such personalized talking head models need to be learned from a few image views of a person, potentially even a single image.
- In the paper **"Few-Shot Adversarial Learning of Realistic Neural Talking Head Models"**, a system with such few-shot capability is presented.
- The model performs lengthy meta-learning on a large dataset of videos, and after that it is able to frame few- and one-shot learning of neural talking head models of previously

unseen people as adversarial training problems with high capacity generators and discriminators.

- Paper Link
  - PyTorch Code on GitHub

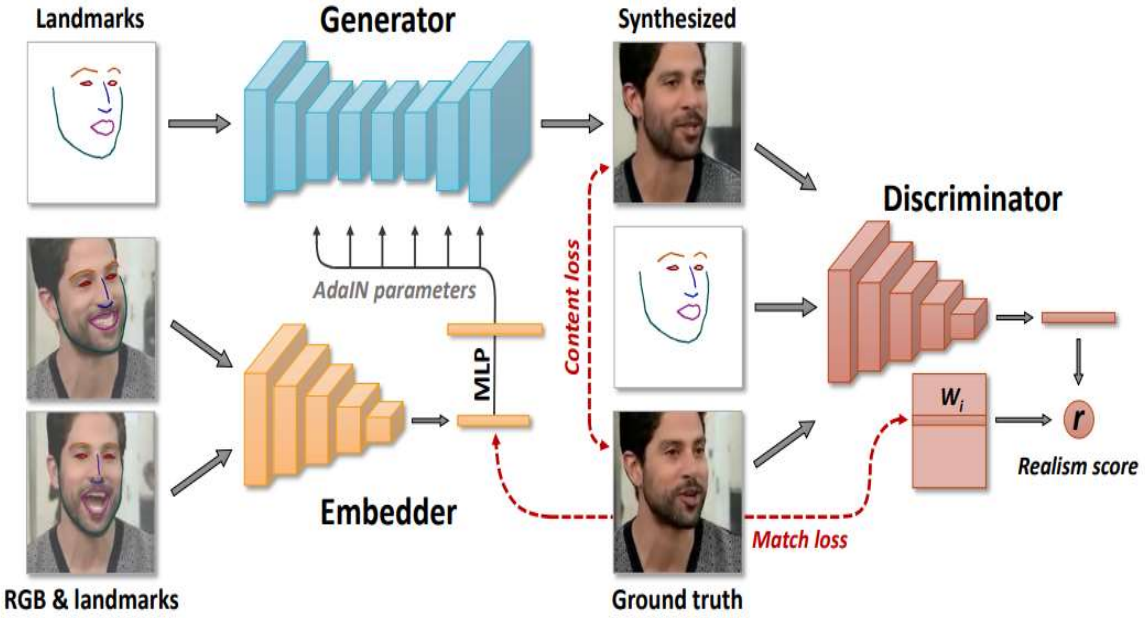
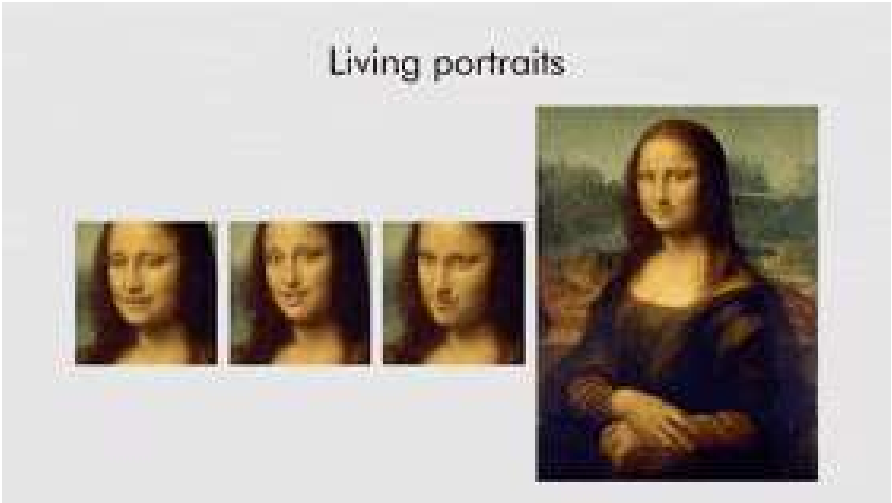


Figure 2: Our meta-learning architecture involves the embedder network that maps head images (with estimated face landmarks) to the embedding vectors, which contain pose-independent information. The generator network maps input face landmarks into output frames through the set of convolutional layers, which are modulated by the embedding vectors via adaptive instance normalization. During meta-learning, we pass sets of frames from the same video through the embedder, average the resulting embeddings and use them to predict adaptive parameters of the generator. Then, we pass the landmarks of a different frame through the generator, comparing the resulting image with the ground truth. Our objective function includes perceptual and adversarial losses, with the latter being implemented via a conditional projection discriminator.



- First Order Motion Model for Image Animation - Aliaksandr Siarohin, Stéphane Lathuilière, Sergey Tulyakov, Elisa Ricci, Nicu Sebe - NeurIPS 2019
  - Code and Colab Demo

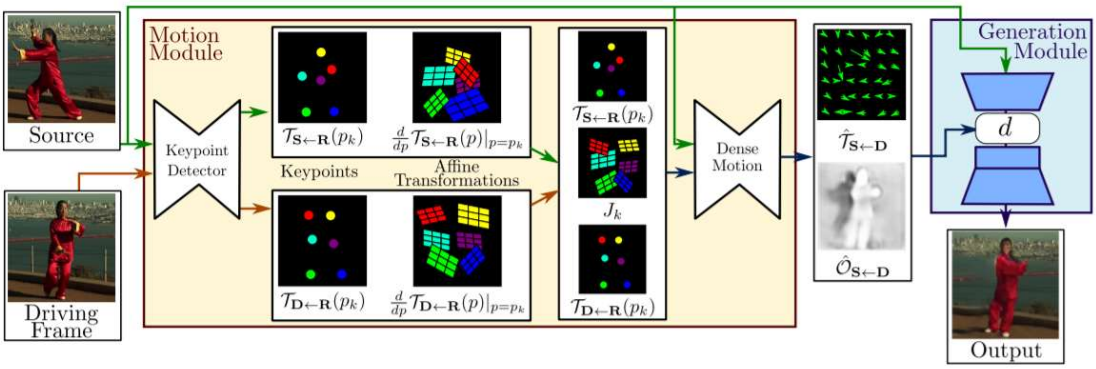


Figure 2: Overview of our approach. Our method assumes a source image $\mathbf{S}$ and a frame of a driving video frame $\mathbf{D}$ as inputs. The unsupervised keypoint detector extracts first order motion representation consisting of sparse keypoints and local affine transformations with respect to the reference frame $\mathbf{R}$. The dense motion network uses the motion representation to generate dense optical flow $\hat{\mathcal{T}}_{\mathbf{S}\leftarrow\mathbf{D}}$ from $\mathbf{D}$ to $\mathbf{S}$ and occlusion map $\hat{\mathcal{O}}_{\mathbf{S}\leftarrow\mathbf{D}}$. The source image and the outputs of the dense motion network are used by the generator to render the target image.

Face Aging with Conditional GANs

- A GAN-based method for automatic face aging.
- This model puts emphasize on preserving the original person's identity in the aged version of his/her face.
- Introduces a novel approach for "Identity-Preserving" optimization of GAN's latent vectors.
- The objective evaluation of the resulting aged and rejuvenated face images is done by the state-of-the-art face recognition and age estimation solutions.
- Paper Link
  - PyTorch Code on GitHub

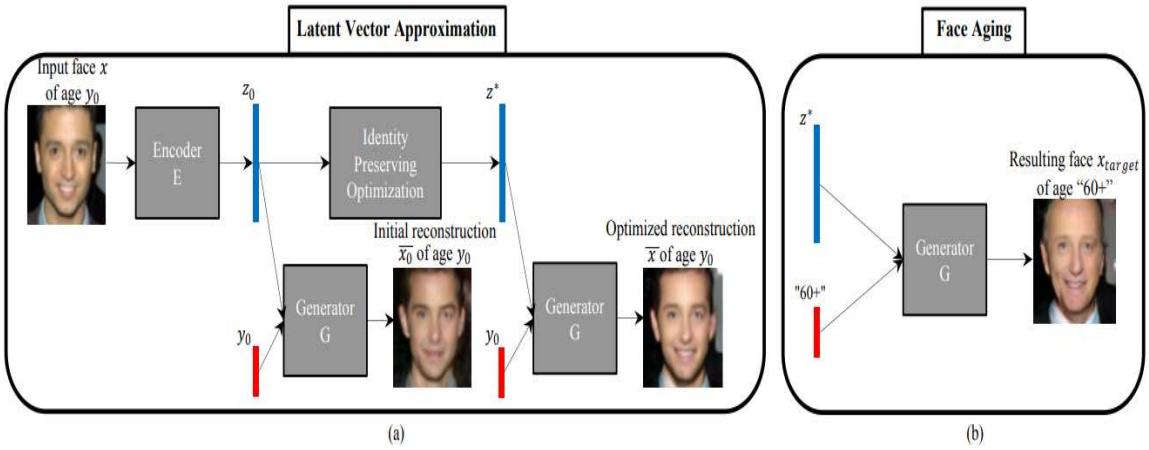


Fig. 1. Our face aging method. (a) approximation of the latent vector to reconstruct the input image; (b) switching the age condition at the input of the generator G to perform face aging.

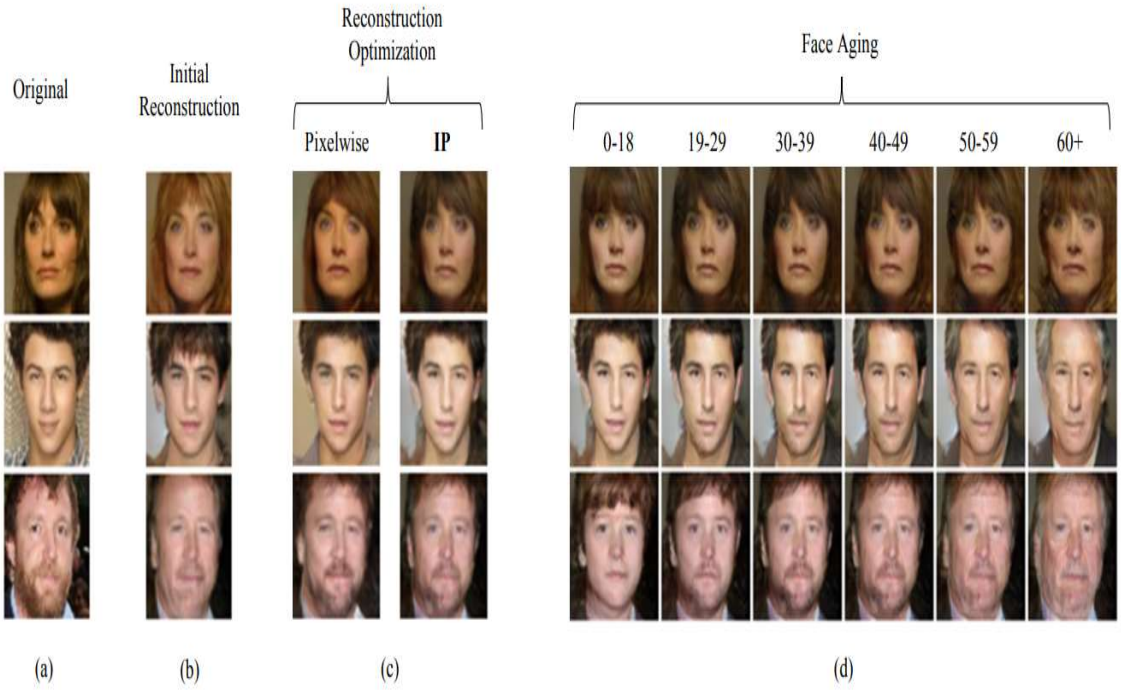


Fig. 3. Examples of face reconstruction and aging. (a) original test images, (b) reconstructed images generated using the initial latent approximations: z_0, (c) reconstructed images generated using the "Pixelwise" and "Identity-Preserving" optimized latent approximations: z^*_{pixel} and z^*_{IP}, and (d) aging of the reconstructed images generated using the identity-preserving z^*_{IP} latent approximations and conditioned on the respective age categories y (one per column).

Cool GAN Projects (with Code)

- gans-awesome-applications
- pytorch-generative-model-collections

Recommended Videos

Warning!

- These videos do not replace the lectures and tutorials.
- Please use these to get a better understanding of the material, and not as an alternative to the written material.

Video By Subject

- Introduction to GANs - Introduction to GANs, NIPS 2016 | Ian Goodfellow, OpenAI
- Generative Models - Stanford CS231n - Lecture 13 | Generative Models
- Deep Generative Modeling - MIT 6.S191 (2019): Deep Generative Modeling
- Face Editing - Face editing with Generative Adversarial Networks

- Wasserstein GANs - Nuts and Bolts of WGANs, Kantorovich-Rubistein Duality, Earth Movers Distance
- Energy-Based GANs - Energy-Based Adversarial Training and Video Prediction, NIPS 2016 | Yann LeCun, Facebook AI Research
- CycleGAN - Zebras, Horses & CycleGAN - Computerphile
- Pix2Pix - Neural Networks: pix2pix (Conditional GANs)

Credits

- Slides from CS 598 LAZ
- Slides by Lihi Zelnik-Mannor
- Slides from CMU - 16720B – Computer Vision
- Some material from Alexander Amini and Ava Soleimany, MIT 6.S191: Introduction to Deep Learning, IntroToDeepLearning.com
- Icons from Icon8.com - https://icons8.com