Import Modules

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

import warnings
from tqdm.notebook import tqdm

import tensorflow as tf
from tensorflow import keras
from tensorflow.keras.preprocessing.image import load_img,array_to_img

from tensorflow.keras.models import Sequential,Model
from tensorflow.keras import layers
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.losses import BinaryCrossentropy

warnings.filterwarnings('ignore')
```

```
2024-07-09 15:12:46.947770: E
external/local_xla/xla/stream_executor/cuda/cuda_dnn.cc:9261] Unable
to register cuDNN factory: Attempting to register factory for plugin
cuDNN when one has already been registered
2024-07-09 15:12:46.947876: E
external/local_xla/xla/stream_executor/cuda/cuda_fft.cc:607] Unable to
register cuFFT factory: Attempting to register factory for plugin
cuFFT when one has already been registered
2024-07-09 15:12:47.104276: E
external/local_xla/xla/stream_executor/cuda/cuda_blas.cc:1515] Unable
to register cuBLAS factory: Attempting to register factory for plugin
cuBLAS when one has already been registered
```

Load Files

```
base_dir='/kaggle/input/anime-faces/data'

#Collect the full paths of all the files in the specified directory
(base_dir).
img_paths=[]
for i in os.listdir(base_dir):
    image_path=os.path.join(base_dir,i)
    img_paths.append(image_path)

img_paths[:5]
```

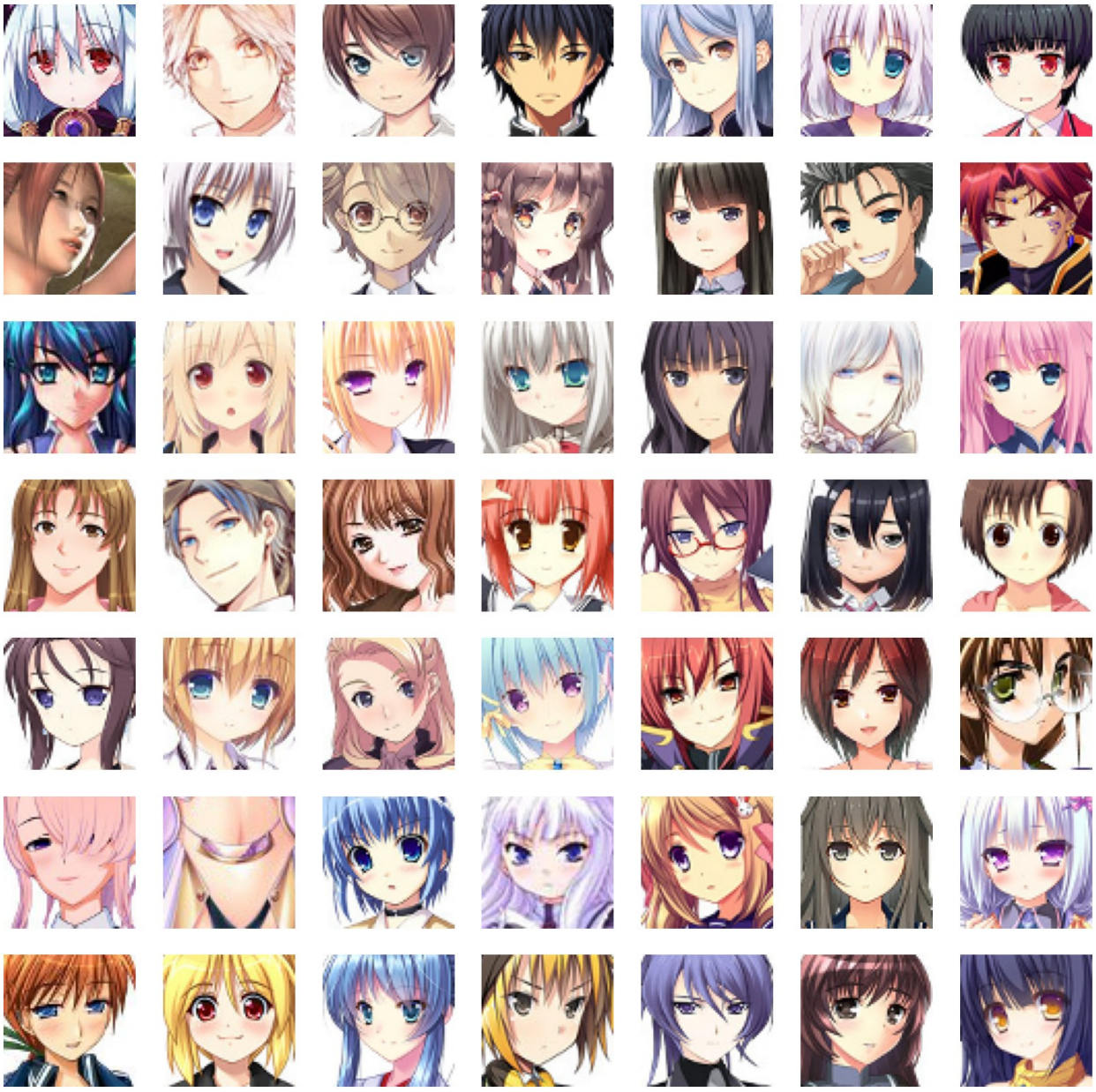
```
['/kaggle/input/anime-faces/data/21130.png',
 '/kaggle/input/anime-faces/data/9273.png',
 '/kaggle/input/anime-faces/data/18966.png',
 '/kaggle/input/anime-faces/data/14127.png',
 '/kaggle/input/anime-faces/data/18054.png']
```

```
#remove the files inside file(data)
img_paths.remove('/kaggle/input/anime-faces/data/data')
```

```
len(img_paths)
```

```
21551
```

```
#To display grid of image
plt.figure(figsize=(20,20))
temp_img=img_paths[:49]
index=1
for i in temp_img:
    plt.subplot(7,7,index) #creating a 7*7 grid of subplots
    img=load_img(i)
    img=np.array(img)
    plt.imshow(img)
    plt.axis('off')
    index+=1
plt.show()
```



Preprocess Images

```
#Wraps image_path (a list of image file paths) with tqdm to display a progress bar during iteration.
trained_images=[np.array(load_img(i))for i in tqdm(img_paths)]

#This converts the list of image arrays into a single NumPy array.
trained_images=np.array(trained_images)
```

{"model_id":"27113e1089144b55a8cf96a407700f13","version_major":2,"version_minor":0}

```
trained_images[0].shape
```

```
(64, 64, 3)
```

```
trained_images[0]
```

```
array([[[ 35,  12,  36],
        [ 37,  15,  40],
        [ 38,   0,  36],
        ...,
        [116, 129, 177],
        [103, 135, 185],
        [ 84, 110, 165]],

       [[ 35,  12,  36],
        [ 34,  17,  38],
        [ 61,  17,  55],
        ...,
        [ 90, 129, 166],
        [117, 147, 195],
        [ 97, 139, 186]],

       [[ 38,  14,  38],
        [ 36,  16,  39],
        [ 55,  19,  53],
        ...,
        [ 99, 150, 189],
        [110, 150, 188],
        [119, 159, 192]],

       ...,

       [[ 88,  40,  90],
        [ 83,  40,  80],
        [ 88,  59,  82],
        ...,
        [ 26,   2,  29],
        [ 22,  16,  18],
        [ 30,  10,  25]],

       [[ 86,  46,  90],
        [ 71,  35,  72],
        [ 58,  27,  58],
        ...,
        [ 21,   3,  37],
        [ 35,  16,  31],
        [ 33,  11,  39]],

       [[ 54,  22,  58],
        [ 56,  17,  60],
        [ 53,  18,  54],
```

```
        ...,
        [ 33,  17,  49],
        [ 35,   8,  37],
        [ 30,   9,  36]]], dtype=uint8)
```

```
#Determine Number of Images:
num_img=len(trained_images)

#Calculate New Shape:
new_shape=(num_img,64,64,3)

#Use np.float32 for compatibility with most deep learning frameworks.
trained_images=trained_images.reshape(new_shape).astype(np.float32)

len(trained_images)
```

```
21551
```

```
#Standardize the images [-1,1], due to use of tanh as the activation
function
trained_images=(trained_images-127.5)/127.5

trained_images[0]
```

```
array([[[-0.7254902 , -0.90588236, -0.7176471 ],
        [-0.70980394, -0.88235295, -0.6862745 ],
        [-0.7019608 , -1.        , -0.7176471 ],
        ...,
        [-0.09019608,  0.01176471,  0.3882353 ],
        [-0.19215687,  0.05882353,  0.4509804 ],
        [-0.34117648, -0.13725491,  0.29411766]],

       [[-0.7254902 , -0.90588236, -0.7176471 ],
        [-0.73333335, -0.8666667 , -0.7019608 ],
        [-0.52156866, -0.8666667 , -0.5686275 ],
        ...,
        [-0.29411766,  0.01176471,  0.3019608 ],
        [-0.08235294,  0.15294118,  0.5294118 ],
        [-0.23921569,  0.09019608,  0.45882353]],

       [[-0.7019608 , -0.8901961 , -0.7019608 ],
        [-0.7176471 , -0.8745098 , -0.69411767],
        [-0.5686275 , -0.8509804 , -0.58431375],
        ...,
        [-0.22352941,  0.1764706 ,  0.48235294],
        [-0.13725491,  0.1764706 ,  0.4745098 ],
        [-0.06666667,  0.24705882,  0.5058824 ]],

       ...,

       [[-0.30980393, -0.6862745 , -0.29411766],
```

```
        [-0.34901962, -0.6862745 , -0.37254903],
        [-0.30980393, -0.5372549 , -0.35686275],
        ...,
        [-0.79607844, -0.9843137 , -0.77254903],
        [-0.827451  , -0.8745098 , -0.85882354],
        [-0.7647059 , -0.92156863, -0.8039216 ]],

       [[-0.3254902 , -0.6392157 , -0.29411766],
        [-0.44313726, -0.7254902 , -0.43529412],
        [-0.54509807, -0.7882353 , -0.54509807],
        ...,
        [-0.8352941 , -0.9764706 , -0.70980394],
        [-0.7254902 , -0.8745098 , -0.75686276],
        [-0.7411765 , -0.9137255 , -0.69411767]],

       [[-0.5764706 , -0.827451  , -0.54509807],
        [-0.56078434, -0.8666667 , -0.5294118 ],
        [-0.58431375, -0.85882354, -0.5764706 ],
        ...,
        [-0.7411765 , -0.8666667 , -0.6156863 ],
        [-0.7254902 , -0.9372549 , -0.70980394],
        [-0.7647059 , -0.92941177, -0.7176471 ]]], dtype=float32)
```

Create Generator and Discriminator model

```
#Latent dimension for random noise
latent_dim=100
#Proper initialization of the weights of a neural network is crucial
for effective training.
wg_init=keras.initializers.random_normal(mean=0.0,stddev=0.02)
channels=3 #refers to rgb
```

Generator Model

```
model=Sequential(name='Generator')

#1:- 1-d Random noise
model.add(layers.Dense(8*8*512,input_dim=latent_dim))
model.add(layers.ReLU())

#convert 1-d to 3-d
model.add(layers.Reshape((8,8,512)))

#upsample to 16*16
model.add(layers.Conv2DTranspose(256,
(4,4),strides=(2,2),padding='same',kernel_initializer=wg_init))
```

```
model.add(layers.ReLU())

model.add(layers.Conv2DTranspose(128,
(4,4),strides=(2,2),padding='same',kernel_initializer=wg_init))
model.add(layers.ReLU())

model.add(layers.Conv2DTranspose(64,
(4,4),strides=(2,2),padding='same',kernel_initializer=wg_init))
model.add(layers.ReLU())

model.add(layers.Conv2D(channels,
(4,4),padding='same',activation='tanh'))

model.summary()
```

```
Model: "Generator"
```

```
┏━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━┓
┃ Layer (type)                    ┃ Output Shape           ┃       Param # ┃
┡━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━┩
│ dense (Dense)                   │ (None, 32768)          │     3,309,568 │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ re_lu (ReLU)                    │ (None, 32768)          │             0 │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ reshape (Reshape)               │ (None, 8, 8, 512)      │             0 │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ conv2d_transpose                │ (None, 16, 16, 256)    │     2,097,408 │
│ (Conv2DTranspose)               │                        │               │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ re_lu_1 (ReLU)                  │ (None, 16, 16, 256)    │             0 │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ conv2d_transpose_1              │ (None, 32, 32, 128)    │       524,416 │
│ (Conv2DTranspose)               │                        │               │
├─────────────────────────────────┼────────────────────────┼───────────────┤
```

```
──────┤
| re_lu_2 (ReLU)                    | (None, 32, 32, 128)    |
0 |
├─────────────────────────────────┼──────────────────────────┼──────────
──────┤
| conv2d_transpose_2                | (None, 64, 64, 64)     |
131,136 |
| (Conv2DTranspose)                 |                        |
|
├─────────────────────────────────┼──────────────────────────┼──────────
──────┤
| re_lu_3 (ReLU)                    | (None, 64, 64, 64)     |
0 |
├─────────────────────────────────┼──────────────────────────┼──────────
──────┤
| conv2d (Conv2D)                   | (None, 64, 64, 3)      |
3,075 |
└─────────────────────────────────┴──────────────────────────┴──────────
──────┘

 Total params: 6,065,603 (23.14 MB)

 Trainable params: 6,065,603 (23.14 MB)

 Non-trainable params: 0 (0.00 B)
```

Discriminator Model

```
model1=Sequential(name='Discriminator')
input_shape=(64,64,3)
alpha=0.2   # Sets the negative slope coefficient for the Leaky ReLU
activation function.

#create conv layers

model1.add(layers.Conv2D(64,
(4,4),strides=(2,2),padding='same',input_shape=input_shape))
model1.add(layers.BatchNormalization())
model1.add(layers.LeakyReLU(alpha=alpha))

model1.add(layers.Conv2D(128,(4,4),strides=(2,2),padding='same'))
model1.add(layers.BatchNormalization())
model1.add(layers.LeakyReLU(alpha=alpha))

model1.add(layers.Conv2D(128,(4,4),strides=(2,2),padding='same'))
model1.add(layers.BatchNormalization())
model1.add(layers.LeakyReLU(alpha=alpha))

model1.add(layers.Flatten())
```

```
model1.add(layers.Dropout(0.3))

#output class
model1.add(layers.Dense(1,activation='sigmoid'))

model1.summary()
```

```
Model: "Discriminator"
```

```
┏━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━┓
┃ Layer (type)                         ┃ Output Shape                ┃
Param # ┃
┡━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━┩
│ conv2d_1 (Conv2D)                    │ (None, 32, 32, 64)          │
3,136 │
├──────────────────────────────────────┼─────────────────────────────┼─────────────────
┤
│ batch_normalization                  │ (None, 32, 32, 64)          │
256 │
│ (BatchNormalization)                 │                             │
│
├──────────────────────────────────────┼─────────────────────────────┼─────────────────
┤
│ leaky_re_lu (LeakyReLU)              │ (None, 32, 32, 64)          │
0 │
├──────────────────────────────────────┼─────────────────────────────┼─────────────────
┤
│ conv2d_2 (Conv2D)                    │ (None, 16, 16, 128)         │
131,200 │
├──────────────────────────────────────┼─────────────────────────────┼─────────────────
┤
│ batch_normalization_1                │ (None, 16, 16, 128)         │
512 │
│ (BatchNormalization)                 │                             │
│
├──────────────────────────────────────┼─────────────────────────────┼─────────────────
┤
│ leaky_re_lu_1 (LeakyReLU)            │ (None, 16, 16, 128)         │
0 │
├──────────────────────────────────────┼─────────────────────────────┼─────────────────
┤
│ conv2d_3 (Conv2D)                    │ (None, 8, 8, 128)           │
262,272 │
├──────────────────────────────────────┼─────────────────────────────┼─────────────────
┤
│ batch_normalization_2                │ (None, 8, 8, 128)           │
512 │
│ (BatchNormalization)                 │                             │
```

```
│
├─────────────────────────────────────┼─────────────────────────────┼──────────
──────┤
│ leaky_re_lu_2 (LeakyReLU)           │ (None, 8, 8, 128)           │
0 │
├─────────────────────────────────────┼─────────────────────────────┼──────────
──────┤
│ flatten (Flatten)                   │ (None, 8192)                │
0 │
├─────────────────────────────────────┼─────────────────────────────┼──────────
──────┤
│ dropout (Dropout)                   │ (None, 8192)                │
0 │
├─────────────────────────────────────┼─────────────────────────────┼──────────
──────┤
│ dense_1 (Dense)                     │ (None, 1)                   │
8,193 │
└─────────────────────────────────────┴─────────────────────────────┴──────────
──────┘

 Total params: 406,081 (1.55 MB)

 Trainable params: 405,441 (1.55 MB)

 Non-trainable params: 640 (2.50 KB)
```

Create DCGAN Class

```
class DCGAN(keras.Model):
    def __init__(self,generator,discriminator,latent_dim):
        super().__init__()
        self.generator=generator
        self.discriminator=discriminator
        self.latent_dim=latent_dim
        self.g_loss_metric=keras.metrics.Mean(name='g_loss')
        self.d_loss_metric=keras.metrics.Mean(name='d_loss')

    @property
    def metrics(self):
        return[self.g_loss_metric,self.d_loss_metric]

    #compile Method: Sets up the optimizers for the generator
(g_optimizer) and discriminator (d_optimizer),
    #and the loss function (loss_fn).
    def compile(self,g_optimizer,d_optimizer,loss_fn):
        super(DCGAN,self).compile()
        self.g_optimizer=g_optimizer
        self.d_optimizer=d_optimizer
```

```
        self.loss_fn=loss_fn

    #train_step Method: Implements the custom training logic for each
step.

    def train_step(self,real_images):
        #get the batch size from the data
        batch_size=tf.shape(real_images)[0]
        #generate random noise

random_noise=tf.random.normal(shape=(batch_size,self.latent_dim))


        #train the discriminator with real(1) and fake(0) images
        #Discriminator Training:
        with tf.GradientTape() as tape:
            #compute loss on real images
            pred_real=self.discriminator(real_images,training=True)
            #generate real image labels
            real_labels=tf.ones((batch_size,1))
            real_labels +=
0.05*tf.random.uniform(tf.shape(real_labels))
            d_loss_real = self.loss_fn(real_labels,pred_real)

            #compute loss on fake images
            fake_images=self.generator(random_noise)
            pred_fake=self.discriminator(fake_images,training=True)
            #generate fake labels
            fake_labels = tf.zeros((batch_size,1))
            d_loss_fake = self.loss_fn(fake_labels,pred_fake)

            #total discriminator loss
            d_loss = (d_loss_real + d_loss_fake) / 2

        #compute discriminator gradients
        gradients =
tape.gradient(d_loss,self.discriminator.trainable_variables)
        #update the gradients

self.d_optimizer.apply_gradients(zip(gradients,self.discriminator.trai
nable_variables))


        #train the generator model
        #Generator Training:
        labels = tf.ones((batch_size,1))
        #generator want the discriminator to think that fake images
are real
        with tf.GradientTape() as tape:
```

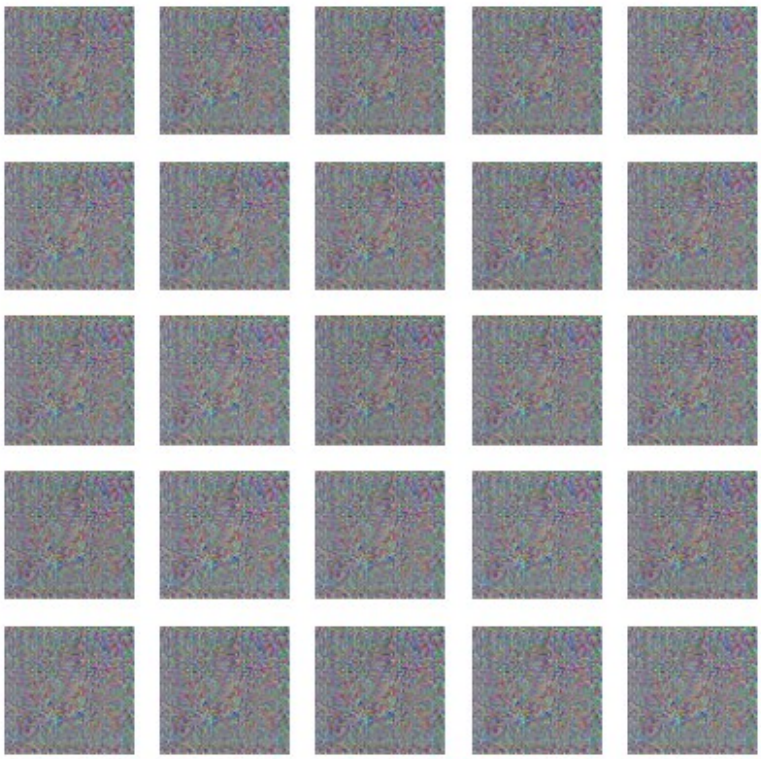
```
            #generate the fake images from the generator
            fake_images = self.generator(random_noise,training=True)
            #classify images as real or fake
            pred_fake = self.discriminator(fake_images,training=True)
            #compute loss
            g_loss = self.loss_fn(labels,pred_fake)

        #compute gradients
        gradients =
tape.gradient(g_loss,self.generator.trainable_variables)
        #update the gradients

self.g_optimizer.apply_gradients(zip(gradients,self.generator.trainabl
e_variables))

        #update states(metrics) for both models(losses)
        self.d_loss_metric.update_state(d_loss)
        self.g_loss_metric.update_state(g_loss)

        #Return Losses: Returns the current values of the
discriminator and generator losses for monitoring.
        return {'d_loss' : self.d_loss_metric.result(), 'g_loss':
self.g_loss_metric.result()}
```

Moniter Class

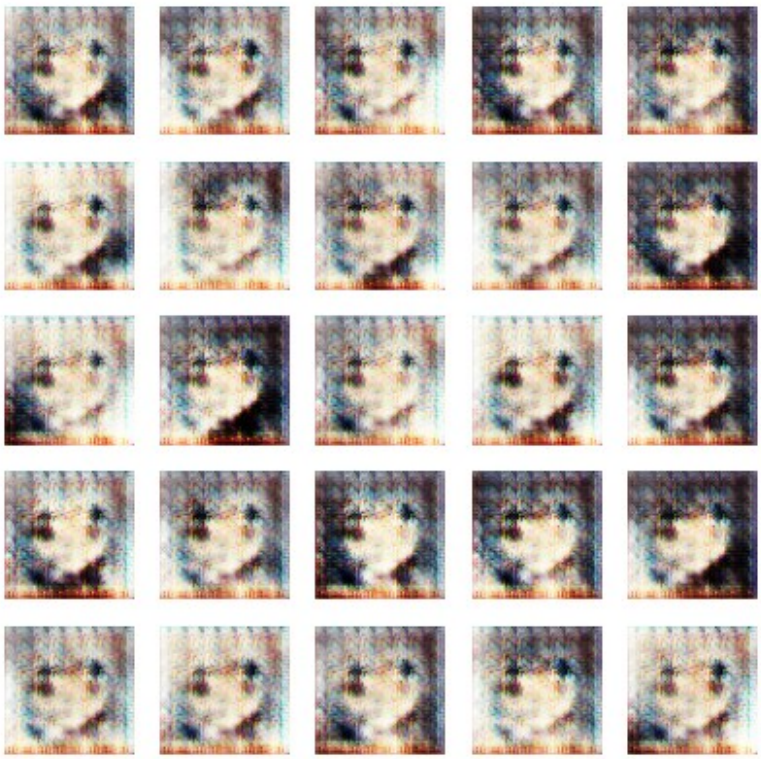
```
#It is a custom Keras callback designed to monitor the training
process of a Deep Convolutional Generative Adversarial
# Network (DCGAN). It generates and displays images at the end of each
epoch and saves the generator model at the end
# of the training. This helps in visualizing the progress of the
generator during the training process.

class DCGANMoniter(keras.callbacks.Callback):
    def __init__(self,num_imgs=25,latent_dim=100):
        self.num_imgs=num_imgs
        self.latent_dim=latent_dim
        #create random noise for generating images
        self.noise = tf.random.normal([25,latent_dim])

    def on_epoch_end(self,epoch,logs=None):
        #generate the image from noise
        g_img=self.model.generator(self.noise)
        #denormalize the image
        g_img=(g_img * 127.5)*127.5
        g_img.numpy()

        fig = plt.figure(figsize=(5,5))
```

```
        for i in range(self.num_imgs):
            plt.subplot(5,5,i+1)
            img=array_to_img(g_img[i])
            plt.imshow(img)
            plt.axis('off')
        #plt.savefig('epoch_{:03d}.png'.format(epoch))  to save the
image that is generated
        plt.show()

    def on_train_end(self,logs=None):
        self.model.generator.save('generator.h5')

#Initialize the DCGAN Model:
dcgan=DCGAN(generator=model,discriminator=model1,latent_dim=latent_dim
)

#Different learning rates are used to control the training speed of
the generator and discriminator.
d_lr=0.0001
g_lr=0.0003
#Typically, the generator learning rate is higher to ensure that it
learns faster than the discriminator.

#Compile the model:
dcgan.compile(g_optimizer=Adam(learning_rate=g_lr,beta_1=0.5),d_optimi
zer=Adam(learning_rate=d_lr,beta_1=0.5),loss_fn=BinaryCrossentropy())

# Instantiate the DCGANMoniter callback
dcgan_monitor = DCGANMoniter(num_imgs=25, latent_dim=latent_dim)

# Train the DCGAN model
n_epochs = 100
dcgan.fit(trained_images, epochs=n_epochs, callbacks=[dcgan_monitor])
```

```
Epoch 1/100
  2/674 ━━━━━━━━━━━━━━━━━━━━ 54s 81ms/step - d_loss: 0.9070 - g_loss:
1.1154

WARNING: All log messages before absl::InitializeLog() is called are
written to STDERR
I0000 00:00:1720538177.539359     121 device_compiler.h:186] Compiled
cluster using XLA!  This line is logged at most once for the lifetime
of the process.

674/674 ━━━━━━━━━━━━━━━━━━━━ 0s 65ms/step - d_loss: 1.3018 - g_loss:
1.8264
```



```
674/674 ━━━━━━━━━━━━━━━━━━━━ 68s 68ms/step - d_loss: 1.3008 - g_loss:
1.8278
Epoch 2/100
673/674 ━━━━━━━━━━━━━━━━━━━━ 0s 44ms/step - d_loss: 0.1554 - g_loss:
3.4273
```

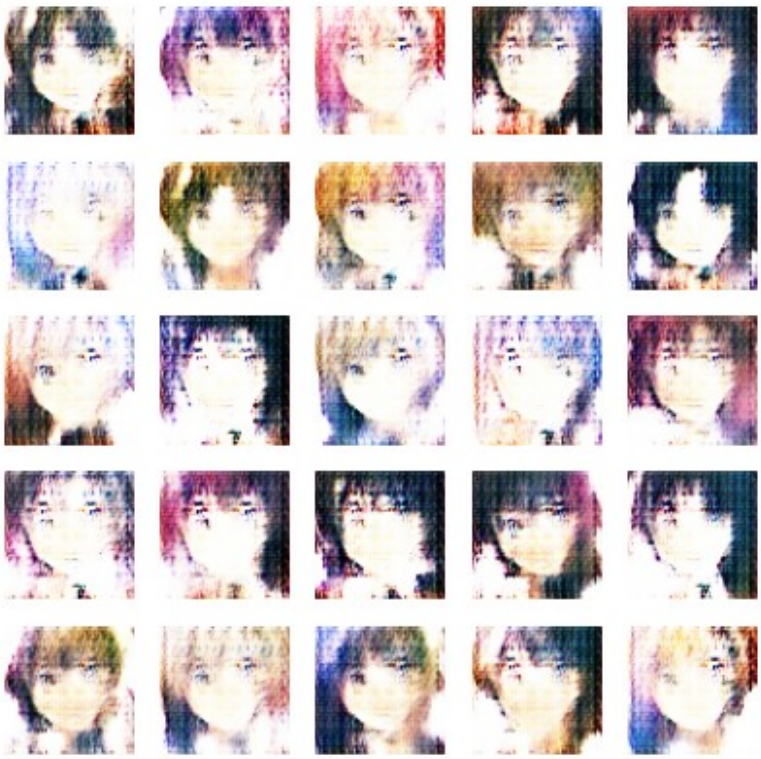


```
674/674 ━━━━━━━━━━━━━━━━━━━━ 31s 46ms/step - d_loss: 0.1561 - g_loss:
3.4250
Epoch 3/100
673/674 ━━━━━━━━━━━━━━━━━━━━ 0s 46ms/step - d_loss: 0.5784 - g_loss:
1.7857
```

```
674/674 ━━━━━━━━━━━━━━━━━━━━ 32s 47ms/step - d_loss: 0.5784 - g_loss:
1.7856
Epoch 4/100
673/674 ━━━━━━━━━━━━━━━━━━━━ 0s 47ms/step - d_loss: 0.6040 - g_loss:
1.6797
```

```
674/674 ━━━━━━━━━━━━━━━━━━━━ 32s 48ms/step - d_loss: 0.6040 - g_loss:
1.6797
Epoch 5/100
674/674 ━━━━━━━━━━━━━━━━━━━━ 0s 46ms/step - d_loss: 0.6106 - g_loss:
1.6516
```

```
674/674 ━━━━━━━━━━━━━━━━━━━━ 32s 47ms/step - d_loss: 0.6106 - g_loss:
1.6516
Epoch 6/100
673/674 ━━━━━━━━━━━━━━━━━━━━ 0s 46ms/step - d_loss: 0.6210 - g_loss:
1.6051
```



```
674/674 ━━━━━━━━━━━━━━━━━━━━ 32s 47ms/step - d_loss: 0.6209 - g_loss:
1.6053
Epoch 7/100
673/674 ━━━━━━━━━━━━━━━━━━━━ 0s 46ms/step - d_loss: 0.5656 - g_loss:
1.7666
```

```
674/674 ━━━━━━━━━━━━━━━━━━━━ 32s 47ms/step - d_loss: 0.5656 - g_loss:
1.7665
Epoch 8/100
674/674 ━━━━━━━━━━━━━━━━━━━━ 0s 46ms/step - d_loss: 0.5605 - g_loss:
1.5444
```

```
674/674 ━━━━━━━━━━━━━━━━━━━━ 41s 48ms/step - d_loss: 0.5605 - g_loss:
1.5444
Epoch 9/100
674/674 ━━━━━━━━━━━━━━━━━━━━ 0s 46ms/step - d_loss: 0.5420 - g_loss:
1.6690
```

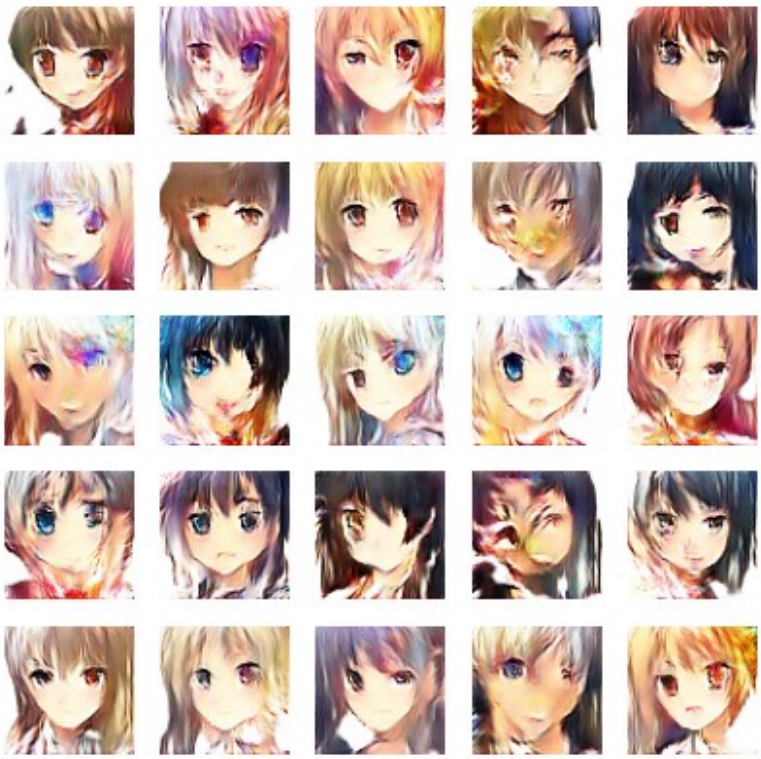
```
674/674 ━━━━━━━━━━━━━━━━━━━━ 41s 48ms/step - d_loss: 0.5419 - g_loss:
1.6691
Epoch 10/100
673/674 ━━━━━━━━━━━━━━━━━━━━ 0s 46ms/step - d_loss: 0.5281 - g_loss:
1.6972
```

```
674/674 ━━━━━━━━━━━━━━━━━━━━ 32s 47ms/step - d_loss: 0.5281 - g_loss:
1.6971
Epoch 11/100
674/674 ━━━━━━━━━━━━━━━━━━━━ 0s 46ms/step - d_loss: 0.5609 - g_loss:
1.5174
```

```
674/674 ━━━━━━━━━━━━━━━━━━━━ 32s 47ms/step - d_loss: 0.5610 - g_loss:
1.5173
Epoch 12/100
673/674 ━━━━━━━━━━━━━━━━━━━━ 0s 46ms/step - d_loss: 0.6178 - g_loss:
1.3016
```

```
674/674 ━━━━━━━━━━━━━━━━━━━━ 32s 48ms/step - d_loss: 0.6178 - g_loss:
1.3015
Epoch 13/100
674/674 ━━━━━━━━━━━━━━━━━━━━ 0s 46ms/step - d_loss: 0.6352 - g_loss:
1.1770
```

```
674/674 ━━━━━━━━━━━━━━━━━━━━ 32s 48ms/step - d_loss: 0.6352 - g_loss:
1.1770
Epoch 14/100
674/674 ━━━━━━━━━━━━━━━━━━━━ 0s 46ms/step - d_loss: 0.6324 - g_loss:
1.1746
```

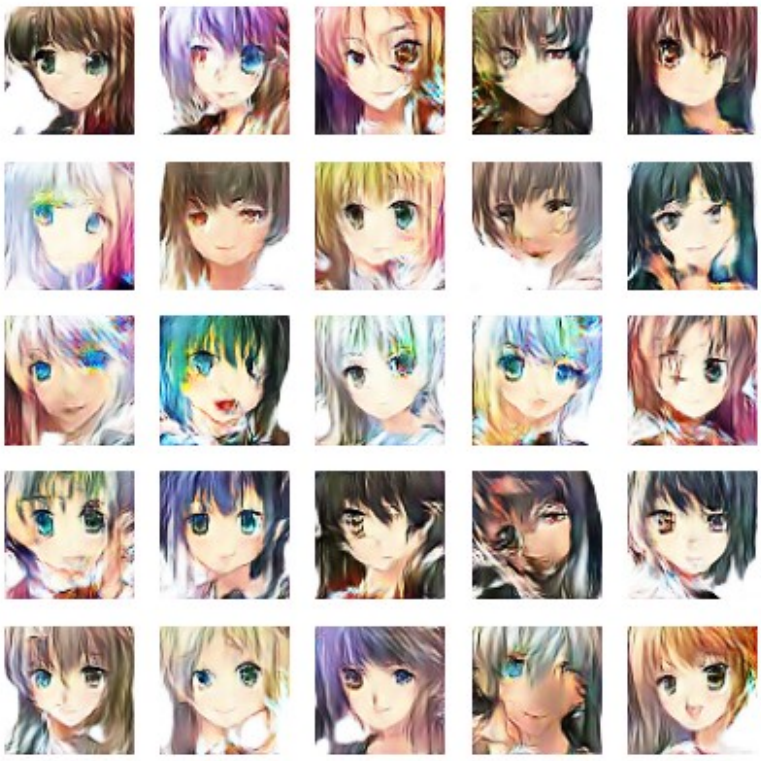


```
674/674 ━━━━━━━━━━━━━━━━━━━━ 32s 48ms/step - d_loss: 0.6324 - g_loss:
1.1746
Epoch 15/100
673/674 ━━━━━━━━━━━━━━━━━━━━ 0s 47ms/step - d_loss: 0.6447 - g_loss:
1.0981
```

```
674/674 ━━━━━━━━━━━━━━━━━━━━ 41s 48ms/step - d_loss: 0.6447 - g_loss:
1.0981
Epoch 16/100
673/674 ━━━━━━━━━━━━━━━━━━━━ 0s 46ms/step - d_loss: 0.6567 - g_loss:
1.0357
```

```
674/674 ━━━━━━━━━━━━━━━━━━━━ 32s 48ms/step - d_loss: 0.6567 - g_loss:
1.0357
Epoch 17/100
673/674 ━━━━━━━━━━━━━━━━━━━━ 0s 46ms/step - d_loss: 0.6571 - g_loss:
1.0216
```

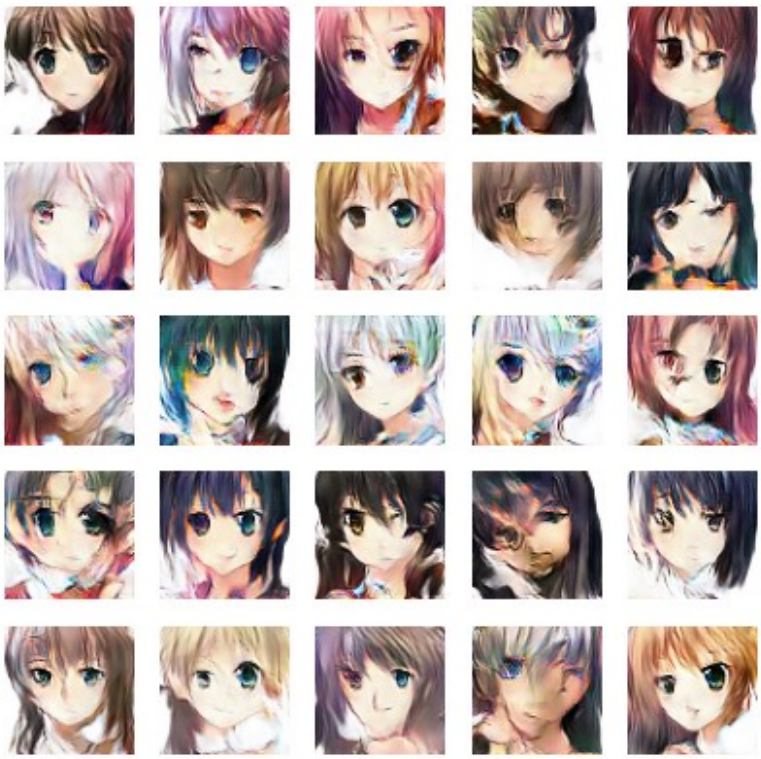
```
674/674 ━━━━━━━━━━━━━━━━━━━━ 32s 48ms/step - d_loss: 0.6571 - g_loss:
1.0216
Epoch 18/100
674/674 ━━━━━━━━━━━━━━━━━━━━ 0s 46ms/step - d_loss: 0.6654 - g_loss:
0.9858
```



```
674/674 ━━━━━━━━━━━━━━━━━━━━ 32s 47ms/step - d_loss: 0.6654 - g_loss:
0.9858
Epoch 19/100
674/674 ━━━━━━━━━━━━━━━━━━━━ 0s 46ms/step - d_loss: 0.6668 - g_loss:
0.9669
```

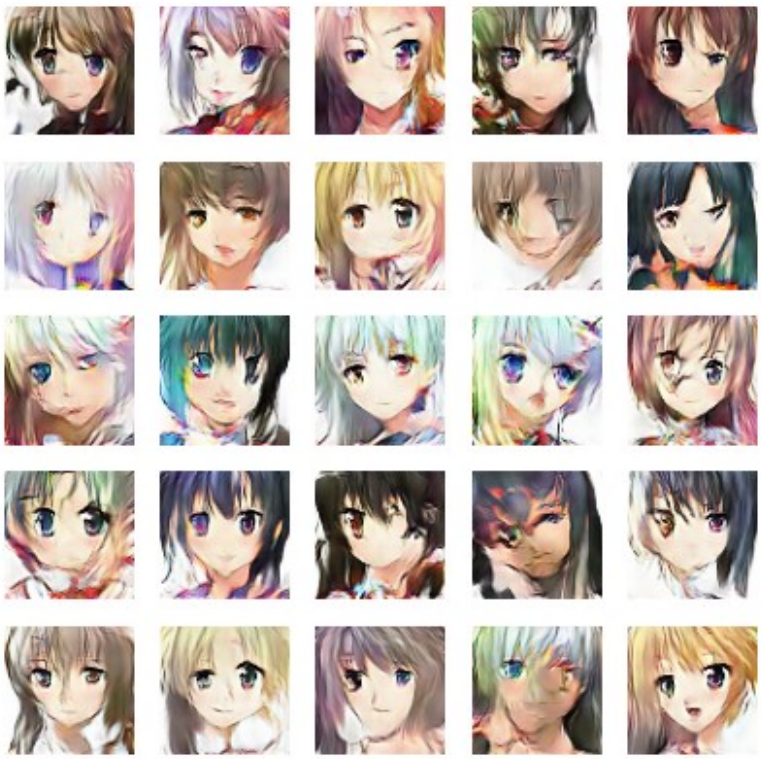
```
674/674 ━━━━━━━━━━━━━━━━━━━━ 32s 47ms/step - d_loss: 0.6668 - g_loss:
0.9669
Epoch 20/100
674/674 ━━━━━━━━━━━━━━━━━━━━ 0s 46ms/step - d_loss: 0.6701 - g_loss:
0.9456
```

```
674/674 ━━━━━━━━━━━━━━━━━━━━ 32s 47ms/step - d_loss: 0.6701 - g_loss:
0.9456
Epoch 21/100
673/674 ━━━━━━━━━━━━━━━━━━━━ 0s 47ms/step - d_loss: 0.6662 - g_loss:
0.9365
```

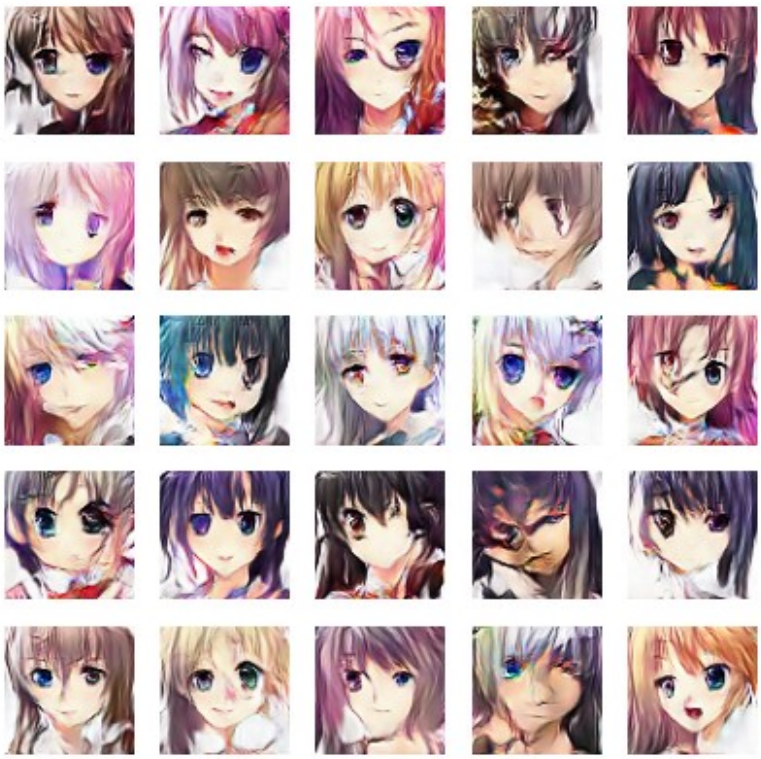


```
674/674 ━━━━━━━━━━━━━━━━━━━━ 42s 48ms/step - d_loss: 0.6662 - g_loss:
0.9365
Epoch 22/100
674/674 ━━━━━━━━━━━━━━━━━━━━ 0s 46ms/step - d_loss: 0.6648 - g_loss:
0.9290
```

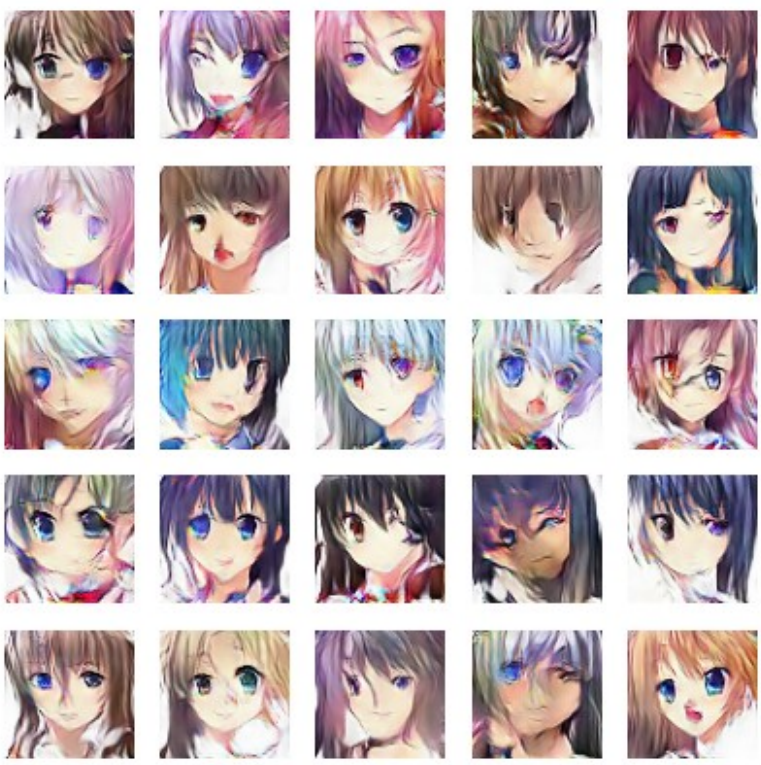
```
674/674 ━━━━━━━━━━━━━━━━━━━━ 32s 47ms/step - d_loss: 0.6648 - g_loss:
0.9290
Epoch 23/100
673/674 ━━━━━━━━━━━━━━━━━━━━ 0s 46ms/step - d_loss: 0.6633 - g_loss:
0.9322
```



```
674/674 ━━━━━━━━━━━━━━━━━━━━ 32s 47ms/step - d_loss: 0.6633 - g_loss:
0.9321
Epoch 24/100
674/674 ━━━━━━━━━━━━━━━━━━━━ 0s 46ms/step - d_loss: 0.6635 - g_loss:
0.9306
```



```
674/674 ━━━━━━━━━━━━━━━━━━━━ 32s 47ms/step - d_loss: 0.6635 - g_loss:
0.9306
Epoch 25/100
673/674 ━━━━━━━━━━━━━━━━━━━━ 0s 46ms/step - d_loss: 0.6651 - g_loss:
0.9243
```



```
674/674 ━━━━━━━━━━━━━━━━━━━━ 32s 47ms/step - d_loss: 0.6651 - g_loss:
0.9243
Epoch 26/100
674/674 ━━━━━━━━━━━━━━━━━━━━ 0s 46ms/step - d_loss: 0.6619 - g_loss:
0.9254
```

```
674/674 ━━━━━━━━━━━━━━━━━━━━ 32s 47ms/step - d_loss: 0.6619 - g_loss:
0.9254
Epoch 27/100
673/674 ━━━━━━━━━━━━━━━━━━━━ 0s 46ms/step - d_loss: 0.6569 - g_loss:
0.9313
```

```
674/674 ━━━━━━━━━━━━━━━━━━━━ 32s 47ms/step - d_loss: 0.6569 - g_loss:
0.9313
Epoch 28/100
674/674 ━━━━━━━━━━━━━━━━━━━━ 0s 46ms/step - d_loss: 0.6561 - g_loss:
0.9262
```

```
674/674 ━━━━━━━━━━━━━━━━━━━━ 32s 47ms/step - d_loss: 0.6561 - g_loss:
0.9262
Epoch 29/100
674/674 ━━━━━━━━━━━━━━━━━━━━ 0s 46ms/step - d_loss: 0.6525 - g_loss:
0.9295
```

```
674/674 ━━━━━━━━━━━━━━━━━━━━ 32s 47ms/step - d_loss: 0.6525 - g_loss:
0.9295
Epoch 30/100
673/674 ━━━━━━━━━━━━━━━━━━━━ 0s 46ms/step - d_loss: 0.6491 - g_loss:
0.9482
```

```
674/674 ━━━━━━━━━━━━━━━━━━━━ 32s 47ms/step - d_loss: 0.6491 - g_loss:
0.9482
Epoch 31/100
674/674 ━━━━━━━━━━━━━━━━━━━━ 0s 46ms/step - d_loss: 0.6448 - g_loss:
0.9466
```

```
674/674 ━━━━━━━━━━━━━━━━━━━━ 32s 47ms/step - d_loss: 0.6448 - g_loss:
0.9466
Epoch 32/100
674/674 ━━━━━━━━━━━━━━━━━━━━ 0s 46ms/step - d_loss: 0.6480 - g_loss:
0.9580
```

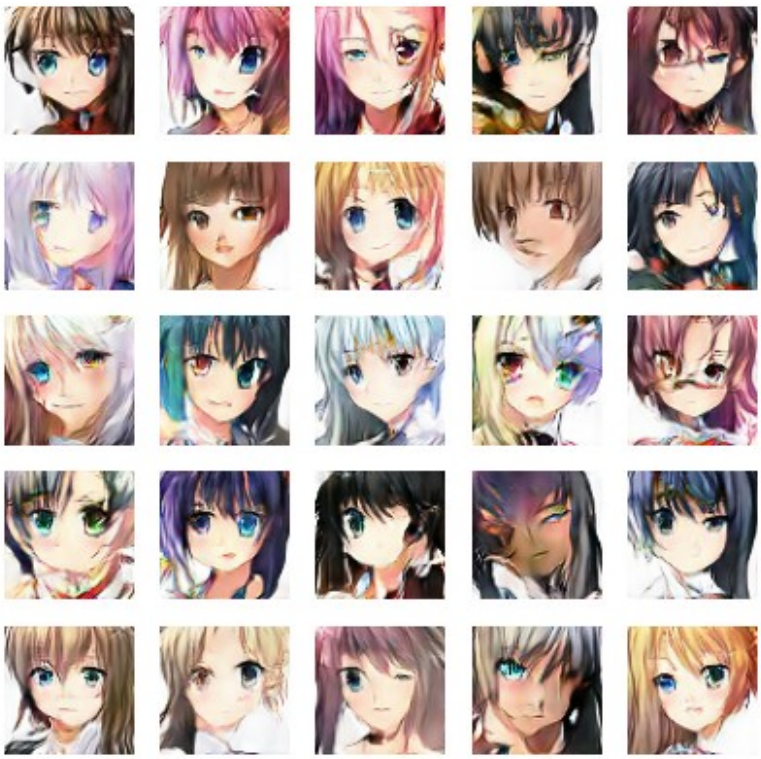
```
674/674 ━━━━━━━━━━━━━━━━━━━━ 32s 47ms/step - d_loss: 0.6480 - g_loss:
0.9580
Epoch 33/100
674/674 ━━━━━━━━━━━━━━━━━━━━ 0s 46ms/step - d_loss: 0.6403 - g_loss:
0.9529
```

```
674/674 ━━━━━━━━━━━━━━━━━━━━ 32s 47ms/step - d_loss: 0.6403 - g_loss:
0.9529
Epoch 34/100
674/674 ━━━━━━━━━━━━━━━━━━━━ 0s 46ms/step - d_loss: 0.6354 - g_loss:
0.9809
```

```
674/674 ━━━━━━━━━━━━━━━━━━━━ 31s 47ms/step - d_loss: 0.6354 - g_loss:
0.9809
Epoch 35/100
673/674 ━━━━━━━━━━━━━━━━━━━━ 0s 46ms/step - d_loss: 0.6342 - g_loss:
0.9860
```

```
674/674 ━━━━━━━━━━━━━━━━━━━━ 32s 47ms/step - d_loss: 0.6342 - g_loss:
0.9860
Epoch 36/100
674/674 ━━━━━━━━━━━━━━━━━━━━ 0s 46ms/step - d_loss: 0.6302 - g_loss:
0.9832
```

```
674/674 ━━━━━━━━━━━━━━━━━━━━ 32s 47ms/step - d_loss: 0.6302 - g_loss:
0.9832
Epoch 37/100
673/674 ━━━━━━━━━━━━━━━━━━━━ 0s 46ms/step - d_loss: 0.6260 - g_loss:
0.9927
```



```
674/674 ━━━━━━━━━━━━━━━━━━━━ 32s 47ms/step - d_loss: 0.6261 - g_loss:
0.9927
Epoch 38/100
673/674 ━━━━━━━━━━━━━━━━━━━━ 0s 46ms/step - d_loss: 0.6235 - g_loss:
1.0008
```

```
674/674 ━━━━━━━━━━━━━━━━━━━━ 32s 47ms/step - d_loss: 0.6235 - g_loss:
1.0008
Epoch 39/100
673/674 ━━━━━━━━━━━━━━━━━━━━ 0s 46ms/step - d_loss: 0.6240 - g_loss:
1.0134
```

```
674/674 ━━━━━━━━━━━━━━━━━━━━ 32s 47ms/step - d_loss: 0.6240 - g_loss:
1.0134
Epoch 40/100
674/674 ━━━━━━━━━━━━━━━━━━━━ 0s 46ms/step - d_loss: 0.6211 - g_loss:
1.0195
```

```
674/674 ━━━━━━━━━━━━━━━━━━━━ 32s 47ms/step - d_loss: 0.6211 - g_loss:
1.0196
Epoch 41/100
673/674 ━━━━━━━━━━━━━━━━━━━━ 0s 46ms/step - d_loss: 0.6072 - g_loss:
1.0451
```

```
674/674 ━━━━━━━━━━━━━━━━━━━━ 32s 47ms/step - d_loss: 0.6073 - g_loss:
1.0451
Epoch 42/100
674/674 ━━━━━━━━━━━━━━━━━━━━ 0s 46ms/step - d_loss: 0.6077 - g_loss:
1.0655
```

```
674/674 ━━━━━━━━━━━━━━━━━━━━ 32s 47ms/step - d_loss: 0.6077 - g_loss:
1.0655
Epoch 43/100
674/674 ━━━━━━━━━━━━━━━━━━━━ 0s 46ms/step - d_loss: 0.6033 - g_loss:
1.0693
```

```
674/674 ━━━━━━━━━━━━━━━━━━━━ 32s 47ms/step - d_loss: 0.6033 - g_loss:
1.0693
Epoch 44/100
673/674 ━━━━━━━━━━━━━━━━━━━━ 0s 46ms/step - d_loss: 0.5981 - g_loss:
1.0895
```

```
674/674 ━━━━━━━━━━━━━━━━━━━━ 32s 47ms/step - d_loss: 0.5981 - g_loss:
1.0895
Epoch 45/100
674/674 ━━━━━━━━━━━━━━━━━━━━ 0s 46ms/step - d_loss: 0.5874 - g_loss:
1.1033
```

```
674/674 ━━━━━━━━━━━━━━━━━━━━ 32s 47ms/step - d_loss: 0.5874 - g_loss:
1.1033
Epoch 46/100
673/674 ━━━━━━━━━━━━━━━━━━━━ 0s 46ms/step - d_loss: 0.5895 - g_loss:
1.1092
```

```
674/674 ━━━━━━━━━━━━━━━━━━━━ 32s 47ms/step - d_loss: 0.5895 - g_loss:
1.1092
Epoch 47/100
674/674 ━━━━━━━━━━━━━━━━━━━━ 0s 46ms/step - d_loss: 0.5860 - g_loss:
1.1248
```

```
674/674 ━━━━━━━━━━━━━━━━━━━━ 32s 47ms/step - d_loss: 0.5860 - g_loss:
1.1248
Epoch 48/100
673/674 ━━━━━━━━━━━━━━━━━━━━ 0s 46ms/step - d_loss: 0.5849 - g_loss:
1.1328
```

```
674/674 ━━━━━━━━━━━━━━━━━━━━ 32s 47ms/step - d_loss: 0.5849 - g_loss:
1.1328
Epoch 49/100
673/674 ━━━━━━━━━━━━━━━━━━━━ 0s 46ms/step - d_loss: 0.5823 - g_loss:
1.1525
```

```
674/674 ━━━━━━━━━━━━━━━━━━━━ 32s 47ms/step - d_loss: 0.5823 - g_loss:
1.1525
Epoch 50/100
673/674 ━━━━━━━━━━━━━━━━━━━━ 0s 46ms/step - d_loss: 0.5798 - g_loss:
1.1724
```

```
674/674 ━━━━━━━━━━━━━━━━━━━━ 32s 47ms/step - d_loss: 0.5798 - g_loss:
1.1724
Epoch 51/100
674/674 ━━━━━━━━━━━━━━━━━━━━ 0s 46ms/step - d_loss: 0.5747 - g_loss:
1.1741
```

```
674/674 ━━━━━━━━━━━━━━━━━━━━ 32s 47ms/step - d_loss: 0.5747 - g_loss:
1.1741
Epoch 52/100
674/674 ━━━━━━━━━━━━━━━━━━━━ 0s 46ms/step - d_loss: 0.5663 - g_loss:
1.1864
```

```
674/674 ━━━━━━━━━━━━━━━━━━━━ 32s 47ms/step - d_loss: 0.5663 - g_loss:
1.1864
Epoch 53/100
674/674 ━━━━━━━━━━━━━━━━━━━━ 0s 46ms/step - d_loss: 0.5682 - g_loss:
1.1966
```

```
674/674 ━━━━━━━━━━━━━━━━━━━━ 32s 47ms/step - d_loss: 0.5682 - g_loss:
1.1966
Epoch 54/100
674/674 ━━━━━━━━━━━━━━━━━━━━ 0s 46ms/step - d_loss: 0.5675 - g_loss:
1.1961
```

```
674/674 ━━━━━━━━━━━━━━━━━━━━ 32s 47ms/step - d_loss: 0.5675 - g_loss:
1.1961
Epoch 55/100
674/674 ━━━━━━━━━━━━━━━━━━━━ 0s 46ms/step - d_loss: 0.5613 - g_loss:
1.2141
```

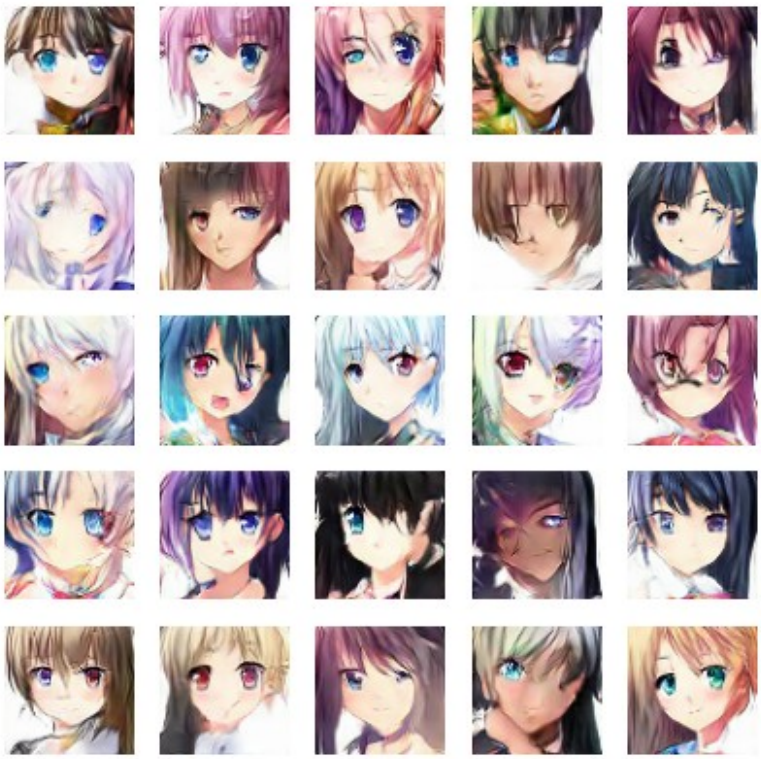
```
674/674 ━━━━━━━━━━━━━━━━━━━━ 32s 47ms/step - d_loss: 0.5613 - g_loss:
1.2141
Epoch 56/100
673/674 ━━━━━━━━━━━━━━━━━━━━ 0s 46ms/step - d_loss: 0.5582 - g_loss:
1.2376
```

```
674/674 ━━━━━━━━━━━━━━━━━━━━ 32s 48ms/step - d_loss: 0.5582 - g_loss:
1.2376
Epoch 57/100
674/674 ━━━━━━━━━━━━━━━━━━━━ 0s 46ms/step - d_loss: 0.5503 - g_loss:
1.2635
```

```
674/674 ━━━━━━━━━━━━━━━━━━━━ 32s 47ms/step - d_loss: 0.5503 - g_loss:
1.2635
Epoch 58/100
674/674 ━━━━━━━━━━━━━━━━━━━━ 0s 46ms/step - d_loss: 0.5509 - g_loss:
1.2440
```

```
674/674 ━━━━━━━━━━━━━━━━━━━━ 32s 47ms/step - d_loss: 0.5509 - g_loss:
1.2440
Epoch 59/100
673/674 ━━━━━━━━━━━━━━━━━━━━ 0s 46ms/step - d_loss: 0.5458 - g_loss:
1.2749
```

```
674/674 ━━━━━━━━━━━━━━━━━━━━ 32s 47ms/step - d_loss: 0.5459 - g_loss:
1.2749
Epoch 60/100
674/674 ━━━━━━━━━━━━━━━━━━━━ 0s 46ms/step - d_loss: 0.5440 - g_loss:
1.2983
```



```
674/674 ━━━━━━━━━━━━━━━━━━━━ 32s 47ms/step - d_loss: 0.5440 - g_loss:
1.2983
Epoch 61/100
674/674 ━━━━━━━━━━━━━━━━━━━━ 0s 46ms/step - d_loss: 0.5364 - g_loss:
1.3051
```

```
674/674 ━━━━━━━━━━━━━━━━━━━━ 32s 47ms/step - d_loss: 0.5364 - g_loss:
1.3051
Epoch 62/100
674/674 ━━━━━━━━━━━━━━━━━━━━ 0s 46ms/step - d_loss: 0.5399 - g_loss:
1.3116
```

```
674/674 ━━━━━━━━━━━━━━━━━━━━ 42s 48ms/step - d_loss: 0.5398 - g_loss:
1.3116
Epoch 63/100
674/674 ━━━━━━━━━━━━━━━━━━━━ 0s 46ms/step - d_loss: 0.5336 - g_loss:
1.3426
```

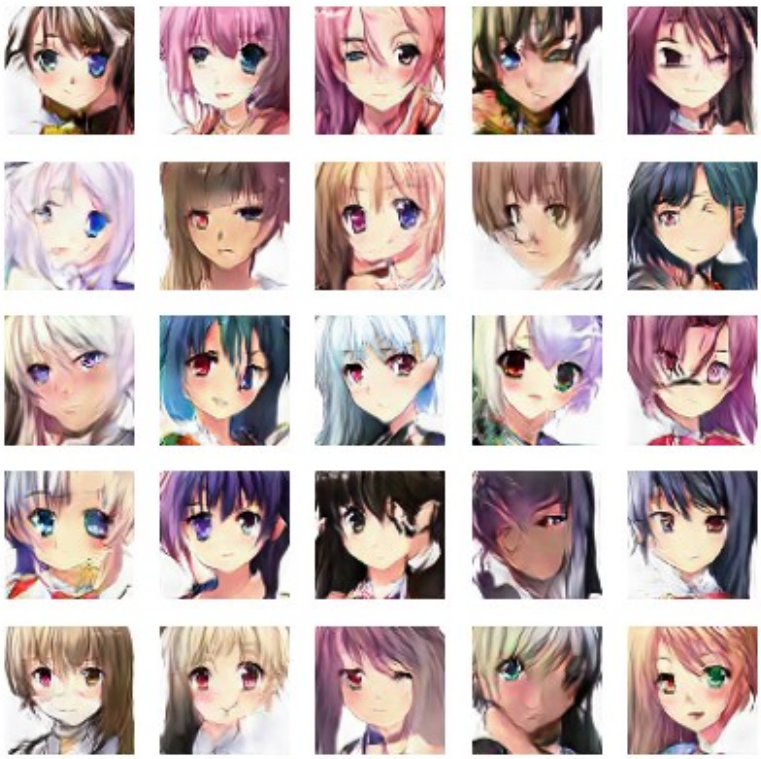
```
674/674 ━━━━━━━━━━━━━━━━━━━━ 32s 47ms/step - d_loss: 0.5336 - g_loss:
1.3426
Epoch 64/100
674/674 ━━━━━━━━━━━━━━━━━━━━ 0s 45ms/step - d_loss: 0.5262 - g_loss:
1.3516
```

```
674/674 ━━━━━━━━━━━━━━━━━━━━ 32s 47ms/step - d_loss: 0.5262 - g_loss:
1.3516
Epoch 65/100
674/674 ━━━━━━━━━━━━━━━━━━━━ 0s 46ms/step - d_loss: 0.5280 - g_loss:
1.3728
```

```
674/674 ━━━━━━━━━━━━━━━━━━━━ 32s 47ms/step - d_loss: 0.5280 - g_loss:
1.3728
Epoch 66/100
674/674 ━━━━━━━━━━━━━━━━━━━━ 0s 46ms/step - d_loss: 0.5234 - g_loss:
1.3804
```

```
674/674 ━━━━━━━━━━━━━━━━━━━━ 32s 48ms/step - d_loss: 0.5234 - g_loss:
1.3804
Epoch 67/100
673/674 ━━━━━━━━━━━━━━━━━━━━ 0s 46ms/step - d_loss: 0.5195 - g_loss:
1.3875
```

```
674/674 ━━━━━━━━━━━━━━━━━━━━ 32s 47ms/step - d_loss: 0.5195 - g_loss:
1.3875
Epoch 68/100
674/674 ━━━━━━━━━━━━━━━━━━━━ 0s 46ms/step - d_loss: 0.5107 - g_loss:
1.4035
```

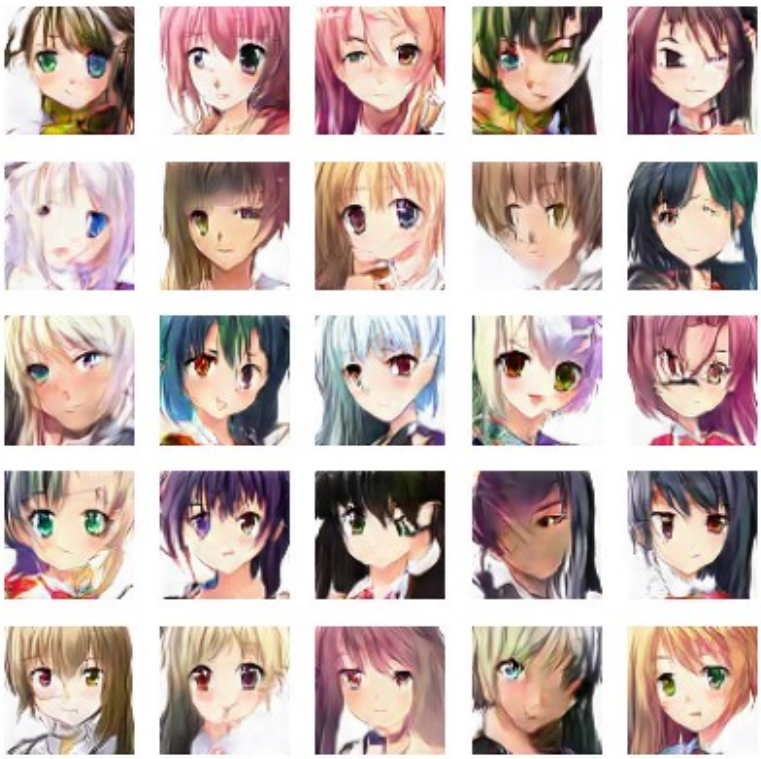


```
674/674 ━━━━━━━━━━━━━━━━━━━━ 32s 47ms/step - d_loss: 0.5107 - g_loss:
1.4034
Epoch 69/100
674/674 ━━━━━━━━━━━━━━━━━━━━ 0s 46ms/step - d_loss: 0.5158 - g_loss:
1.4214
```

```
674/674 ━━━━━━━━━━━━━━━━━━━━ 32s 47ms/step - d_loss: 0.5158 - g_loss:
1.4213
Epoch 70/100
674/674 ━━━━━━━━━━━━━━━━━━━━ 0s 46ms/step - d_loss: 0.5104 - g_loss:
1.4338
```

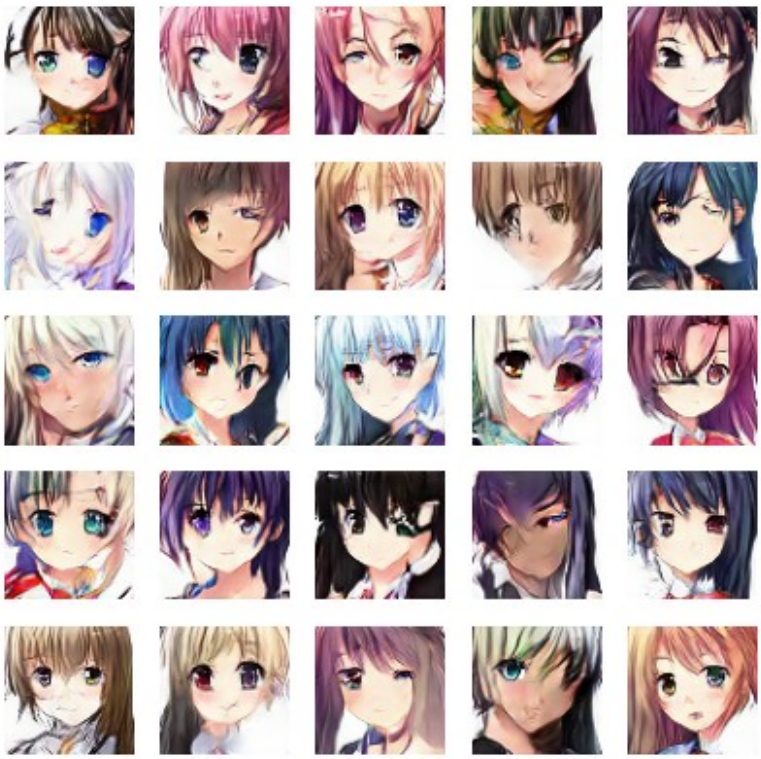
```
674/674 ━━━━━━━━━━━━━━━━━━━━ 32s 47ms/step - d_loss: 0.5104 - g_loss:
1.4337
Epoch 71/100
674/674 ━━━━━━━━━━━━━━━━━━━━ 0s 46ms/step - d_loss: 0.5063 - g_loss:
1.4374
```

```
674/674 ━━━━━━━━━━━━━━━━━━━━ 32s 47ms/step - d_loss: 0.5063 - g_loss:
1.4374
Epoch 72/100
674/674 ━━━━━━━━━━━━━━━━━━━━ 0s 46ms/step - d_loss: 0.5008 - g_loss:
1.4566
```



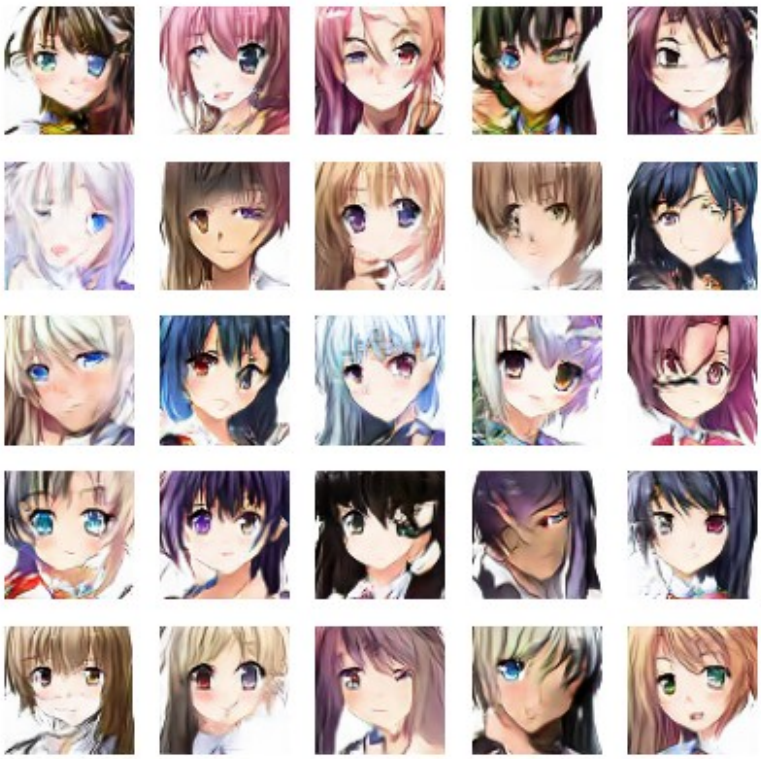
```
674/674 ━━━━━━━━━━━━━━━━━━━━ 32s 47ms/step - d_loss: 0.5008 - g_loss:
1.4566
Epoch 73/100
673/674 ━━━━━━━━━━━━━━━━━━━━ 0s 46ms/step - d_loss: 0.5025 - g_loss:
1.4721
```

```
674/674 ━━━━━━━━━━━━━━━━━━━━ 32s 47ms/step - d_loss: 0.5025 - g_loss:
1.4721
Epoch 74/100
674/674 ━━━━━━━━━━━━━━━━━━━━ 0s 46ms/step - d_loss: 0.4930 - g_loss:
1.5015
```



```
674/674 ━━━━━━━━━━━━━━━━━━━━ 32s 47ms/step - d_loss: 0.4930 - g_loss:
1.5015
Epoch 75/100
674/674 ━━━━━━━━━━━━━━━━━━━━ 0s 46ms/step - d_loss: 0.4874 - g_loss:
1.5250
```

```
674/674 ━━━━━━━━━━━━━━━━━━━━ 32s 47ms/step - d_loss: 0.4874 - g_loss:
1.5249
Epoch 76/100
674/674 ━━━━━━━━━━━━━━━━━━━━ 0s 46ms/step - d_loss: 0.4950 - g_loss:
1.5176
```



```
674/674 ━━━━━━━━━━━━━━━━━━━━ 32s 47ms/step - d_loss: 0.4950 - g_loss:
1.5176
Epoch 77/100
674/674 ━━━━━━━━━━━━━━━━━━━━ 0s 46ms/step - d_loss: 0.4846 - g_loss:
1.5337
```

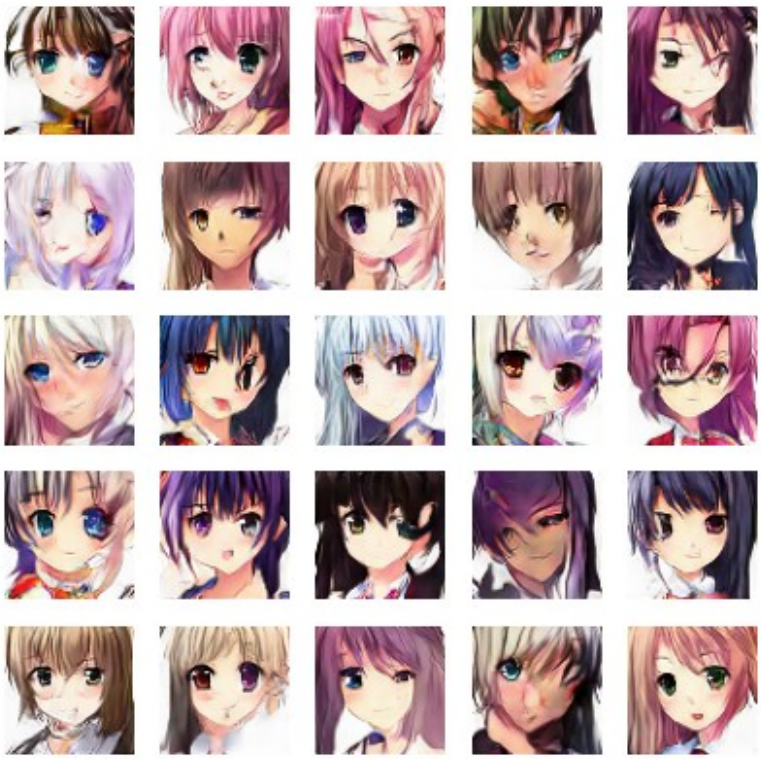
```
674/674 ━━━━━━━━━━━━━━━━━━━━ 32s 48ms/step - d_loss: 0.4846 - g_loss:
1.5337
Epoch 78/100
673/674 ━━━━━━━━━━━━━━━━━━━━ 0s 46ms/step - d_loss: 0.4850 - g_loss:
1.5650
```

```
674/674 ━━━━━━━━━━━━━━━━━━━━ 32s 47ms/step - d_loss: 0.4850 - g_loss:
1.5649
Epoch 79/100
674/674 ━━━━━━━━━━━━━━━━━━━━ 0s 46ms/step - d_loss: 0.4872 - g_loss:
1.5578
```

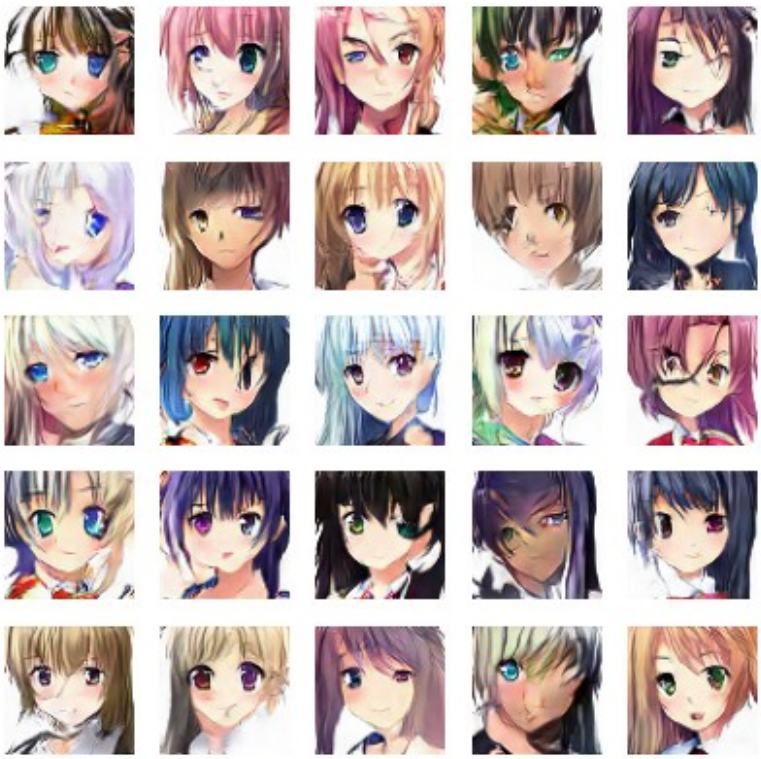
```
674/674 ━━━━━━━━━━━━━━━━━━━━ 32s 48ms/step - d_loss: 0.4872 - g_loss:
1.5578
Epoch 80/100
674/674 ━━━━━━━━━━━━━━━━━━━━ 0s 46ms/step - d_loss: 0.4777 - g_loss:
1.5808
```

```
674/674 ━━━━━━━━━━━━━━━━━━━━ 32s 47ms/step - d_loss: 0.4777 - g_loss:
1.5808
Epoch 81/100
674/674 ━━━━━━━━━━━━━━━━━━━━ 0s 46ms/step - d_loss: 0.4805 - g_loss:
1.5909
```

```
674/674 ━━━━━━━━━━━━━━━━━━━━ 32s 48ms/step - d_loss: 0.4805 - g_loss:
1.5909
Epoch 82/100
673/674 ━━━━━━━━━━━━━━━━━━━━ 0s 46ms/step - d_loss: 0.4726 - g_loss:
1.6019
```



```
674/674 ━━━━━━━━━━━━━━━━━━━━ 32s 47ms/step - d_loss: 0.4726 - g_loss:
1.6019
Epoch 83/100
674/674 ━━━━━━━━━━━━━━━━━━━━ 0s 46ms/step - d_loss: 0.4671 - g_loss:
1.6159
```



```
674/674 ━━━━━━━━━━━━━━━━━━━━ 32s 48ms/step - d_loss: 0.4671 - g_loss:
1.6159
Epoch 84/100
673/674 ━━━━━━━━━━━━━━━━━━━━ 0s 46ms/step - d_loss: 0.4663 - g_loss:
1.6458
```

```
674/674 ━━━━━━━━━━━━━━━━━━━━ 32s 47ms/step - d_loss: 0.4663 - g_loss:
1.6458
Epoch 85/100
674/674 ━━━━━━━━━━━━━━━━━━━━ 0s 46ms/step - d_loss: 0.4712 - g_loss:
1.6630
```

```
674/674 ━━━━━━━━━━━━━━━━━━━━ 32s 47ms/step - d_loss: 0.4712 - g_loss:
1.6630
Epoch 86/100
674/674 ━━━━━━━━━━━━━━━━━━━━ 0s 46ms/step - d_loss: 0.4633 - g_loss:
1.6725
```

```
674/674 ━━━━━━━━━━━━━━━━━━━━ 32s 48ms/step - d_loss: 0.4634 - g_loss:
1.6725
Epoch 87/100
674/674 ━━━━━━━━━━━━━━━━━━━━ 0s 46ms/step - d_loss: 0.4569 - g_loss:
1.6901
```

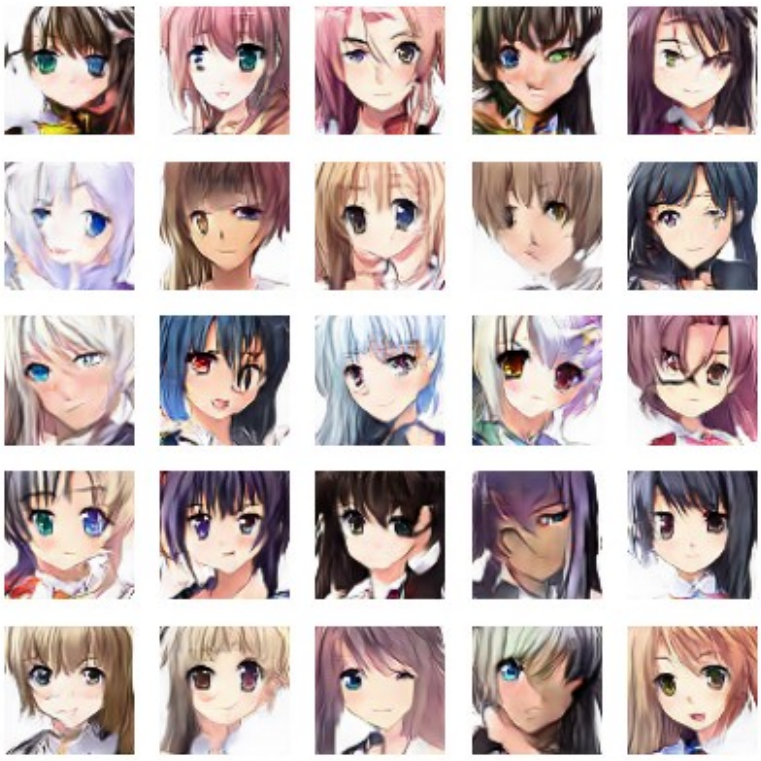
```
674/674 ━━━━━━━━━━━━━━━━━━━━ 32s 47ms/step - d_loss: 0.4569 - g_loss:
1.6901
Epoch 88/100
674/674 ━━━━━━━━━━━━━━━━━━━━ 0s 46ms/step - d_loss: 0.4622 - g_loss:
1.6892
```

```
674/674 ━━━━━━━━━━━━━━━━━━━━ 32s 47ms/step - d_loss: 0.4622 - g_loss:
1.6892
Epoch 89/100
673/674 ━━━━━━━━━━━━━━━━━━━━ 0s 46ms/step - d_loss: 0.4516 - g_loss:
1.6970
```

```
674/674 ━━━━━━━━━━━━━━━━━━━━ 32s 48ms/step - d_loss: 0.4516 - g_loss:
1.6970
Epoch 90/100
674/674 ━━━━━━━━━━━━━━━━━━━━ 0s 46ms/step - d_loss: 0.4500 - g_loss:
1.7353
```

```
674/674 ━━━━━━━━━━━━━━━━━━━━ 32s 47ms/step - d_loss: 0.4500 - g_loss:
1.7353
Epoch 91/100
674/674 ━━━━━━━━━━━━━━━━━━━━ 0s 46ms/step - d_loss: 0.4548 - g_loss:
1.7452
```

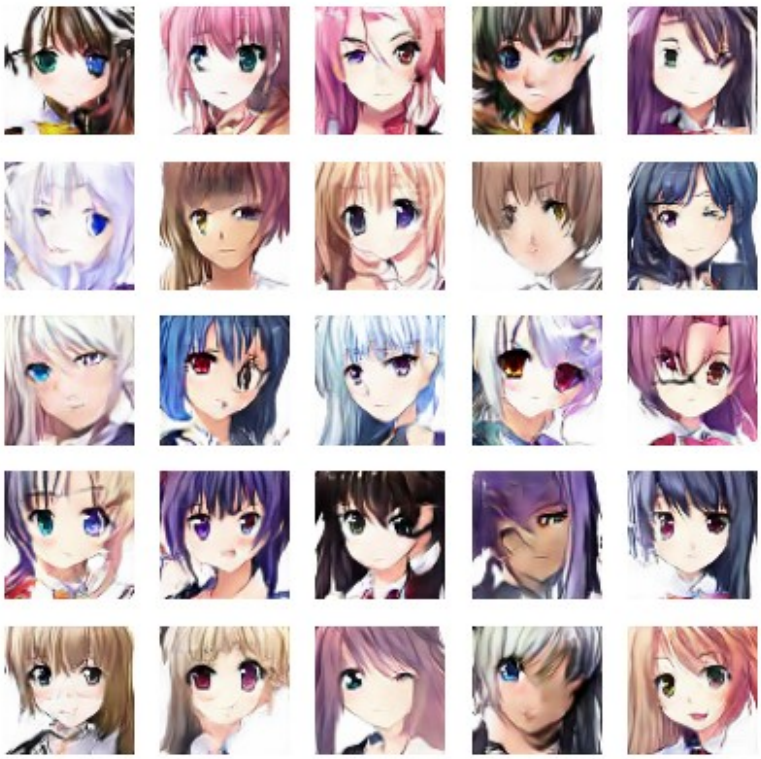
```
674/674 ━━━━━━━━━━━━━━━━━━━━ 32s 47ms/step - d_loss: 0.4548 - g_loss:
1.7451
Epoch 92/100
674/674 ━━━━━━━━━━━━━━━━━━━━ 0s 46ms/step - d_loss: 0.4523 - g_loss:
1.7443
```



```
674/674 ━━━━━━━━━━━━━━━━━━━━ 32s 47ms/step - d_loss: 0.4523 - g_loss:
1.7443
Epoch 93/100
673/674 ━━━━━━━━━━━━━━━━━━━━ 0s 46ms/step - d_loss: 0.4458 - g_loss:
1.7699
```

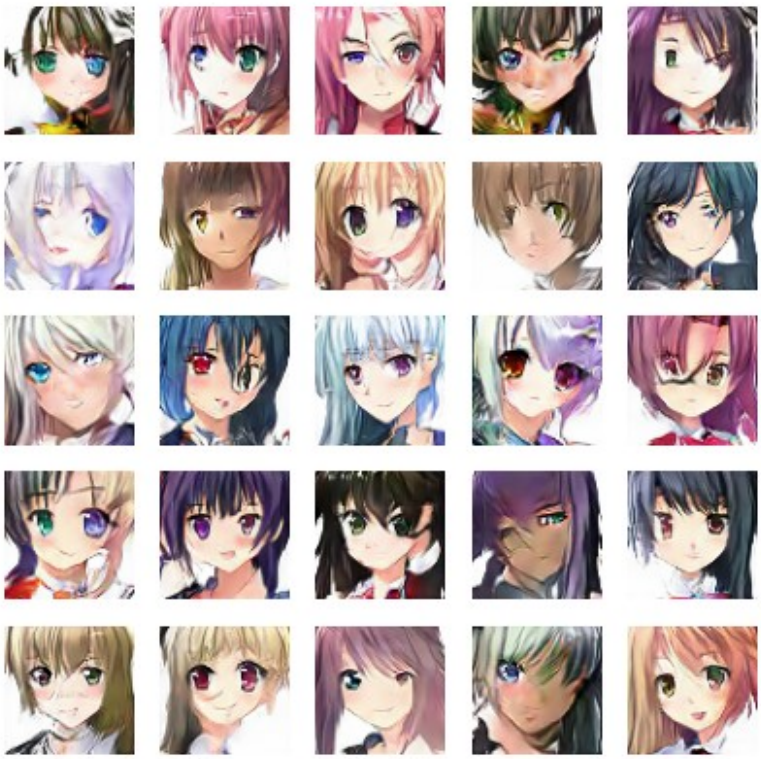
```
674/674 ━━━━━━━━━━━━━━━━━━━━ 41s 47ms/step - d_loss: 0.4458 - g_loss:
1.7699
Epoch 94/100
673/674 ━━━━━━━━━━━━━━━━━━━━ 0s 46ms/step - d_loss: 0.4456 - g_loss:
1.8003
```

```
674/674 ━━━━━━━━━━━━━━━━━━━━ 32s 47ms/step - d_loss: 0.4456 - g_loss:
1.8003
Epoch 95/100
674/674 ━━━━━━━━━━━━━━━━━━━━ 0s 46ms/step - d_loss: 0.4429 - g_loss:
1.7997
```



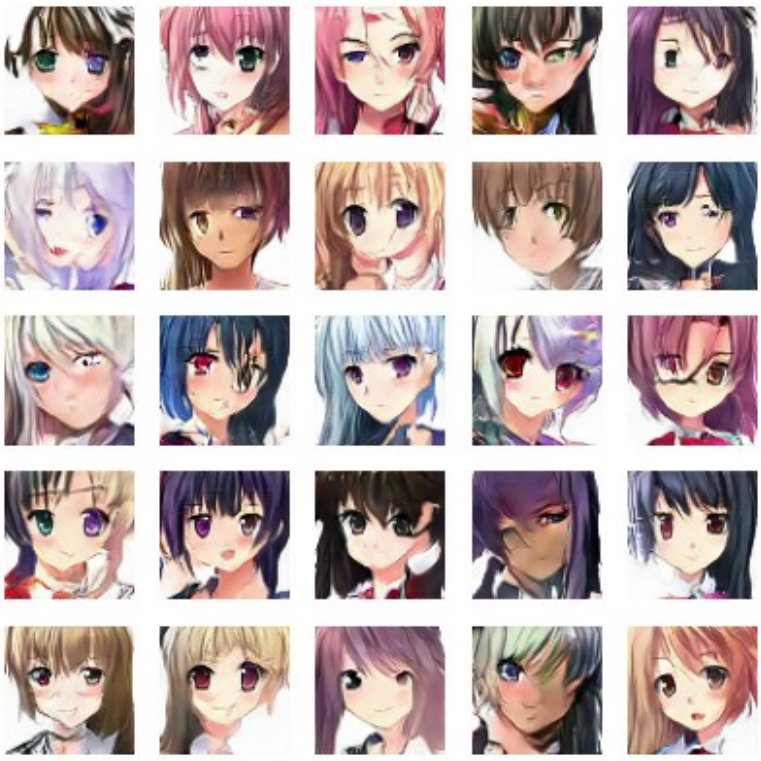
```
674/674 ━━━━━━━━━━━━━━━━━━━━ 32s 47ms/step - d_loss: 0.4429 - g_loss:
1.7997
Epoch 96/100
673/674 ━━━━━━━━━━━━━━━━━━━━ 0s 46ms/step - d_loss: 0.4361 - g_loss:
1.7967
```

```
674/674 ━━━━━━━━━━━━━━━━━━━━ 32s 47ms/step - d_loss: 0.4361 - g_loss:
1.7967
Epoch 97/100
673/674 ━━━━━━━━━━━━━━━━━━━━ 0s 46ms/step - d_loss: 0.4360 - g_loss:
1.8035
```



```
674/674 ━━━━━━━━━━━━━━━━━━━━ 32s 47ms/step - d_loss: 0.4360 - g_loss:
1.8035
Epoch 98/100
674/674 ━━━━━━━━━━━━━━━━━━━━ 0s 46ms/step - d_loss: 0.4327 - g_loss:
1.8510
```

```
674/674 ━━━━━━━━━━━━━━━━━━━━ 41s 47ms/step - d_loss: 0.4327 - g_loss:
1.8509
Epoch 99/100
674/674 ━━━━━━━━━━━━━━━━━━━━ 0s 46ms/step - d_loss: 0.4224 - g_loss:
1.8519
```



```
674/674 ━━━━━━━━━━━━━━━━━━━━ 32s 47ms/step - d_loss: 0.4224 - g_loss:
1.8519
Epoch 100/100
674/674 ━━━━━━━━━━━━━━━━━━━━ 0s 45ms/step - d_loss: 0.4323 - g_loss:
1.8593
```

```
674/674 ━━━━━━━━━━━━━━━━━━━━ 31s 47ms/step - d_loss: 0.4323 - g_loss: 1.8593

<keras.src.callbacks.history.History at 0x7fe82f4c7d30>
```

```
#This code generates and displays a single image using the trained DCGAN generator model.
noise=tf.random.normal([1,100])
fig=plt.figure(figsize=(2,3))
g_img=dcgan.generator(noise)
g_img=(g_img *127.5)+127.5
g_img.numpy()
img=array_to_img(g_img[0])
plt.imshow(img)
plt.axis('off')
plt.show()
```

```
#To display a series of generated images one by one with a specified
interval(2 sec)
import time

def display_generated_images(generator, latent_dim, num_images=5,
interval=2):
    for _ in range(num_images):
        # Generate random noise
        noise = tf.random.normal([1, latent_dim])
        fig=plt.figure(figsize=(2,2))
        # Generate image using the generator
        g_img = generator(noise, training=False)

        # Denormalize the image: from [-1, 1] to [0, 255]
        g_img = (g_img * 127.5) + 127.5

        # Convert tensor to numpy array and cast to uint8
        g_img = tf.clip_by_value(g_img, 0,
255).numpy().astype('uint8')

        # Convert array to image
        img = array_to_img(g_img[0])

        # Display the image
        plt.imshow(img)
        plt.axis('off')
        plt.show()

        # Wait for the specified interval before showing the next
image
        time.sleep(interval)

# Display 5 generated images at an interval of 2 seconds
display_generated_images(dcgan.generator, latent_dim, num_images=5,
interval=2)
```