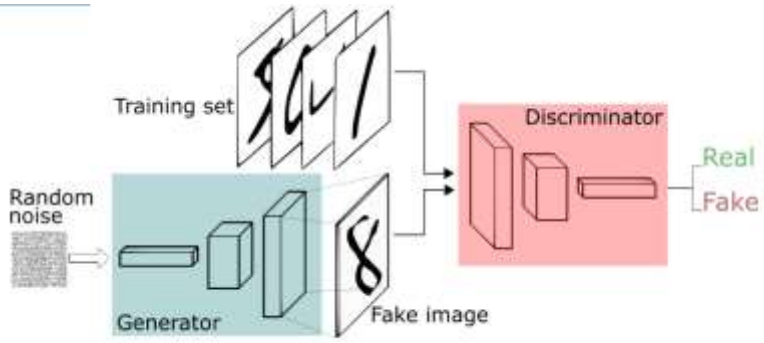
2602191301 - Allin Setiawan

Jelaskan cara kerja dari arsitektur tersebut yang dideskripsikan dalam gambar dibawah ini:




**Random noise** bakal masuk kedalam **Generator** yang berfungsi untuk menghasilkan data baru yang mirip dengan data aslinya lalu data baru yang dihasilkan generator akan dimasukkan ke dalam **Discriminator** yang berfungsi untuk membedakan antara data asli dan data palsu yang dihasilkan oleh generator tadi. Dari proses diatas saya bisa simpulkan bahwa proses GAN ada dua langkah yaitu pada **Generator dan Discriminator**. proses ini akan terus berulang-ulang dan dengan seiringnya waktu si generator akan semakin jago menghasilkan data yang mirip dengan data asli dan discriminator semakin jago membedakan antara data asli dan data palsu.

# Import Dataset Fashion Mnist

```
!unzip '/content/t10k-images-idx3-ubyte.zip'
!unzip '/content/train-images-idx3-ubyte.zip'
!unzip '/content/train-labels-idx1-ubyte.zip'
!unzip '/content/t10k-labels-idx1-ubyte.zip'
```

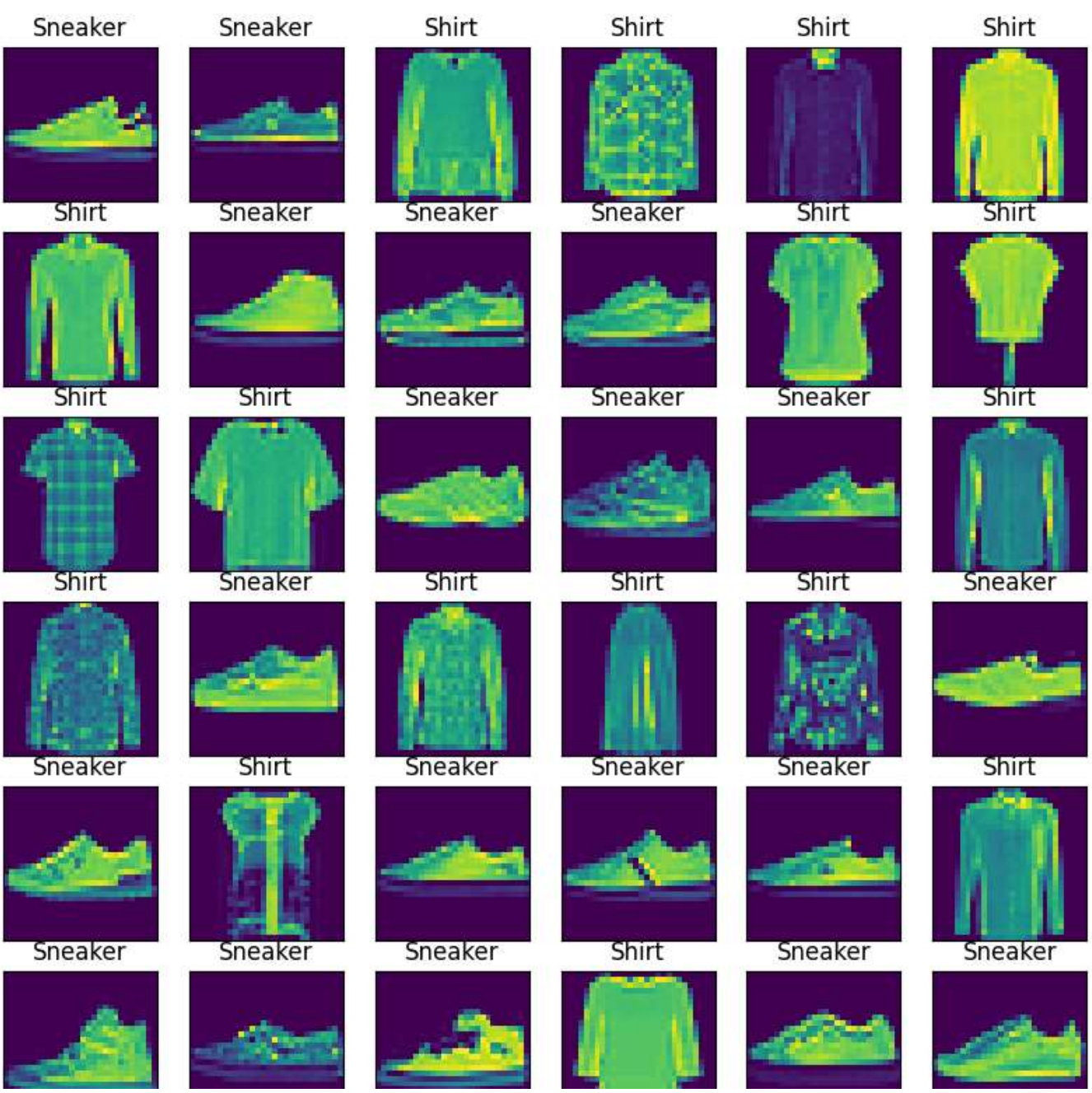
```
Archive:  /content/t10k-images-idx3-ubyte.zip
  inflating: t10k-images-idx3-ubyte
Archive:  /content/train-images-idx3-ubyte.zip
  inflating: train-images-idx3-ubyte
Archive:  /content/train-labels-idx1-ubyte.zip
  inflating: train-labels-idx1-ubyte
Archive:  /content/t10k-labels-idx1-ubyte.zip
  inflating: t10k-labels-idx1-ubyte
```

```
!pip install idx2numpy
import idx2numpy

x_train = idx2numpy.convert_from_file('train-images-idx3-ubyte')
y_train = idx2numpy.convert_from_file('train-labels-idx1-ubyte')

x_test = idx2numpy.convert_from_file('t10k-images-idx3-ubyte')
y_test = idx2numpy.convert_from_file('t10k-labels-idx1-ubyte')
```

```
Collecting idx2numpy
  Downloading idx2numpy-1.2.3.tar.gz (6.8 kB)
  Preparing metadata (setup.py) ... done
Requirement already satisfied: numpy in /usr/local/lib/python3.10/dist-packages (from id
Requirement already satisfied: six in /usr/local/lib/python3.10/dist-packages (from idx2
Building wheels for collected packages: idx2numpy
  Building wheel for idx2numpy (setup.py) ... done
  Created wheel for idx2numpy: filename=idx2numpy-1.2.3-py3-none-any.whl size=7904 sha25
  Stored in directory: /root/.cache/pip/wheels/e0/f4/e7/643fc5f932ec2ff92997f43f007660fe
Successfully built idx2numpy
Installing collected packages: idx2numpy
Successfully installed idx2numpy-1.2.3
```

# Preprocessing and Scaling Dataset

```
import matplotlib.pyplot as plt
import numpy as np

#Sesuai dengan soal mengambil label 'shirt' dan 'sneaker' untuk dataset ini
classes_of_interest = ['Shirt', 'Sneaker']
original_labels = [6, 7] #shirt = 6, sneaker = 7

mask = np.isin(y_train, original_labels)

# Filter pada training data
x_train_filtered = x_train[mask]
y_train_filtered = y_train[mask]

# Melakukan Mapping dari original labels ke new labels (0 for 'Shirt', 1 for 'Sneaker')
mapping = {6: 0, 7: 1}
y_train_filtered = np.array([mapping[label] for label in y_train_filtered])

class_names = ['Shirt', 'Sneaker']

plt.figure(figsize=(10, 10))
for i in range(min(36, len(x_train_filtered))):
    plt.subplot(6, 6, i + 1)
    plt.xticks([])
    plt.yticks([])
    plt.grid(False)
    plt.imshow(x_train_filtered[i].reshape((28,28)))
    label_index = int(y_train_filtered[i])
    plt.title(class_names[label_index])
plt.show()
```




Saya membuat class baru untuk model GAN ini dikarenakan pada soal kelas yang diberikan hanya 'shirt' dan 'sneakers'

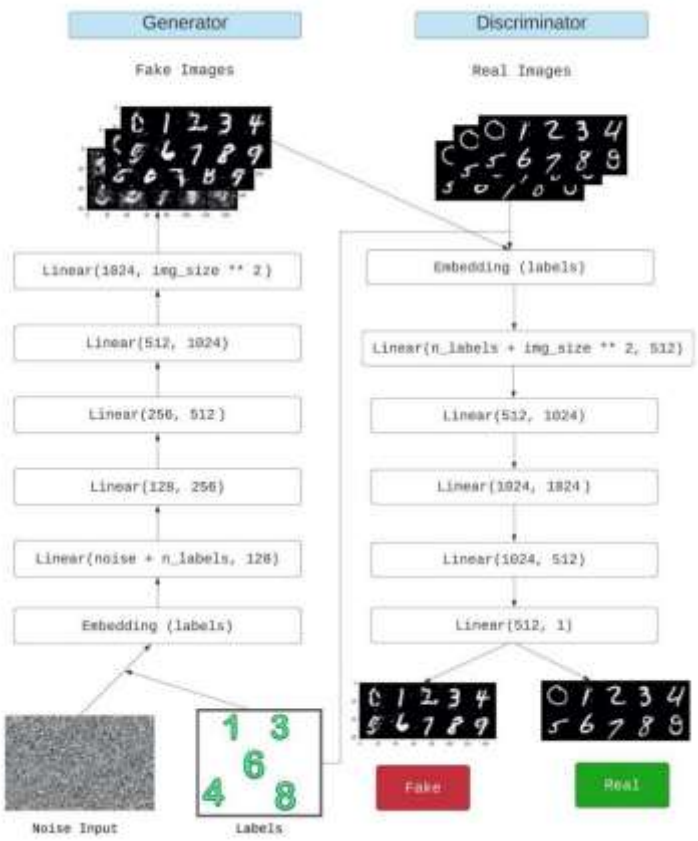
```
#Scaling Dataset
x_train_filtered = x_train_filtered.astype('float32') / 255.0
x_test = x_test.astype('float32') / 255.0
```

```
import matplotlib.pyplot as plt
import numpy as np
from sklearn.model_selection import train_test_split

# Melakukan split data training 80% dan remaining untuk (validation dan test)
x_train, x_remaining, y_train, y_remaining = train_test_split(
    x_train_filtered, y_train_filtered, test_size=0.2, random_state=42
)

# Melakukan split data remaining dibagi menjadi ke test dan validation
x_val, x_test, y_val, y_test = train_test_split(
    x_remaining, y_remaining, test_size=0.5, random_state=42
)


print("Train shapes:", x_train.shape, y_train.shape)
print("Validation shapes:", x_val.shape, y_val.shape)
print("Test shapes:", x_test.shape, y_test.shape)
```

```
Train shapes: (9600, 28, 28) (9600,)
Validation shapes: (1200, 28, 28) (1200,)
Test shapes: (1200, 28, 28) (1200,)
```

# Model GAN

Membuat generator model sesuai dengan arsitektur model GAN pada umumnya




```
from tensorflow.keras import layers
import tensorflow as tf
def generator_model():
    model = tf.keras.Sequential()
    model.add(layers.Dense(7*7*256, use_bias=False, input_shape=(100,)))
    model.add(layers.BatchNormalization())
    model.add(layers.LeakyReLU())

    model.add(layers.Reshape((7, 7, 256)))
    assert model.output_shape == (None, 7, 7, 256)  # Note: None is the batch size

    model.add(layers.Conv2DTranspose(128, (5, 5), strides=(1, 1), padding='same', use_bias=F
    assert model.output_shape == (None, 7, 7, 128)
    model.add(layers.BatchNormalization())
    model.add(layers.LeakyReLU())

    model.add(layers.Conv2DTranspose(64, (5, 5), strides=(2, 2), padding='same', use_bias=Fa
    assert model.output_shape == (None, 14, 14, 64)
    model.add(layers.BatchNormalization())
    model.add(layers.LeakyReLU())

    model.add(layers.Conv2DTranspose(1, (5, 5), strides=(2, 2), padding='same', use_bias=Fal
    assert model.output_shape == (None, 28, 28, 1)

    return model
```

Model arsitektur diatas adalah code dari gambar yang sebelumnya(yang diminta soal). Dalam generator model ada layer dense dengan unit yang berbeda-beda tiap blocknya, lalu ada BatchNormalization yang digunakan untuk mempercepat konvergenti dan mencegah overfitting serta dapat memungkinkan penggunaan learning rate yang tinggi, lalu ada Conv2DTranspose yang digunkaan untuk meningkatkan resolusi data dan yang terakhir ada LeakyReLU itu aktivasi yang digunakan untuk mencegah adanya "Dying ReLU" dimana neuron berhenti karena adanya input negatif terus menerus lalu untuk memperlancar backpropagation

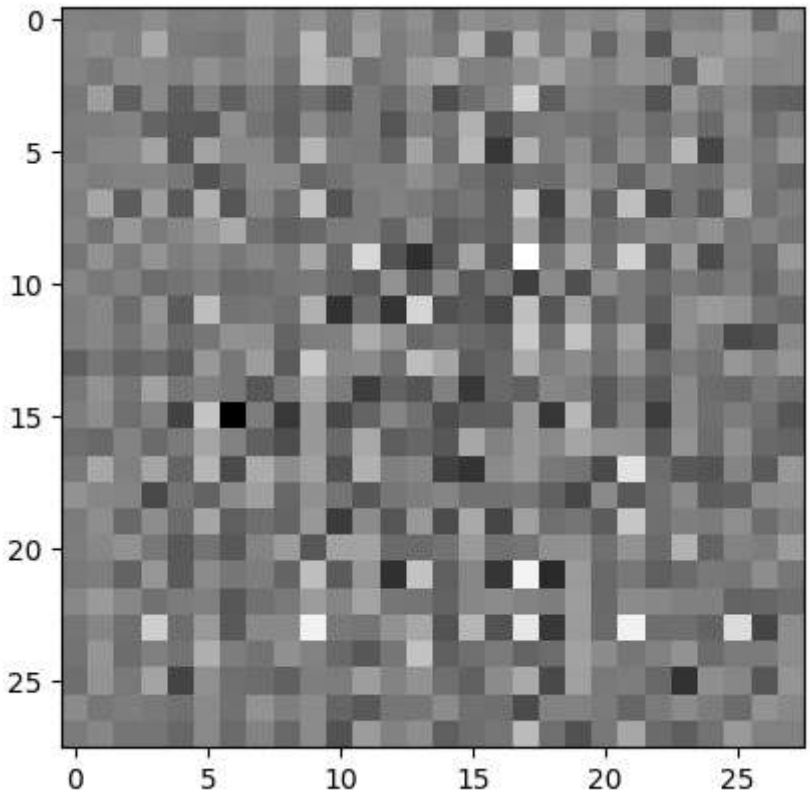
```
generate = generator_model()
generate.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 12544)             1254400

 batch_normalization (Batch  (None, 12544)             50176
 Normalization)

 leaky_re_lu (LeakyReLU)     (None, 12544)             0

 reshape (Reshape)           (None, 7, 7, 256)         0

 conv2d_transpose (Conv2DTr  (None, 7, 7, 128)         819200
 anspose)

 batch_normalization_1 (Bat  (None, 7, 7, 128)         512
 chNormalization)

 leaky_re_lu_1 (LeakyReLU)   (None, 7, 7, 128)         0

 conv2d_transpose_1 (Conv2D  (None, 14, 14, 64)        204800
 Transpose)

 batch_normalization_2 (Bat  (None, 14, 14, 64)        256
 chNormalization)

 leaky_re_lu_2 (LeakyReLU)   (None, 14, 14, 64)        0

 conv2d_transpose_2 (Conv2D  (None, 28, 28, 1)         1600
 Transpose)

=================================================================
Total params: 2330944 (8.89 MB)
Trainable params: 2305472 (8.79 MB)
Non-trainable params: 25472 (99.50 KB)
_________________________________________________________________
```

Memberikan noise pada gambar yang akan di generate nantinya oleh model

```
noise = tf.random.normal([1, 100])
generated_image = generate(noise, training=False)

plt.imshow(generated_image[0, :, :, 0], cmap='gray')
```

```
<matplotlib.image.AxesImage at 0x7a6b3bd41480>
```




Membuat model discriminator untuk menjadi penilai mengenai gambar tersebut. Jadi nantinya discriminator inilah yang membedakan antara gambar asli dan gambar buatan

```
def discriminator_model():
    model = tf.keras.Sequential()
    model.add(layers.Conv2D(64, (5, 5), strides=(2, 2), padding='same',
                                    input_shape=[28, 28, 1]))
    model.add(layers.LeakyReLU())
    model.add(layers.Dropout(0.5))

    model.add(layers.Conv2D(128, (5, 5), strides=(2, 2), padding='same'))
    model.add(layers.LeakyReLU())
    model.add(layers.Dropout(0.5))

    model.add(layers.Flatten())
    model.add(layers.Dense(1))

    return model


discriminator = discriminator_model()
decision = discriminator(generated_image)
print (decision)
```

```
tf.Tensor([[-0.02012704]], shape=(1, 1), dtype=float32)
```

```
cross_entropy = tf.keras.losses.BinaryCrossentropy()
```

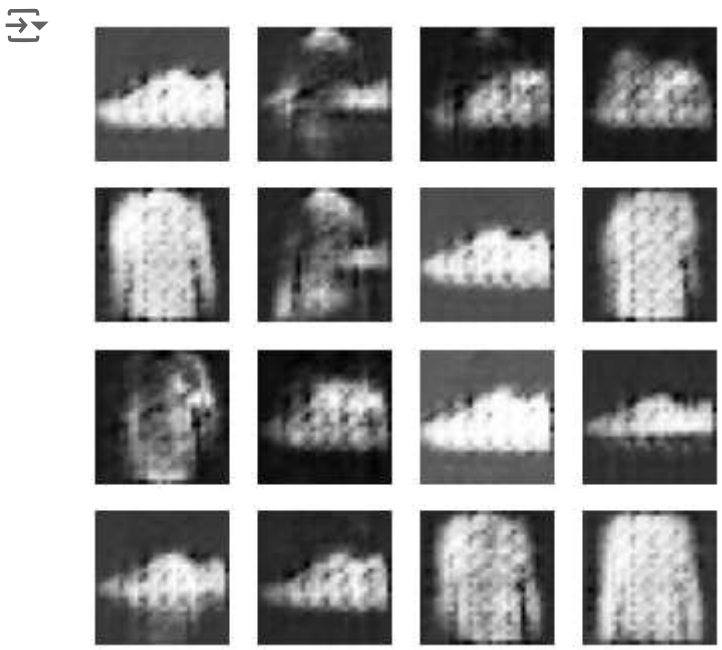
Menggunakan Binary Crossentropy dikarenakan saya ingin melihat probabilitas output dari generator tersebut dan distribusi targetnya cuma ada 2 yaitu 1 untuk gambar nyata dan 2 untuk gambar palsu

```
def discriminator_loss(real_output, fake_output):
    real_loss = cross_entropy(tf.ones_like(real_output), real_output)
    fake_loss = cross_entropy(tf.zeros_like(fake_output), fake_output)
    total_loss = real_loss + fake_loss
    return total_loss
def generate_loss(fake_output):
    return cross_entropy(tf.ones_like(fake_output), fake_output)

generator_optimizer = tf.keras.optimizers.Adam(1e-4)
discriminator_optimizer = tf.keras.optimizers.Adam(1e-4)


EPOCHS = 20
noise_dim = 100
num_examples_to_generate = 16
seed = tf.random.normal([num_examples_to_generate, noise_dim])
```

```
import tensorflow as tf

BATCH_SIZE = 32
noise_dim = 100

def train_step(images, generator, discriminator, generator_optimizer, discriminator_optimize
               generate_loss, discriminator_loss):
  with tf.GradientTape() as gen_tape, tf.GradientTape() as disc_tape:
    noise = tf.random.normal([BATCH_SIZE, noise_dim])

    generated_images = generator(noise, training=True)
    real_output = discriminator(images, training=True)
    fake_output = discriminator(generated_images, training=True)

    gen_loss = generate_loss(fake_output)
    disc_loss = discriminator_loss(real_output, fake_output)

  gradients_of_generator = gen_tape.gradient(gen_loss, generator.trainable_variables)
  gradients_of_discriminator = disc_tape.gradient(disc_loss, discriminator.trainable_variabl

  generator_optimizer.apply_gradients(zip(gradients_of_generator, generator.trainable_variat
  discriminator_optimizer.apply_gradients(zip(gradients_of_discriminator, discriminator.trai

  return gen_loss, disc_loss
```

```
import time
from IPython import display # Import the IPython display module

def train(dataset, epochs, generator, discriminator, generator_optimizer, discriminator_opti
  for epoch in range(epochs):
    start = time.time()

    for image_batch in dataset:
      train_step(image_batch, generator, discriminator, generator_optimizer, discriminator_o

    # Produce images for the GIF as you go
    display.clear_output(wait=True) # Now refers to the correct display object
    generate_and_save_images(generate,
                             epoch + 1,
                             seed)

    # Save the model every 15 epochs
    if (epoch + 1) % 15 == 0:
      checkpoint.save(file_prefix = checkpoint_prefix)

    print ('Time for epoch {} is {} sec'.format(epoch + 1, time.time()-start))

  # Generate after the final epoch
  display.clear_output(wait=True)
  generate_and_save_images(generate,
                           epochs,
                           seed)


import os
checkpoint_dir = './training_checkpoints'
checkpoint_prefix = os.path.join(checkpoint_dir, "ckpt")
checkpoint = tf.train.Checkpoint(generator_optimizer=generator_optimizer,
                                 discriminator_optimizer=discriminator_optimizer,
                                 generator=generate,
                                 discriminator=discriminator)
```

```
def generate_and_save_images(model, epoch, test_input):
  # Notice `training` is set to False.
  # This is so all layers run in inference mode (batchnorm).
  predictions = model(test_input, training=False)

  fig = plt.figure(figsize=(4, 4))

  for i in range(predictions.shape[0]):
      plt.subplot(4, 4, i+1)
      plt.imshow(predictions[i, :, :, 0] * 127.5 + 127.5, cmap='gray')
      plt.axis('off')

  plt.savefig('image_at_epoch_{:04d}.png'.format(epoch))
  plt.show()


batch_size = 32
train_dataset = tf.data.Dataset.from_tensor_slices(x_train_filtered).shuffle(1000).batch(bat


EPOCHS = 25
# Assuming you have defined generator, discriminator, etc.
train(train_dataset, EPOCHS, generate, discriminator, generator_optimizer, discriminator_opt
```



Dari hasil generate diatas, model dapat melakukan generate image dengan baik. walaupun pada gambar diatas tidak terlalu begitu jelas dikarena hasil gambar hanya berasal dari 25 epochs saja sedangkan untuk menghasilkan gambar generate yang baik butuh banyak epoch yang dilakukan

```
noise = tf.random.normal([1, 100])
generated_image = generate(noise, training=False)

plt.imshow(generated_image[0, :, :, 0], cmap='gray')
```

```
<matplotlib.image.AxesImage at 0x7a6bacd91c90>
```