Dependencies

```
import numpy as np
import tensorflow as tf
import tensorflow_datasets as tfds
import matplotlib.pyplot as plt
gpus = tf.config.experimental.list_physical_devices('GPU')
for gpu in gpus:
    tf.config.experimental.set_memory_growth(gpu, True)
```

Loading the data

```
data = tfds.load('fashion_mnist', split='train')
```

```
Downloading and preparing dataset 29.45 MiB (download: 29.45 MiB, generated: 36.42 MiB,
Dl Completed...: 0 url [00:00, ? url/s]
Dl Size...: 0 MiB [00:00, ? MiB/s]
Extraction completed...: 0 file [00:00, ? file/s]
Generating splits...:   0%|          | 0/2 [00:00<?, ? splits/s]
Generating train examples...:   0%|          | 0/60000 [00:00<?, ? examples/s]
Shuffling fashion_mnist-train.tfrecord...:   0%|          | 0/60000 [00:00<?, ?
examples/s]
Generating test examples...:   0%|          | 0/10000 [00:00<?, ? examples/s]
Shuffling fashion_mnist-test.tfrecord...:   0%|          | 0/10000 [00:00<?, ?
```

```
data.as_numpy_iterator().next().keys()
```

```
dict_keys(['image', 'label'])
```

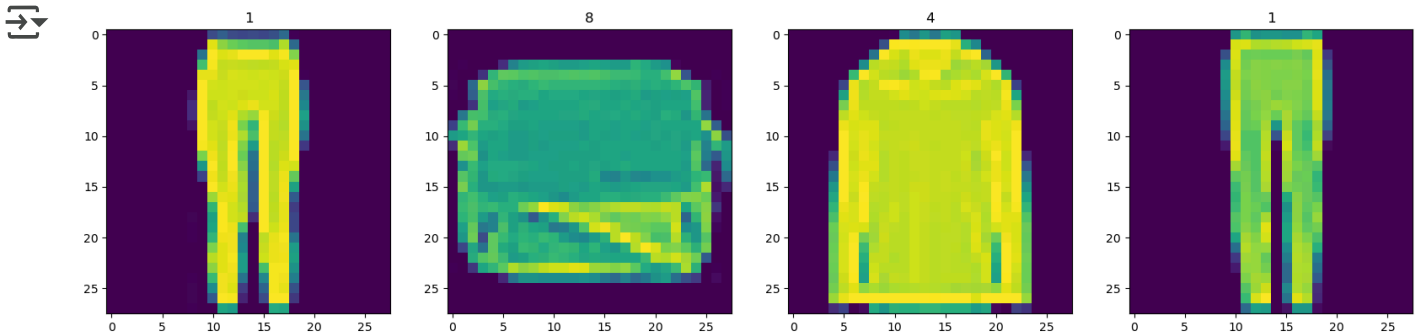
Data Visualization

```
dataiterator = data.as_numpy_iterator()
```

```
dataiterator.next()
```

```
               [  1],
               [  0],
               [  0],
               [  0],
               [  0],
               [  0],
               [  0],
               [  0],
               [  0],
               [  0],
               [  0],
               [  0],
               [153],
               [ 78],
               [106],
               [ 37],
               [  0],
               [  0],
               [  0]],

              [[  0],
               [  0],
               [  0],
               [ 18],
               [ 61],
               [ 41],
               [103],
               [  0],
               [  0],
               [  0],
               [  0],
               [  0],
               [  0],
               [  0],
               [  0],
               [  0],
               [  0],
               [  0],
               [  0],
               [  0],
               [106],
               [ 47],
               [ 69],
               [ 23],
               [  0],
               [  0],
               [  0]]], dtype=uint8),
      'label': 2}
```

```
fig, ax =plt.subplots(ncols= 4, figsize=(20, 20))
for idx in range(4):
    sample = dataiterator.next()
    ax[idx].imshow(np.squeeze(sample['image']))
    ax[idx].title.set_text(sample['label'])
```



preprocessing

```
def scale_images(data):
    image = data['image']
    return image / 255
```

Preparing data for tensorflow

```
data = data.map(scale_images)
data = data.cache()
data = data.shuffle(60000)
data = data.batch(128)
data = data.prefetch(64)
```

```
print(data)
```

```
<PrefetchDataset element_spec=TensorSpec(shape=(None, 28, 28, 1), dtype=tf.float32, name
```

```
data.as_numpy_iterator().next().shape
```

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

Neural Network

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, Dense, Flatten, Reshape, LeakyReLU, Dropout, UpS
```

Generator

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

    model.add(Dense(7*7*128, input_dim= 128))
    model.add(LeakyReLU(0.2))
    model.add(Reshape((7,7,128)))

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

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

    model.add(Conv2D(128, 4, padding= 'same'))
    model.add(LeakyReLU(0.2))

    model.add(Conv2D(128, 4, padding= 'same'))
    model.add(LeakyReLU(0.2))

    model.add(Conv2D(1, 4, padding= 'same', activation= 'sigmoid'))

    return model
```

```
generator = build_generator()
```

```
generator.summary()
```

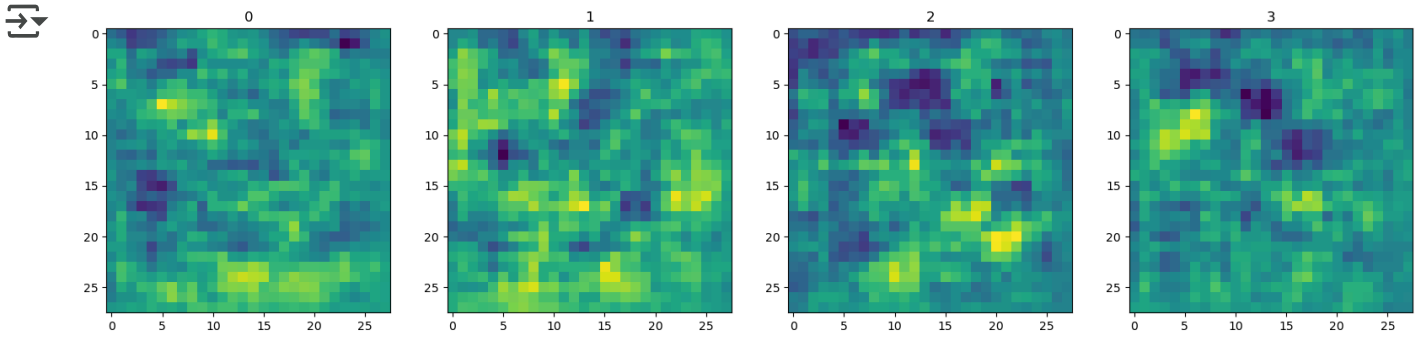
```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 6272)              809088

 leaky_re_lu (LeakyReLU)     (None, 6272)              0

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

 up_sampling2d (UpSampling2D  (None, 14, 14, 128)      0
 )

 conv2d (Conv2D)             (None, 14, 14, 128)       409728
```

```
 leaky_re_lu_1 (LeakyReLU)   (None, 14, 14, 128)       0

 up_sampling2d_1 (UpSampling  (None, 28, 28, 128)      0
 2D)

 conv2d_1 (Conv2D)           (None, 28, 28, 128)       409728

 leaky_re_lu_2 (LeakyReLU)   (None, 28, 28, 128)       0

 conv2d_2 (Conv2D)           (None, 28, 28, 128)       262272

 leaky_re_lu_3 (LeakyReLU)   (None, 28, 28, 128)       0

 conv2d_3 (Conv2D)           (None, 28, 28, 128)       262272

 leaky_re_lu_4 (LeakyReLU)   (None, 28, 28, 128)       0

 conv2d_4 (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))
```

```
1/1 [==============================] - 7s 7s/step
```

```
fig, ax = plt.subplots(ncols= 4, figsize=(20, 20))

for idx, img in enumerate(img):
    ax[idx].imshow(np.squeeze(img))
    ax[idx].title.set_text(idx)
```



discriminator

```
def build_discriminator():

    model = Sequential()

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

    model.add(Conv2D(64, 5))
    model.add(LeakyReLU(0.2))
    model.add(Dropout(0.4))

    model.add(Conv2D(128, 5))
    model.add(LeakyReLU(0.2))
    model.add(Dropout(0.4))

    model.add(Conv2D(256, 5))
    model.add(LeakyReLU(0.2))
    model.add(Dropout(0.4))

    model.add(Flatten())
    model.add(Dropout(0.4))
    model.add(Dense(1, activation= 'sigmoid'))

    return model
```

```
discriminator = build_discriminator()
```

```
discriminator.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_5 (Conv2D)           (None, 24, 24, 32)        832

 leaky_re_lu_5 (LeakyReLU)   (None, 24, 24, 32)        0

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

 conv2d_6 (Conv2D)           (None, 20, 20, 64)        51264

 leaky_re_lu_6 (LeakyReLU)   (None, 20, 20, 64)        0

 dropout_1 (Dropout)         (None, 20, 20, 64)        0

 conv2d_7 (Conv2D)           (None, 16, 16, 128)       204928

 leaky_re_lu_7 (LeakyReLU)   (None, 16, 16, 128)       0

 dropout_2 (Dropout)         (None, 16, 16, 128)       0

 conv2d_8 (Conv2D)           (None, 12, 12, 256)       819456

 leaky_re_lu_8 (LeakyReLU)   (None, 12, 12, 256)       0

 dropout_3 (Dropout)         (None, 12, 12, 256)       0

 flatten (Flatten)           (None, 36864)             0

 dropout_4 (Dropout)         (None, 36864)             0

 dense_1 (Dense)             (None, 1)                 36865

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

```
img.shape
```

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

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

```
1/1 [==============================] - 0s 20ms/step
```

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

```
1/1 [==============================] - 0s 158ms/step
array([[0.49692342],
       [0.49718642],
       [0.49706194],
       [0.49711314]], dtype=float32)
```

Custom Training Loop

Losses and optimizers

```
from tensorflow.keras.optimizers import Adam
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()
```

subclasses Model

```
from tensorflow.keras.models import Model
class imageGAN(Model):
    def __init__(self, generator, discriminator, *args, **kwargs):
        super().__init__(*args, **kwargs)

        self.generator = generator
        self.discriminator = discriminator

    def compile(self, g_opt, d_opt, g_loss, d_loss, *args, **kwargs):

        super().compile(*args, **kwargs)

        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):
        real_images = batch
        fake_images = self.generator(tf.random.normal((128, 128, 1)), training= False)

        #Training Discriminator
        with tf.GradientTape() as d_tape:

            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)

            y_realfake = tf.concat([tf.zeros_like(yhat_real), tf.ones_like(yhat_fake)], axis

            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)

            total_d_loss = self.d_loss(y_realfake, yhat_realfake)

            d_grad = d_tape.gradient(total_d_loss, self.discriminator.trainable_variables)
            self.d_opt.apply_gradients(zip(d_grad, self.discriminator.trainable_variables))

        #Training Generator
        with tf.GradientTape() as g_tape:

            gen_images = self.generator(tf.random.normal((128, 128, 1)), training= True)

            predicted_labels = self.discriminator(gen_images, training= False)

            total_g_loss = self.g_loss(tf.zeros_like(predicted_labels), predicted_labels)

        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}
```

instantiating the imageGAN subclassed model

```
imggan = imageGAN(generator, discriminator)
```

compiling the model

```
imggan.compile(g_opt, d_opt, g_loss, d_loss)
```

Building 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(f'generated_img_{epoch}_{i}.png'))
```

Training

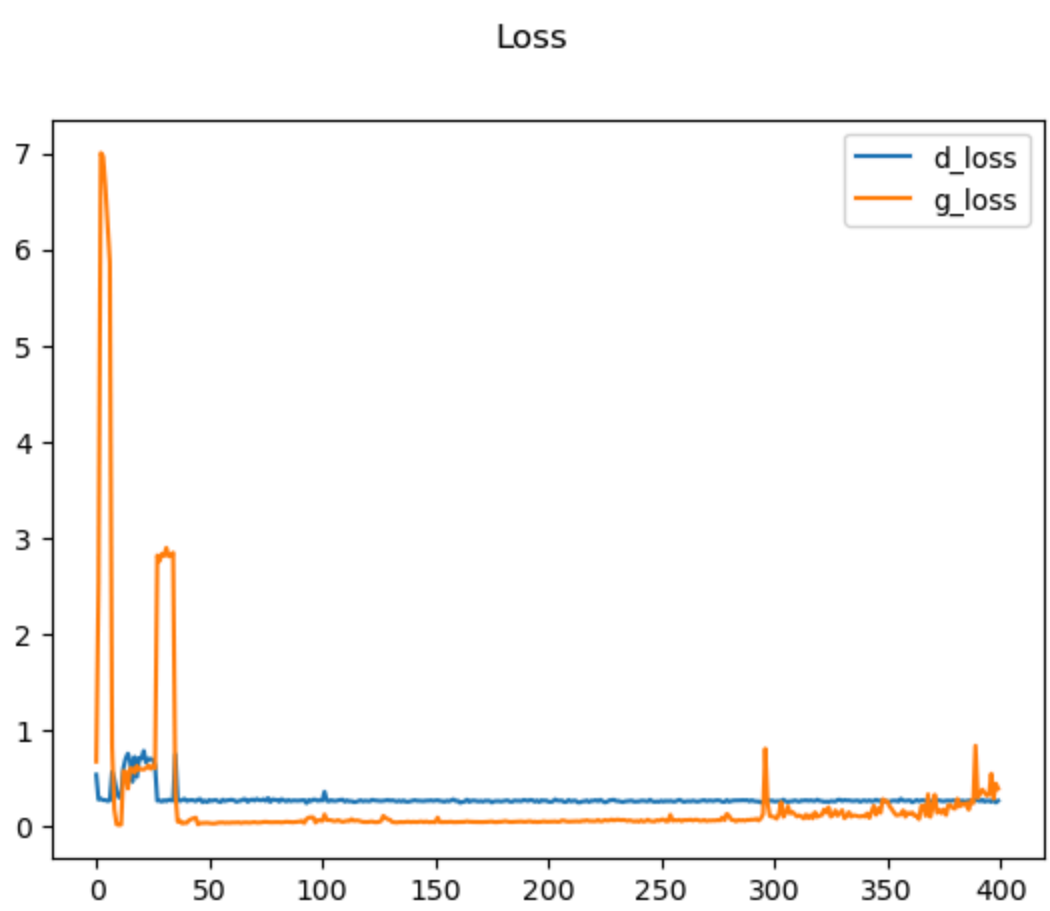
```
%%time
hist = imggan.fit(data, epochs= 400, callbacks=[ModelMonitor()])
```

```
Epoch 1/400
469/469 [==============================] - 73s 146ms/step - d_loss : 0.6111 - g_loss :
Epoch 2/400
469/469 [==============================] - 72s 153ms/step - d_loss : 0.3994 - g_loss :
Epoch 3/400
469/469 [==============================] - 72s 153ms/step - d_loss : 0.3042 - g_loss :
Epoch 4/400
469/469 [==============================] - 72s 153ms/step - d_loss : 0.2877 - g_loss :
Epoch 5/400
469/469 [==============================] - 71s 152ms/step - d_loss : 0.2801 - g_loss :
Epoch 6/400
469/469 [==============================] - 71s 152ms/step - d_loss : 0.2758 - g_loss :
Epoch 7/400
469/469 [==============================] - 71s 152ms/step - d_loss : 0.2754 - g_loss :
Epoch 8/400
469/469 [==============================] - 72s 153ms/step - d_loss : 0.3796 - g_loss :
Epoch 9/400
469/469 [==============================] - 72s 153ms/step - d_loss : 0.6565 - g_loss :
Epoch 10/400
469/469 [==============================] - 72s 153ms/step - d_loss : 0.4821 - g_loss :
Epoch 11/400
469/469 [==============================] - 72s 154ms/step - d_loss : 0.3381 - g_loss :
Epoch 12/400
469/469 [==============================] - 72s 154ms/step - d_loss : 0.3236 - g_loss :
Epoch 13/400
469/469 [==============================] - 72s 154ms/step - d_loss : 0.5460 - g_loss :
Epoch 14/400
469/469 [==============================] - 72s 153ms/step - d_loss : 0.6809 - g_loss :
Epoch 15/400
469/469 [==============================] - 72s 154ms/step - d_loss : 0.6546 - g_loss :
Epoch 16/400
469/469 [==============================] - 72s 153ms/step - d_loss : 0.6825 - g_loss :
Epoch 17/400
469/469 [==============================] - 72s 154ms/step - d_loss : 0.6719 - g_loss :
Epoch 18/400
469/469 [==============================] - 72s 153ms/step - d_loss : 0.6844 - g_loss :
```

```
Epoch 19/400
469/469 [==============================] - 72s 153ms/step - d_loss : 0.6800 - g_loss :
Epoch 20/400
469/469 [==============================] - 72s 153ms/step - d_loss : 0.6764 - g_loss :
Epoch 21/400
469/469 [==============================] - 72s 153ms/step - d_loss : 0.6821 - g_loss :
Epoch 22/400
469/469 [==============================] - 72s 154ms/step - d_loss : 0.6816 - g_loss :
Epoch 23/400
469/469 [==============================] - 72s 154ms/step - d_loss : 0.6913 - g_loss :
Epoch 24/400
469/469 [==============================] - 72s 154ms/step - d_loss : 0.6966 - g_loss :
Epoch 25/400
469/469 [==============================] - 72s 153ms/step - d_loss : 0.6856 - g_loss :
Epoch 26/400
469/469 [==============================] - 72s 154ms/step - d_loss : 0.6720 - g_loss :
Epoch 27/400
469/469 [==============================] - 72s 154ms/step - d_loss : 0.6750 - g_loss :
Epoch 28/400
469/469 [==============================] - 72s 153ms/step - d_loss : 0.4737 - g_loss :
Epoch 29/400
```

Performance Review

```
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()
```

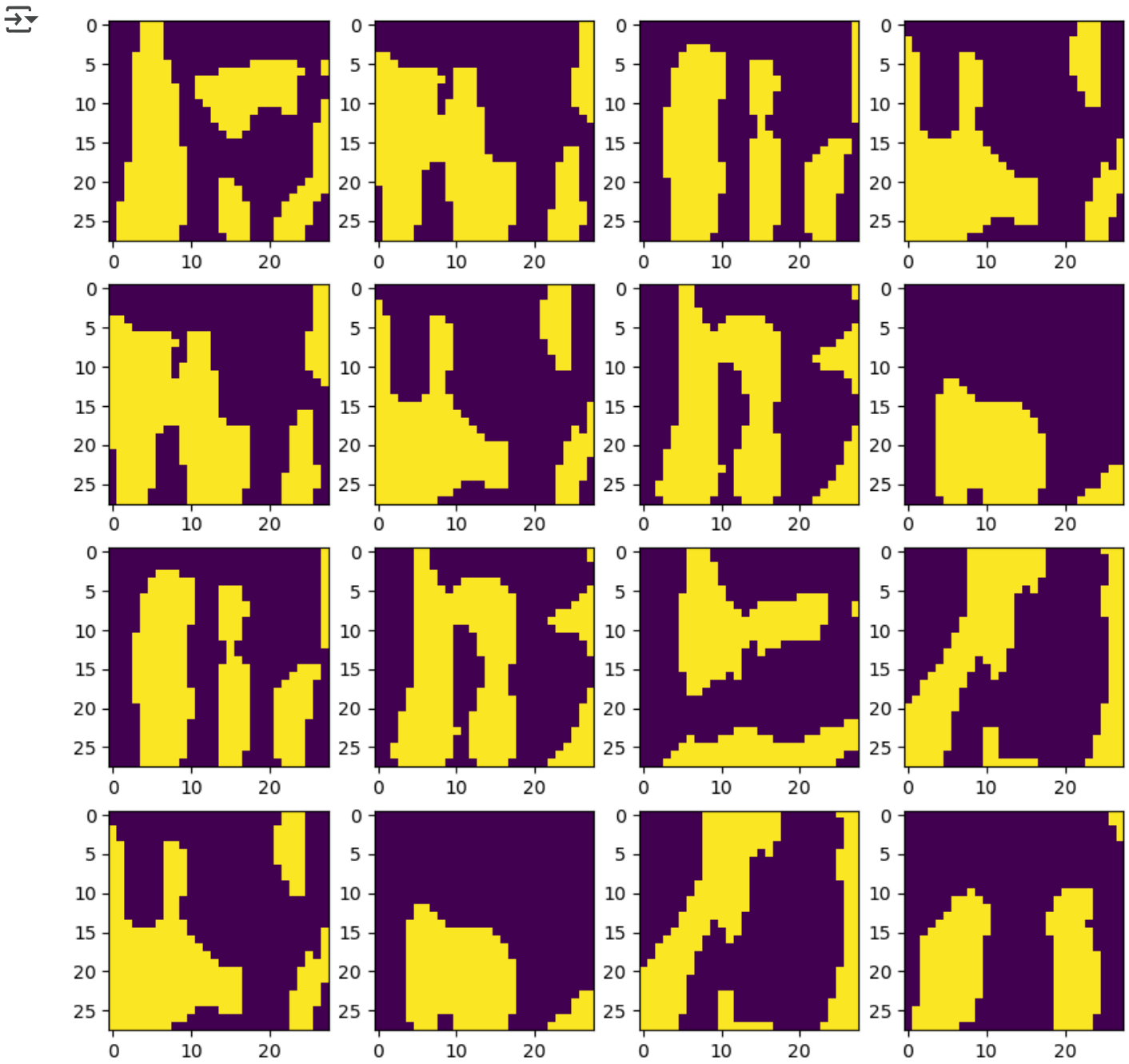


Generating images using trained model

```
imgs = generator.predict(tf.random.normal((16, 128, 1)))
```

```
1/1 [==============================] - 0s 201ms/step
```

```
fig, ax = plt.subplots(ncols= 4, nrows=4, figsize=(10,10))
for r in range(4):
    for c in range(4):
        ax[r][c].imshow(imgs[(r+1)*(c+1)-1])
```



Saving the model

```
generator.save_weights('generator_weights.h5')
```

Loading the saved model

```
from tensorflow.keras.models import load_model
loaded_model = build_generator()
```

```
loaded_model.compile()
```

```
loaded_model.load_weights('generator_weights.h5')
```

```
loaded_model_img = loaded_model.predict(tf.random.normal((16, 128, 1)))
fig, ax = plt.subplots(ncols= 4, nrows=4, figsize=(10,10))
for r in range(4):
    for c in range(4):
        ax[r][c].imshow(loaded_model_img[(r+1)*(c+1)-1])
```

```
1/1 [==============================] - 0s 121ms/step
```