1. Import Dependencies and Data

```
!pip install tensorflow tensorflow-gpu matplotlib tensorflow-datasets ipywidgets
```

```
!pip list
```

```
# Bringing in tensorflow
import tensorflow as tf
gpus = tf.config.experimental.list_physical_devices('GPU')
for gpu in gpus:
    tf.config.experimental.set_memory_growth(gpu, True)
```

```
# Brining in tensorflow datasets for fashion mnist
import tensorflow_datasets as tfds
# Bringing in matplotlib for viz stuff
from matplotlib import pyplot as plt
```

```
# Use the tensorflow datasets api to bring in the data source
ds = tfds.load('fashion_mnist', split='train')
```

```
ds.as_numpy_iterator().next()['label']
```

```
2
```

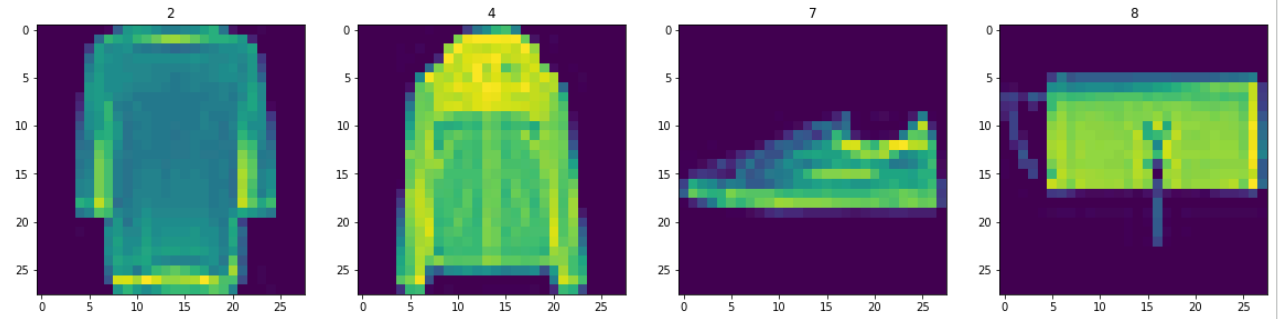
2. Viz Data and Build Dataset

```
# Do some data transformation
import numpy as np
```

```
# Setup connection aka iterator
dataiterator = ds.as_numpy_iterator()
```

```
# Getting data out of the pipeline
dataiterator.next()['image']
```

```
# Setup the subplot formatting
fig, ax = plt.subplots(ncols=4, figsize=(20,20))
# Loop four times and get images
for idx in range(4):
    # Grab an image and label
    sample = dataiterator.next()
    # Plot the image using a specific subplot
    ax[idx].imshow(np.squeeze(sample['image']))
    # Appending the image label as the plot title
    ax[idx].title.set_text(sample['label'])
```




```
# Scale and return images only
def scale_images(data):
    image = data['image']
    return image / 255
```

```
# Reload the dataset
ds = tfds.load('fashion_mnist', split='train')
# Running the dataset through the scale_images preprocessing step
ds = ds.map(scale_images)
# Cache the dataset for that batch
ds = ds.cache()
# Shuffle it up
ds = ds.shuffle(60000)
# Batch into 128 images per sample
ds = ds.batch(128)
# Reduces the likelihood of bottlenecking
ds = ds.prefetch(64)
```

```
ds.as_numpy_iterator().next().shape
```

```
(128, 28, 28, 1)
```

3. Build Neural Network

3.1 Import Modelling Components

```
# Bring in the sequential api for the generator and discriminator
from tensorflow.keras.models import Sequential
# Bring in the layers for the neural network
from tensorflow.keras.layers import Conv2D, Dense, Flatten, Reshape, LeakyReLU, Dropout, UpSampling2D
```

3.2 Build Generator

```
def build_generator():
    model = Sequential()

    # Takes in random values and reshapes it to 7x7x128
    # Beginnings of a generated image
    model.add(Dense(7*7*128, input_dim=128))
    model.add(LeakyReLU(0.2))
    model.add(Reshape((7,7,128)))

    # Upsampling block 1
    model.add(UpSampling2D())
    model.add(Conv2D(128, 5, padding='same'))
    model.add(LeakyReLU(0.2))

    # Upsampling block 2
    model.add(UpSampling2D())
    model.add(Conv2D(128, 5, padding='same'))
    model.add(LeakyReLU(0.2))

    # Convolutional block 1
    model.add(Conv2D(128, 4, padding='same'))
    model.add(LeakyReLU(0.2))

    # Convolutional block 2
    model.add(Conv2D(128, 4, padding='same'))
    model.add(LeakyReLU(0.2))

    # Conv layer to get to one channel
    model.add(Conv2D(1, 4, padding='same', activation='sigmoid'))

    return model
```

```
generator = build_generator()
```

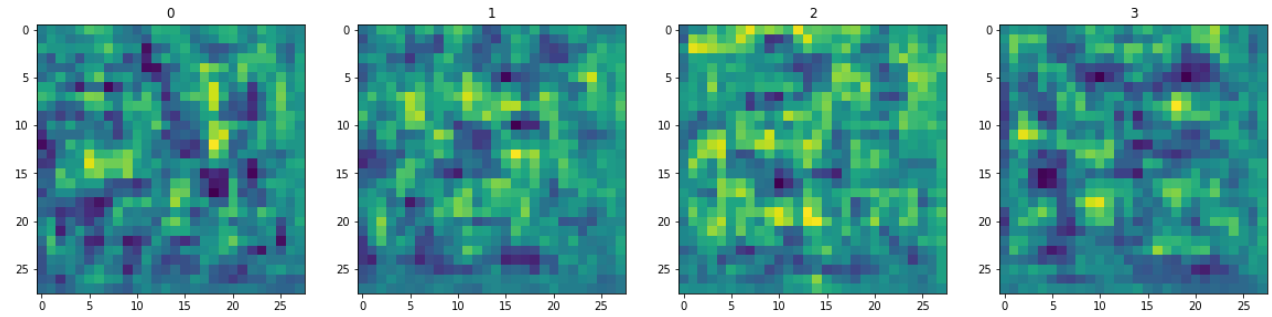
```
generator.summary()
```

```
Model: "sequential_10"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_10 (Dense)            (None, 6272)              809088

 leaky_re_lu_26 (LeakyReLU)  (None, 6272)              0

 reshape_9 (Reshape)         (None, 7, 7, 128)         0

 up_sampling2d_12 (UpSamplin  (None, 14, 14, 128)      0
```

```
g2D)

conv2d_19 (Conv2D)          (None, 14, 14, 128)       409728

leaky_re_lu_27 (LeakyReLU)  (None, 14, 14, 128)       0

up_sampling2d_13 (UpSamplin  (None, 28, 28, 128)      0
g2D)

conv2d_20 (Conv2D)          (None, 28, 28, 128)       409728

leaky_re_lu_28 (LeakyReLU)  (None, 28, 28, 128)       0

conv2d_21 (Conv2D)          (None, 28, 28, 128)       262272

leaky_re_lu_29 (LeakyReLU)  (None, 28, 28, 128)       0

conv2d_22 (Conv2D)          (None, 28, 28, 128)       262272

leaky_re_lu_30 (LeakyReLU)  (None, 28, 28, 128)       0

conv2d_23 (Conv2D)          (None, 28, 28, 1)         2049

=================================================================
Total params: 2,155,137
Trainable params: 2,155,137
Non-trainable params: 0
_________________________________________________________________
```

```
img = generator.predict(np.random.randn(4,128,1))
```

```
# Generate new fashion
img = generator.predict(np.random.randn(4,128,1))
# Setup the subplot formatting
fig, ax = plt.subplots(ncols=4, figsize=(20,20))
# Loop four times and get images
for idx, img in enumerate(img):
    # Plot the image using a specific subplot
    ax[idx].imshow(np.squeeze(img))
    # Appending the image label as the plot title
    ax[idx].title.set_text(idx)
```



3.3 Build Discriminator

```
def build_discriminator():
    model = Sequential()

    # First Conv Block
    model.add(Conv2D(32, 5, input_shape = (28,28,1)))
    model.add(LeakyReLU(0.2))
    model.add(Dropout(0.4))

    # Second Conv Block
    model.add(Conv2D(64, 5))
    model.add(LeakyReLU(0.2))
    model.add(Dropout(0.4))

    # Third Conv Block
    model.add(Conv2D(128, 5))
    model.add(LeakyReLU(0.2))
    model.add(Dropout(0.4))

    # Fourth Conv Block
    model.add(Conv2D(256, 5))
    model.add(LeakyReLU(0.2))
```

```
    model.add(Dropout(0.4))

    # Flatten then pass to dense layer
    model.add(Flatten())
    model.add(Dropout(0.4))
    model.add(Dense(1, activation='sigmoid'))

    return model
```

```
discriminator = build_discriminator()
```

```
discriminator.summary()
```

```
Model: "sequential_15"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_32 (Conv2D)          (None, 24, 24, 32)        832

 leaky_re_lu_39 (LeakyReLU)  (None, 24, 24, 32)        0

 dropout_8 (Dropout)         (None, 24, 24, 32)        0

 conv2d_33 (Conv2D)          (None, 20, 20, 64)        51264

 leaky_re_lu_40 (LeakyReLU)  (None, 20, 20, 64)        0

 dropout_9 (Dropout)         (None, 20, 20, 64)        0

 conv2d_34 (Conv2D)          (None, 16, 16, 128)       204928

 leaky_re_lu_41 (LeakyReLU)  (None, 16, 16, 128)       0

 dropout_10 (Dropout)        (None, 16, 16, 128)       0

 conv2d_35 (Conv2D)          (None, 12, 12, 256)       819456

 leaky_re_lu_42 (LeakyReLU)  (None, 12, 12, 256)       0

 dropout_11 (Dropout)        (None, 12, 12, 256)       0

 flatten (Flatten)           (None, 36864)             0

 dropout_12 (Dropout)        (None, 36864)             0

 dense_11 (Dense)            (None, 1)                 36865

=================================================================
Total params: 1,113,345
Trainable params: 1,113,345
Non-trainable params: 0
_________________________________________________________________
```

```
img = img[0]
```

```
img.shape
```

```
(28, 28, 1)
```

```
discriminator.predict(img)
```

```
array([[0.50057805],
       [0.5006448 ],
       [0.50065   ],
       [0.5005127 ]], dtype=float32)
```

4. Construct Training Loop

4.1 Setup Losses and Optimizers

```
# Adam is going to be the optimizer for both
from tensorflow.keras.optimizers import Adam
# Binary cross entropy is going to be the loss for both
from tensorflow.keras.losses import BinaryCrossentropy
```

```
g_opt = Adam(learning_rate=0.0001)
d_opt = Adam(learning_rate=0.00001)
```

```
g_loss = BinaryCrossentropy()
d_loss = BinaryCrossentropy()
```

4.2 Build Subclassed Model

```
# Importing the base model class to subclass our training step
from tensorflow.keras.models import Model
```

```
class FashionGAN(Model):
    def __init__(self, generator, discriminator, *args, **kwargs):
        # Pass through args and kwargs to base class
        super().__init__(*args, **kwargs)

        # Create attributes for gen and disc
        self.generator = generator
        self.discriminator = discriminator

    def compile(self, g_opt, d_opt, g_loss, d_loss, *args, **kwargs):
        # Compile with base class
        super().compile(*args, **kwargs)

        # Create attributes for losses and optimizers
        self.g_opt = g_opt
        self.d_opt = d_opt
        self.g_loss = g_loss
        self.d_loss = d_loss

    def train_step(self, batch):
        # Get the data
        real_images = batch
        fake_images = self.generator(tf.random.normal((128, 128, 1)), training=False)

        # Train the discriminator
        with tf.GradientTape() as d_tape:
            # Pass the real and fake images to the discriminator model
            yhat_real = self.discriminator(real_images, training=True)
            yhat_fake = self.discriminator(fake_images, training=True)
            yhat_realfake = tf.concat([yhat_real, yhat_fake], axis=0)

            # Create labels for real and fakes images
            y_realfake = tf.concat([tf.zeros_like(yhat_real), tf.ones_like(yhat_fake)], axis=0)

            # Add some noise to the TRUE outputs
            noise_real = 0.15*tf.random.uniform(tf.shape(yhat_real))
            noise_fake = -0.15*tf.random.uniform(tf.shape(yhat_fake))
            y_realfake += tf.concat([noise_real, noise_fake], axis=0)

            # Calculate loss - BINARYCROSS
            total_d_loss = self.d_loss(y_realfake, yhat_realfake)

        # Apply backpropagation - nn learn
        dgrad = d_tape.gradient(total_d_loss, self.discriminator.trainable_variables)
        self.d_opt.apply_gradients(zip(dgrad, self.discriminator.trainable_variables))

        # Train the generator
        with tf.GradientTape() as g_tape:
            # Generate some new images
            gen_images = self.generator(tf.random.normal((128,128,1)), training=True)

            # Create the predicted labels
            predicted_labels = self.discriminator(gen_images, training=False)

            # Calculate loss - trick to training to fake out the discriminator
            total_g_loss = self.g_loss(tf.zeros_like(predicted_labels), predicted_labels)

        # Apply backprop
        ggrad = g_tape.gradient(total_g_loss, self.generator.trainable_variables)
        self.g_opt.apply_gradients(zip(ggrad, self.generator.trainable_variables))

        return {"d_loss":total_d_loss, "g_loss":total_g_loss}
```

```
# Create instance of subclassed model
fashgan = FashionGAN(generator, discriminator)
```

```
# Compile the model
fashgan.compile(g_opt, d_opt, g_loss, d_loss)
```

4.3 Build Callback

```
import os
from tensorflow.keras.preprocessing.image import array_to_img
from tensorflow.keras.callbacks import Callback
```

```
class ModelMonitor(Callback):
    def __init__(self, num_img=3, latent_dim=128):
        self.num_img = num_img
        self.latent_dim = latent_dim

    def on_epoch_end(self, epoch, logs=None):
        random_latent_vectors = tf.random.uniform((self.num_img, self.latent_dim,1))
        generated_images = self.model.generator(random_latent_vectors)
        generated_images *= 255
        generated_images.numpy()
        for i in range(self.num_img):
            img = array_to_img(generated_images[i])
            img.save(os.path.join('images', f'generated_img_{epoch}_{i}.png'))
```

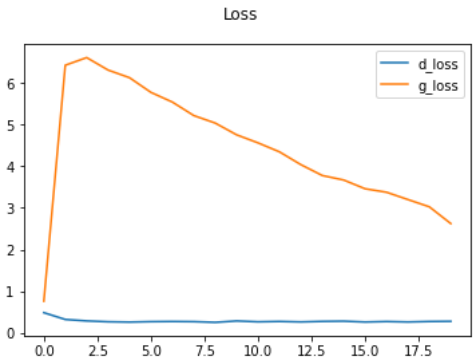
4.3 Train

```
# Recommend 2000 epochs
hist = fashgan.fit(ds, epochs=20, callbacks=[ModelMonitor()])
```

```
Epoch 1/20
469/469 [==============================] - 43s 88ms/step - d_loss: 0.5911 - g_loss: 0.6832
Epoch 2/20
469/469 [==============================] - 41s 88ms/step - d_loss: 0.4158 - g_loss: 3.2742
Epoch 3/20
469/469 [==============================] - 40s 85ms/step - d_loss: 0.2878 - g_loss: 6.5225
Epoch 4/20
469/469 [==============================] - 40s 85ms/step - d_loss: 0.2780 - g_loss: 6.5009
Epoch 5/20
469/469 [==============================] - 40s 85ms/step - d_loss: 0.2755 - g_loss: 6.2294
Epoch 6/20
469/469 [==============================] - 40s 85ms/step - d_loss: 0.2732 - g_loss: 5.9429
Epoch 7/20
469/469 [==============================] - 40s 85ms/step - d_loss: 0.2728 - g_loss: 5.6715
Epoch 8/20
469/469 [==============================] - 40s 85ms/step - d_loss: 0.2708 - g_loss: 5.4097
Epoch 9/20
469/469 [==============================] - 40s 85ms/step - d_loss: 0.2704 - g_loss: 5.1352
Epoch 10/20
469/469 [==============================] - 40s 85ms/step - d_loss: 0.2701 - g_loss: 4.8858
Epoch 11/20
469/469 [==============================] - 40s 85ms/step - d_loss: 0.2694 - g_loss: 4.6344
Epoch 12/20
469/469 [==============================] - 40s 85ms/step - d_loss: 0.2691 - g_loss: 4.3753
Epoch 13/20
469/469 [==============================] - 40s 86ms/step - d_loss: 0.2694 - g_loss: 4.1332
Epoch 14/20
469/469 [==============================] - 40s 86ms/step - d_loss: 0.2687 - g_loss: 3.9182
Epoch 15/20
469/469 [==============================] - 40s 86ms/step - d_loss: 0.2679 - g_loss: 3.7484
Epoch 16/20
469/469 [==============================] - 40s 86ms/step - d_loss: 0.2681 - g_loss: 3.5493
Epoch 17/20
469/469 [==============================] - 40s 86ms/step - d_loss: 0.2683 - g_loss: 3.3835
Epoch 18/20
469/469 [==============================] - 40s 86ms/step - d_loss: 0.2675 - g_loss: 3.2472
Epoch 19/20
469/469 [==============================] - 40s 86ms/step - d_loss: 0.2679 - g_loss: 3.1084
Epoch 20/20
469/469 [==============================] - 40s 86ms/step - d_loss: 0.3128 - g_loss: 2.6171
```

4.4 Review Performance

```
plt.suptitle('Loss')
plt.plot(hist.history['d_loss'], label='d_loss')
plt.plot(hist.history['g_loss'], label='g_loss')
plt.legend()
plt.show()
```



5. Test Out the Generator

5.1 Generate Images

↳ 3 cells hidden

5.2 Save the Model

```
generator.save('generator.h5')
discriminator.save('discriminator.h5')
```

```
Start coding or generate with AI.
```