Code & Output:

```
import tensorflow as tf
from tensorflow.keras import datasets, layers, models
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt
# Load and preprocess the MNIST dataset
(train_images, train_labels), (test_images, test_labels) = datasets.mnist.load_data()
train_images, test_images = train_images / 255.0, test_images / 255.0
```

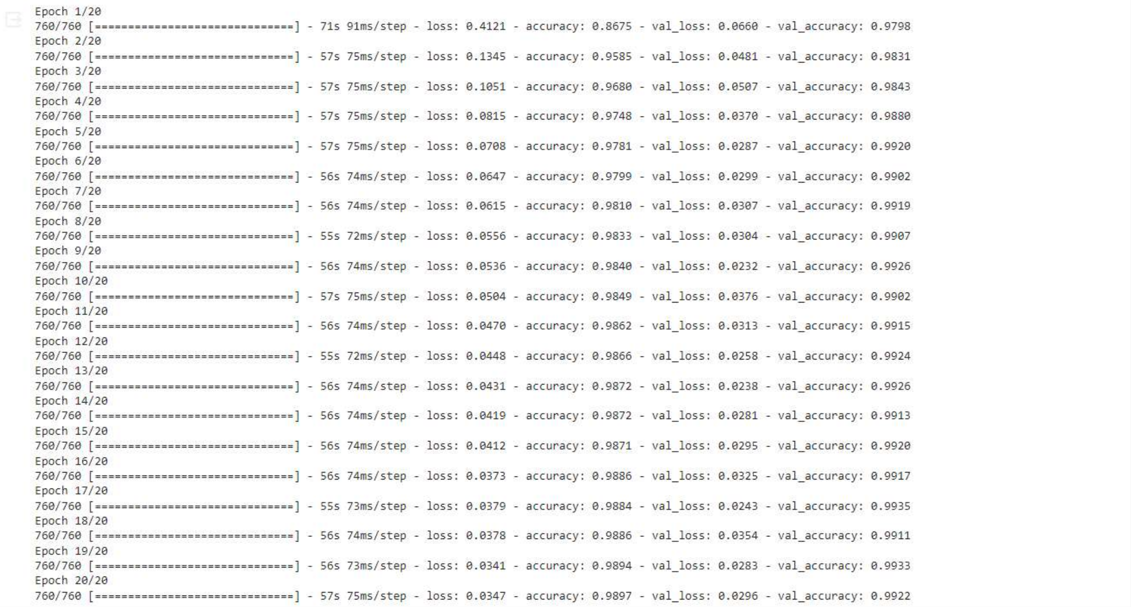
```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.npz
11490434/11490434 [==============================] - 0s 0us/step
```

```
# Add channel dimension to the images
train_images = train_images.reshape((60000, 28, 28, 1))
test_images = test_images.reshape((10000, 28, 28, 1))
# Split the dataset into training and validation sets
train_images, val_images, train_labels, val_labels = train_test_split( train_images,
train_labels, test_size=0.1, random_state=42)

# Data augmentation for training images
datagen = ImageDataGenerator(rotation_range=10, zoom_range=0.1, width_shift_range=0.1,
height_shift_range=0.1)
datagen.fit(train_images)
from keras import models, layers

# Create a CNN model with hyperparameter tuning and regularization
model = models.Sequential()
model.add(layers.Conv2D(32, (3, 3), activation='relu', input_shape=(28, 28, 1)))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(128, (3, 3), activation='relu'))
model.add(layers.Flatten())
model.add(layers.Dropout(0.5))
model.add(layers.Dense(128, activation='relu'))
model.add(layers.Dense(10, activation='softmax'))

# Compile the model
model.compile(optimizer=Adam(learning_rate=0.001),
loss='sparse_categorical_crossentropy', metrics=['accuracy'])
# Train the model with data augmentation
history = model.fit(datagen.flow(train_images, train_labels, batch_size=64),epochs=20,
validation_data=(val_images, val_labels))
```

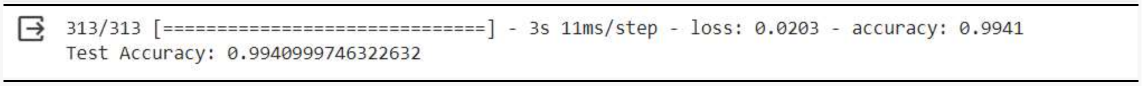
```
# Evaluate the model on the test set
test_loss, test_acc = model.evaluate(test_images, test_labels)
print(f"Test Accuracy: {test_acc}")
```

```
# Plot training history
plt.plot(history.history['accuracy'], label='Train Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()
plt.show()
```

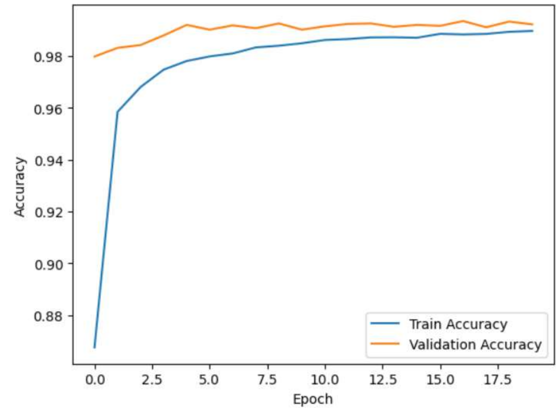
Output:-



```
Epoch 1/20
760/760 [==============================] - 71s 91ms/step - loss: 0.4121 - accuracy: 0.8675 - val_loss: 0.0660 - val_accuracy: 0.9798
Epoch 2/20
760/760 [==============================] - 57s 75ms/step - loss: 0.1345 - accuracy: 0.9585 - val_loss: 0.0481 - val_accuracy: 0.9831
Epoch 3/20
760/760 [==============================] - 57s 75ms/step - loss: 0.1051 - accuracy: 0.9680 - val_loss: 0.0507 - val_accuracy: 0.9843
Epoch 4/20
760/760 [==============================] - 57s 75ms/step - loss: 0.0815 - accuracy: 0.9748 - val_loss: 0.0370 - val_accuracy: 0.9880
Epoch 5/20
760/760 [==============================] - 57s 75ms/step - loss: 0.0708 - accuracy: 0.9781 - val_loss: 0.0287 - val_accuracy: 0.9920
Epoch 6/20
760/760 [==============================] - 56s 74ms/step - loss: 0.0647 - accuracy: 0.9799 - val_loss: 0.0299 - val_accuracy: 0.9902
Epoch 7/20
760/760 [==============================] - 56s 74ms/step - loss: 0.0615 - accuracy: 0.9810 - val_loss: 0.0307 - val_accuracy: 0.9919
Epoch 8/20
760/760 [==============================] - 55s 72ms/step - loss: 0.0556 - accuracy: 0.9833 - val_loss: 0.0304 - val_accuracy: 0.9907
Epoch 9/20
760/760 [==============================] - 56s 74ms/step - loss: 0.0536 - accuracy: 0.9840 - val_loss: 0.0232 - val_accuracy: 0.9926
Epoch 10/20
760/760 [==============================] - 57s 75ms/step - loss: 0.0504 - accuracy: 0.9849 - val_loss: 0.0376 - val_accuracy: 0.9902
Epoch 11/20
760/760 [==============================] - 56s 74ms/step - loss: 0.0470 - accuracy: 0.9862 - val_loss: 0.0313 - val_accuracy: 0.9915
Epoch 12/20
760/760 [==============================] - 55s 72ms/step - loss: 0.0448 - accuracy: 0.9866 - val_loss: 0.0258 - val_accuracy: 0.9924
Epoch 13/20
760/760 [==============================] - 56s 74ms/step - loss: 0.0431 - accuracy: 0.9872 - val_loss: 0.0238 - val_accuracy: 0.9926
Epoch 14/20
760/760 [==============================] - 56s 74ms/step - loss: 0.0419 - accuracy: 0.9872 - val_loss: 0.0281 - val_accuracy: 0.9913
Epoch 15/20
760/760 [==============================] - 56s 74ms/step - loss: 0.0412 - accuracy: 0.9871 - val_loss: 0.0295 - val_accuracy: 0.9920
Epoch 16/20
760/760 [==============================] - 56s 74ms/step - loss: 0.0373 - accuracy: 0.9886 - val_loss: 0.0325 - val_accuracy: 0.9917
Epoch 17/20
760/760 [==============================] - 55s 73ms/step - loss: 0.0379 - accuracy: 0.9884 - val_loss: 0.0243 - val_accuracy: 0.9935
Epoch 18/20
760/760 [==============================] - 56s 74ms/step - loss: 0.0378 - accuracy: 0.9886 - val_loss: 0.0354 - val_accuracy: 0.9911
Epoch 19/20
760/760 [==============================] - 56s 73ms/step - loss: 0.0341 - accuracy: 0.9894 - val_loss: 0.0283 - val_accuracy: 0.9933
Epoch 20/20
760/760 [==============================] - 57s 75ms/step - loss: 0.0347 - accuracy: 0.9897 - val_loss: 0.0296 - val_accuracy: 0.9922
```



```
313/313 [==============================] - 3s 11ms/step - loss: 0.0203 - accuracy: 0.9941
Test Accuracy: 0.9940999746322632
```



Code & Output:

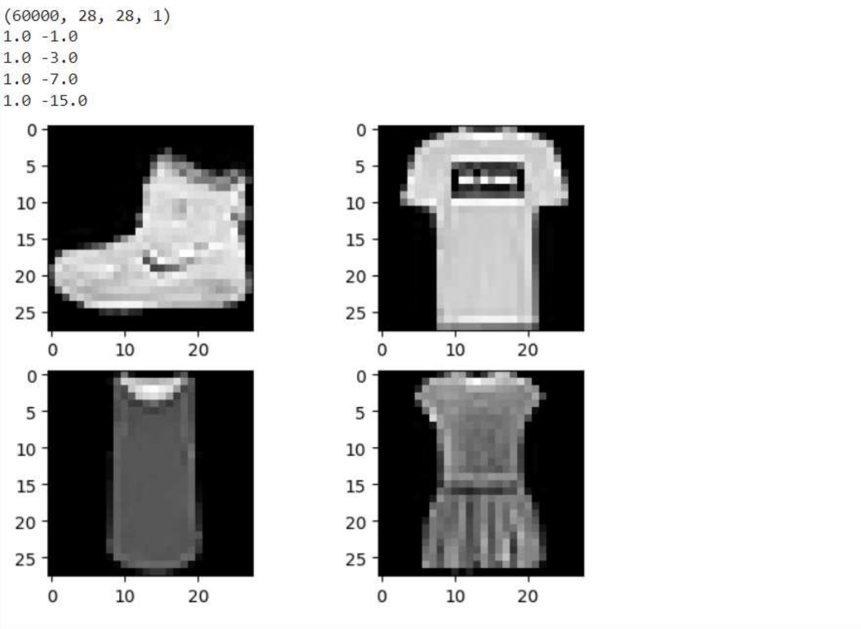
```
from keras.models import Model
from keras.layers import Input,Dense
import  numpy as np
import pandas as pd
import keras.backend as K
import matplotlib.pyplot as plt
from keras import preprocessing
from keras.models import Sequential
from keras.layers import
Conv2D,Dropout,Dense,Flatten,Conv2DTranspose,BatchNormalization,LeakyReLU,Reshape
import tensorflow as tf
from keras.layers import *
from keras.datasets import fashion_mnist
(train_x, train_y), (val_x, val_y) = fashion_mnist.load_data()
train_x = train_x/255.
val_x = val_x/255.
train_x=train_x.reshape(-1,28,28,1)
print(train_x.shape)
#train_x = train_x.reshape(-1, 784)
#val_x = val_x.reshape(-1, 784)
fig,axe=plt.subplots(2,2)
idx = 0
for i in range(2):
  for j in range(2):
    axe[i,j].imshow(train_x[idx].reshape(28,28),cmap='gray')
    idx+=1
    train_x = train_x*2 - 1
    print(train_x.max(),train_x.min())
generator = Sequential()
generator.add(Dense(512,input_shape=[100]))
generator.add(LeakyReLU(alpha=0.2))
generator.add(BatchNormalization(momentum=0.8))
generator.add(Dense(256))
generator.add(LeakyReLU(alpha=0.2))
generator.add(BatchNormalization(momentum=0.8))
generator.add(Dense(128))
generator.add(LeakyReLU(alpha=0.2))
generator.add(BatchNormalization(momentum=0.8))
generator.add(Dense(784))
generator.add(Reshape([28,28,1]))
generator.summary()
```

```
discriminator = Sequential()
discriminator.add(Dense(1,input_shape=[28,28,1]))
discriminator.add(Flatten())
discriminator.add(Dense(256))
discriminator.add(LeakyReLU(alpha=0.2))
discriminator.add(Dropout(0.5))
discriminator.add(Dense(128))
discriminator.add(LeakyReLU(alpha=0.2))
discriminator.add(Dropout(0.5))
discriminator.add(Dense(64))
discriminator.add(LeakyReLU(alpha=0.2))
discriminator.add(Dropout(0.5))
discriminator.add(Dense(1,activation='sigmoid'))
discriminator.summary()
```

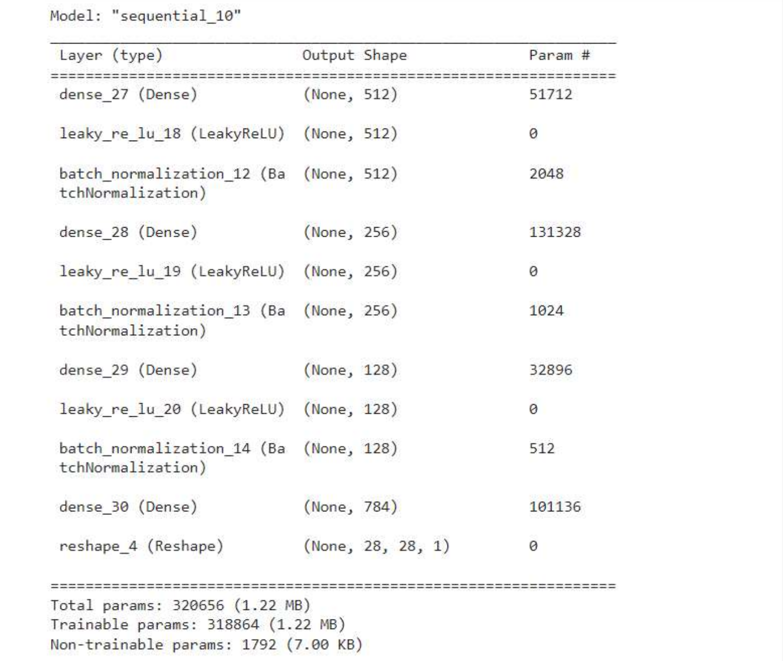
```
GAN =Sequential([generator,discriminator])
discriminator.compile(optimizer='adam',loss='binary_crossentropy')
discriminator.trainable = False
GAN.compile(optimizer='adam',loss='binary_crossentropy')
GAN.summary()
epochs = 30
batch_size = 100
noise_shape=100
with tf.device('/gpu:0'):
  for epoch in range(epochs):
    print(f"Currently on Epoch {epoch+1}")
    for i in range(train_x.shape[0]//batch_size):
      if (i+1)%100 == 0:
        print(f"\tCurrently on batch number {i+1} of {train_x.shape[0]//batch_size}")
      noise=np.random.normal(size=[batch_size,noise_shape])
      gen_image = generator.predict_on_batch(noise)
      train_dataset = train_x[i*batch_size:(i+1)*batch_size]
#training discriminator on real images
train_label=np.ones(shape=(batch_size,1))
#train_label=np.ones((batch_size, 1))
discriminator.trainable = True
#train_dataset=train_x[idx]
d_loss_real=discriminator.train_on_batch(train_dataset,train_label) #training discriminator
on fake images train_label=np.zeros(shape=(batch_size,1))
d_loss_fake=discriminator.train_on_batch(gen_image,train_label)
#training generator
noise=np.random.normal(size=[batch_size,noise_shape])
train_label=np.ones(shape=(batch_size,1))
discriminator.trainable = False
d_g_loss_batch =GAN.train_on_batch(noise, train_label)
#plotting generated images at the start and then after every 10 epoch if epoch % 10 == 0:
samples = 10
x_fake = generator.predict(np.random.normal(loc=0, scale=1, size=(samples, 100)))
for k in range(samples):
  plt.subplot(2, 5, k+1)
  plt.imshow(x_fake[k].reshape(28, 28), cmap='gray')
  plt.xticks([])
  plt.yticks([])
  plt.tight_layout()
  plt.show()
```

```
print('Training is complete')
noise=np.random.normal(size=[10,noise_shape])
gen_image = generator.predict(noise)
plt.imshow(noise)
plt.title('How the noise looks')
fig,axe=plt.subplots(2,5)
fig.suptitle('Generated Images from Noise using GANs')
idx=0
for i in range(2):
  for j in range(5):
    axe[i,j].imshow(gen_image[idx].reshape(28,28),cmap='gray')
```

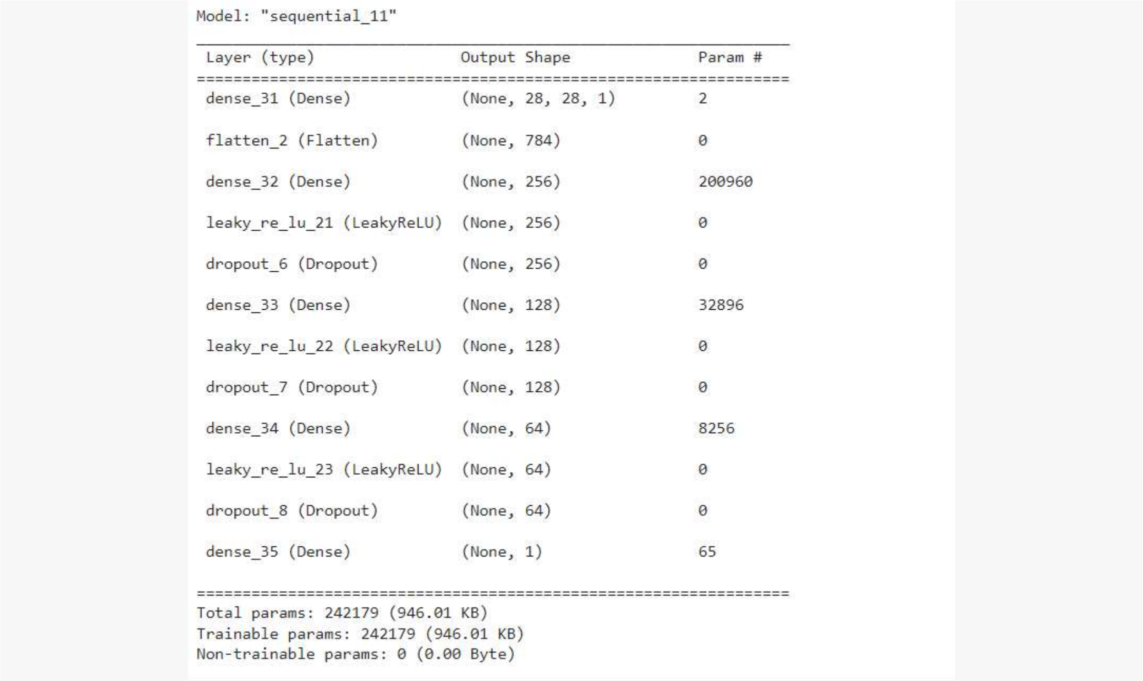
Output:-



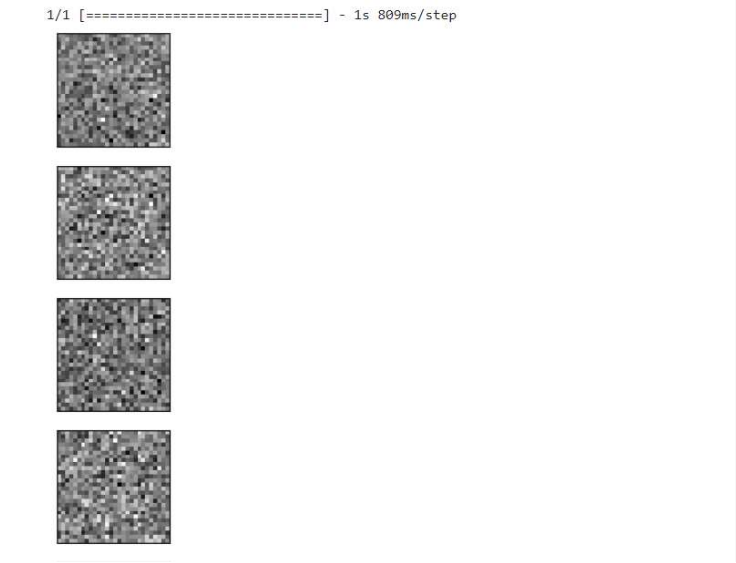
```
(60000, 28, 28, 1)
1.0 -1.0
1.0 -3.0
1.0 -7.0
1.0 -15.0
```



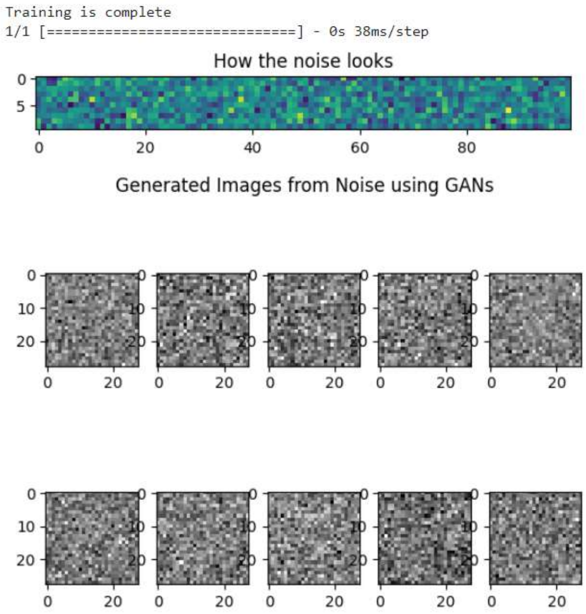
```
Model: "sequential_10"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_27 (Dense)            (None, 512)               51712

 leaky_re_lu_18 (LeakyReLU)  (None, 512)               0

 batch_normalization_12 (Ba  (None, 512)               2048
 tchNormalization)

 dense_28 (Dense)            (None, 256)               131328

 leaky_re_lu_19 (LeakyReLU)  (None, 256)               0

 batch_normalization_13 (Ba  (None, 256)               1024
 tchNormalization)

 dense_29 (Dense)            (None, 128)               32896

 leaky_re_lu_20 (LeakyReLU)  (None, 128)               0

 batch_normalization_14 (Ba  (None, 128)               512
 tchNormalization)

 dense_30 (Dense)            (None, 784)               101136

 reshape_4 (Reshape)         (None, 28, 28, 1)         0

=================================================================
Total params: 320656 (1.22 MB)
Trainable params: 318864 (1.22 MB)
Non-trainable params: 1792 (7.00 KB)
```



```
Model: "sequential_11"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_31 (Dense)            (None, 28, 28, 1)         2

 flatten_2 (Flatten)         (None, 784)               0

 dense_32 (Dense)            (None, 256)               200960

 leaky_re_lu_21 (LeakyReLU)  (None, 256)               0

 dropout_6 (Dropout)         (None, 256)               0

 dense_33 (Dense)            (None, 128)               32896

 leaky_re_lu_22 (LeakyReLU)  (None, 128)               0

 dropout_7 (Dropout)         (None, 128)               0

 dense_34 (Dense)            (None, 64)                8256

 leaky_re_lu_23 (LeakyReLU)  (None, 64)                0

 dropout_8 (Dropout)         (None, 64)                0

 dense_35 (Dense)            (None, 1)                 65

=================================================================
Total params: 242179 (946.01 KB)
Trainable params: 242179 (946.01 KB)
Non-trainable params: 0 (0.00 Byte)
```



```
1/1 [==============================] - 1s 809ms/step
```



```
Training is complete
1/1 [==============================] - 0s 38ms/step
```