```
from google.colab import drive

drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

```
import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
import matplotlib.pyplot as plt
from keras.layers import Dense, Dropout, Input, ReLU
from keras.models import Model, Sequential
from tensorflow.keras.optimizers import Adam
import os
for dirname, _, filenames in os.walk('/content/drive/MyDrive/sign'):
    for filename in filenames:
        print(os.path.join(dirname, filename))
```

```
/content/drive/MyDrive/sign/american_sign_language.PNG
/content/drive/MyDrive/sign/amer_sign3.png
/content/drive/MyDrive/sign/sign_mnist_test.csv
/content/drive/MyDrive/sign/sign_mnist_train.csv
/content/drive/MyDrive/sign/amer_sign2.png
/content/drive/MyDrive/sign/sign_mnist_train/sign_mnist_train.csv
/content/drive/MyDrive/sign/sign_mnist_test/sign_mnist_test.csv
```

```
train = pd.read_csv("/content/drive/MyDrive/sign/sign_mnist_train.csv")
test = pd.read_csv("/content/drive/MyDrive/sign/sign_mnist_test.csv")
```

```
print("Train shape: ", train.shape)
train
```

```
Train shape:  (27455, 785)
```

	label	pixel1	pixel2	pixel3	pixel4	pixel5	pixel6	pixel7	pixel8	pixel9
0	3	107	118	127	134	139	143	146	150	153
1	6	155	157	156	156	156	157	156	158	158
2	2	187	188	188	187	187	186	187	188	187
3	2	211	211	212	212	211	210	211	210	210
4	13	164	167	170	172	176	179	180	184	185
...	...	...	...	...	...	...	...	...	...	...
27450	13	189	189	190	190	192	193	193	193	193
27451	23	151	154	157	158	160	161	163	164	166
27452	18	174	174	174	174	174	175	175	174	173
27453	17	177	181	184	185	187	189	190	191	191
27454	23	179	180	180	180	182	181	182	183	182

27455 rows × 785 columns

```
print("Test shape: ", test.shape)
test
```

```
Test shape:  (7172, 785)
```

	label	pixel1	pixel2	pixel3	pixel4	pixel5	pixel6	pixel7	pixel8	pixel9

```
x_train = train.drop(labels = ["label"], axis = 1)
y_train = train["label"]

x_test = test.drop(labels = ["label"], axis = 1)
y_test = test["label"]
```

```
x_train
```

	pixel1	pixel2	pixel3	pixel4	pixel5	pixel6	pixel7	pixel8	pixel9	pixel1
0	107	118	127	134	139	143	146	150	153	15
1	155	157	156	156	156	157	156	158	158	15
2	187	188	188	187	187	186	187	188	187	18
3	211	211	212	212	211	210	211	210	210	21
4	164	167	170	172	176	179	180	184	185	18
...	...	...	...	...	...	...	...	...	...	.
27450	189	189	190	190	192	193	193	193	193	19
27451	151	154	157	158	160	161	163	164	166	16
27452	174	174	174	174	174	175	175	174	173	17
27453	177	181	184	185	187	189	190	191	191	19
27454	179	180	180	180	182	181	182	183	182	18

27455 rows × 784 columns

```
y_train
```

```
0         3
1         6
2         2
3         2
4        13
         ..
27450    13
27451    23
27452    18
27453    17
27454    23
Name: label, Length: 27455, dtype: int64
```

```
k = 0
row, col = 3, 3
fig, ax = plt.subplots(nrows=row, ncols=col, figsize=(16,20),)
for i in range(row):
    for j in range(col):
        img = x_train.iloc[k].to_numpy()
        img = img.reshape((28,28))
        ax[i,j].imshow(img,cmap = "gray")
        ax[i,j].axis("off")
        k += 1
plt.show()
```

```
x_train = x_train / 255.0 # Normalization
x_train
```

	pixel1	pixel2	pixel3	pixel4	pixel5	pixel6	pixel7	pixel8	
0	0.419608	0.462745	0.498039	0.525490	0.545098	0.560784	0.572549	0.588235	0
1	0.607843	0.615686	0.611765	0.611765	0.611765	0.615686	0.611765	0.619608	0
2	0.733333	0.737255	0.737255	0.733333	0.733333	0.729412	0.733333	0.737255	0
3	0.827451	0.827451	0.831373	0.831373	0.827451	0.823529	0.827451	0.823529	0
4	0.643137	0.654902	0.666667	0.674510	0.690196	0.701961	0.705882	0.721569	0
...	...	...	...	...	...	...	...	...	
27450	0.741176	0.741176	0.745098	0.745098	0.752941	0.756863	0.756863	0.756863	0
27451	0.592157	0.603922	0.615686	0.619608	0.627451	0.631373	0.639216	0.643137	0
27452	0.682353	0.682353	0.682353	0.682353	0.682353	0.686275	0.686275	0.682353	0
27453	0.694118	0.709804	0.721569	0.725490	0.733333	0.741176	0.745098	0.749020	0
27454	0.701961	0.705882	0.705882	0.705882	0.713725	0.709804	0.713725	0.717647	0

27455 rows × 784 columns

```
k = 0
row, col = 3, 3
fig, ax = plt.subplots(nrows=row, ncols=col, figsize=(16,20),)
for i in range(row):
    for j in range(col):
        img = x_train.iloc[k].to_numpy()
        img = img.reshape((28,28))
        ax[i,j].imshow(img,cmap = "gray")
        ax[i,j].axis("off")
        k += 1
plt.show()
```

```
x_train = x_train.to_numpy()
x_train = x_train * 2 - 1
print("x_train shape: ", x_train.shape)
x_train
```

```
x_train shape:  (27455, 784)
array([[-0.16078431, -0.0745098 , -0.00392157, ...,  0.6       ,
         0.59215686,  0.58431373],
       [ 0.21568627,  0.23137255,  0.22352941, ..., -0.19215686,
         0.05882353,  0.16862745],
       [ 0.46666667,  0.4745098 ,  0.4745098 , ...,  0.52941176,
         0.52156863,  0.52941176],
       ...,
       [ 0.36470588,  0.36470588,  0.36470588, ...,  0.58431373,
         0.56862745,  0.56862745],
       [ 0.38823529,  0.41960784,  0.44313725, ..., -0.49803922,
        -0.31764706, -0.27058824],
       [ 0.40392157,  0.41176471,  0.41176471, ...,  0.60784314,
         0.63921569,  0.68627451]])
```

```
def create_generator():

    generator = Sequential()
    generator.add(Dense(units = 256, input_dim = 100))
    generator.add(ReLU())

    generator.add(Dense(units = 512))
    generator.add(ReLU())

    generator.add(Dense(units = 1024))
    generator.add(ReLU())

    generator.add(Dense(units = 784, activation = "tanh"))

    generator.compile(loss = "binary_crossentropy",
                      optimizer = Adam(learning_rate = 0.0002, beta_1 = 0.5))
    return generator


g = create_generator()
g.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 dense (Dense)               (None, 256)               25856     
                                                                 
 re_lu (ReLU)                (None, 256)               0         
                                                                 
 dense_1 (Dense)             (None, 512)               131584    
                                                                 
 re_lu_1 (ReLU)              (None, 512)               0         
                                                                 
 dense_2 (Dense)             (None, 1024)              525312    
                                                                 
 re_lu_2 (ReLU)              (None, 1024)              0         
                                                                 
 dense_3 (Dense)             (None, 784)               803600    
                                                                 
=================================================================
Total params: 1486352 (5.67 MB)
Trainable params: 1486352 (5.67 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

```
def create_discriminator():
    discriminator = Sequential()
    discriminator.add(Dense(units = 1024, input_dim = 784))
    discriminator.add(ReLU())
    discriminator.add(Dropout(0.3))

    discriminator.add(Dense(units = 512))
    discriminator.add(ReLU())
    discriminator.add(Dropout(0.3))

    discriminator.add(Dense(units = 256))
    discriminator.add(ReLU())

    discriminator.add(Dense(units = 1, activation = "sigmoid"))

    discriminator.compile(loss = "binary_crossentropy",
                          optimizer = Adam(learning_rate = 0.0002, beta_1 = 0.5))

    return discriminator


d = create_discriminator()
d.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 dense_4 (Dense)             (None, 1024)              803840    
                                                                 
 re_lu_3 (ReLU)              (None, 1024)              0         
                                                                 
 dropout (Dropout)           (None, 1024)              0         
                                                                 
 dense_5 (Dense)             (None, 512)               524800    
                                                                 
 re_lu_4 (ReLU)              (None, 512)               0         
                                                                 
 dropout_1 (Dropout)         (None, 512)               0         
                                                                 
 dense_6 (Dense)             (None, 256)               131328    
                                                                 
 re_lu_5 (ReLU)              (None, 256)               0         
                                                                 
 dense_7 (Dense)             (None, 1)                 257       
                                                                 
=================================================================
Total params: 1460225 (5.57 MB)
Trainable params: 1460225 (5.57 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

```
def create_gan(discriminator, generator):
    discriminator.trainable = False
    gan_input = Input(shape = (100,))
    x = generator(gan_input)
    gan_output = discriminator(x)
    gan = Model(inputs = gan_input, outputs = gan_output)
    gan.compile(loss = "binary_crossentropy", optimizer = "adam")
    return gan


gan = create_gan(d,g)
gan.summary()
```

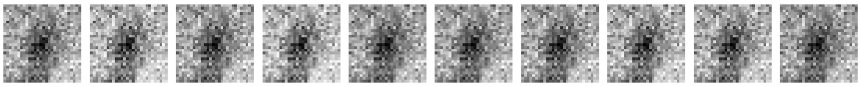
```
Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_1 (InputLayer)        [(None, 100)]             0

 sequential (Sequential)     (None, 784)               1486352

 sequential_1 (Sequential)   (None, 1)                 1460225

=================================================================
Total params: 2946577 (11.24 MB)
Trainable params: 1486352 (5.67 MB)
Non-trainable params: 1460225 (5.57 MB)
_________________________________________________________________
```

```
import time

epochs = 5
batch_size = 128

dis_loss = []
gen_loss = []

for e in range(epochs):
    for _ in range(batch_size):
        start = time.time()
        noise = np.random.normal(0, 1, [batch_size, 100])

        generated_image = g.predict(noise)

        image_batch = x_train[np.random.randint(low = 0, high = x_train.shape[0], size = batch_size)]

        x = np.concatenate([image_batch, generated_image])

        y_dis = np.zeros(batch_size*2)
        y_dis[:batch_size] = 0.9
        d.trainable = True
        dloss = d.train_on_batch(x, y_dis)


        noise = np.random.normal(0, 1, [batch_size, 100])

        y_gen = np.ones(batch_size)

        d.trainable = False

        gloss =  gan.train_on_batch(noise, y_gen)

        end = time.time()
        process_time = str(end - start)

    dis_loss.append(dloss)
    gen_loss.append(gloss)

    print("Epoch: {}, Time: {}s, Generator Loss: {:.3f}, Discriminator Loss: {:.3f}".format(e, process_time[2:4], gloss, dloss))
    4/4 [==============================] - 0s 8ms/step
    4/4 [==============================] - 0s 10ms/step
    4/4 [==============================] - 0s 8ms/step
    4/4 [==============================] - 0s 6ms/step
    4/4 [==============================] - 0s 8ms/step
    4/4 [==============================] - 0s 7ms/step
    4/4 [==============================] - 0s 6ms/step
    4/4 [==============================] - 0s 6ms/step
    4/4 [==============================] - 0s 7ms/step
    4/4 [==============================] - 0s 8ms/step
    4/4 [==============================] - 0s 7ms/step
    4/4 [==============================] - 0s 7ms/step
    4/4 [==============================] - 0s 6ms/step
    4/4 [==============================] - 0s 7ms/step
    4/4 [==============================] - 0s 9ms/step
    4/4 [==============================] - 0s 8ms/step
    4/4 [==============================] - 0s 7ms/step
    4/4 [==============================] - 0s 7ms/step
    4/4 [==============================] - 0s 7ms/step
    4/4 [==============================] - 0s 7ms/step
    4/4 [==============================] - 0s 7ms/step
    4/4 [==============================] - 0s 7ms/step
    4/4 [==============================] - 0s 7ms/step
    4/4 [==============================] - 0s 6ms/step
    4/4 [==============================] - 0s 6ms/step
    4/4 [==============================] - 0s 6ms/step
    4/4 [==============================] - 0s 8ms/step
    4/4 [==============================] - 0s 7ms/step
    4/4 [==============================] - 0s 6ms/step
    4/4 [==============================] - 0s 7ms/step
    4/4 [==============================] - 0s 11ms/step
    4/4 [==============================] - 0s 6ms/step
    4/4 [==============================] - 0s 7ms/step
    4/4 [==============================] - 0s 7ms/step
    4/4 [==============================] - 0s 7ms/step
    4/4 [==============================] - 0s 6ms/step
    4/4 [==============================] - 0s 6ms/step
    4/4 [==============================] - 0s 6ms/step
    4/4 [==============================] - 0s 7ms/step
    Epoch: 4, Time: 23s, Generator Loss: 2.805, Discriminator Loss: 0.179
```

```
g.save_weights("gans_model.h5")
```

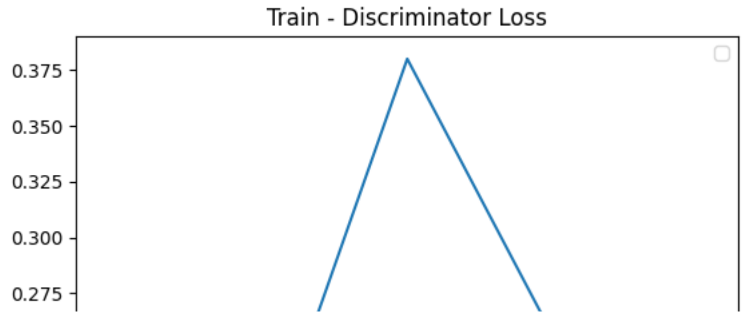
```
noise = np.random.normal(loc = 0, scale = 1, size = [100,100])
generated_image = g.predict(noise)
generated_image = generated_image.reshape(100, 28, 28)
plt.figure(figsize=(15,17))
for i in range(0,10):
    plt.subplot(1, 10, i+1)
    plt.imshow(generated_image[i], interpolation = "nearest", cmap = "gray")
    plt.axis("off")
    plt.tight_layout()
plt.show()
```

```
    4/4 [==============================] - 0s 5ms/step
```



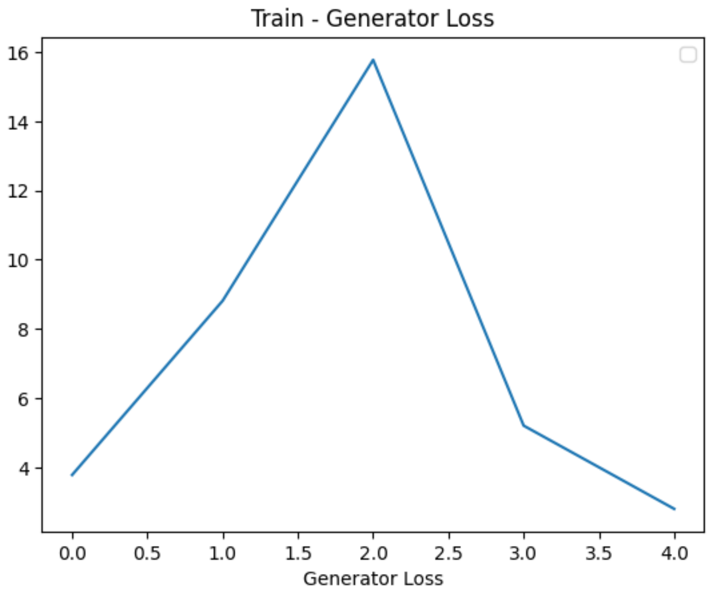
```
epochs_number = []
for i in range(0,epochs):
    epochs_number.append(i)

plt.plot(epochs_number, dis_loss)
plt.title("Train - Discriminator Loss")
plt.xlabel("Number of Epochs")
plt.xlabel("Discriminator Loss")
plt.legend()
plt.show()
```

```
WARNING:matplotlib.legend:No artists with labels found to put in legend.  Note that a
```



```
plt.plot(epochs_number, gen_loss)
plt.title("Train - Generator Loss")
plt.xlabel("Number of Epochs")
plt.xlabel("Generator Loss")
plt.legend()
plt.show()
```

```
WARNING:matplotlib.legend:No artists with labels found to put in legend.  Note that artists whose label start with an underscore are
```


Train - Generator Loss
Generator Loss