yo-bro

July 12, 2024

[1]:
```
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 ImageDataGenerator,␣
↪array_to_img
from tensorflow.keras.models import Sequential
from tensorflow.keras import layers
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.losses import BinaryCrossentropy

warnings.filterwarnings('ignore')
```

[3]:
```
BASE_DIR = 'E:\\cbnst_lab\\GAN_proj\\images'
TARGET_SIZE = (64, 64)  # Set appropriate target size
BATCH_SIZE = 32
```

[5]:
```
# Create an ImageDataGenerator instance
datagen = ImageDataGenerator(rescale=1./127.5, preprocessing_function=lambda x:␣
↪(x - 1))
```

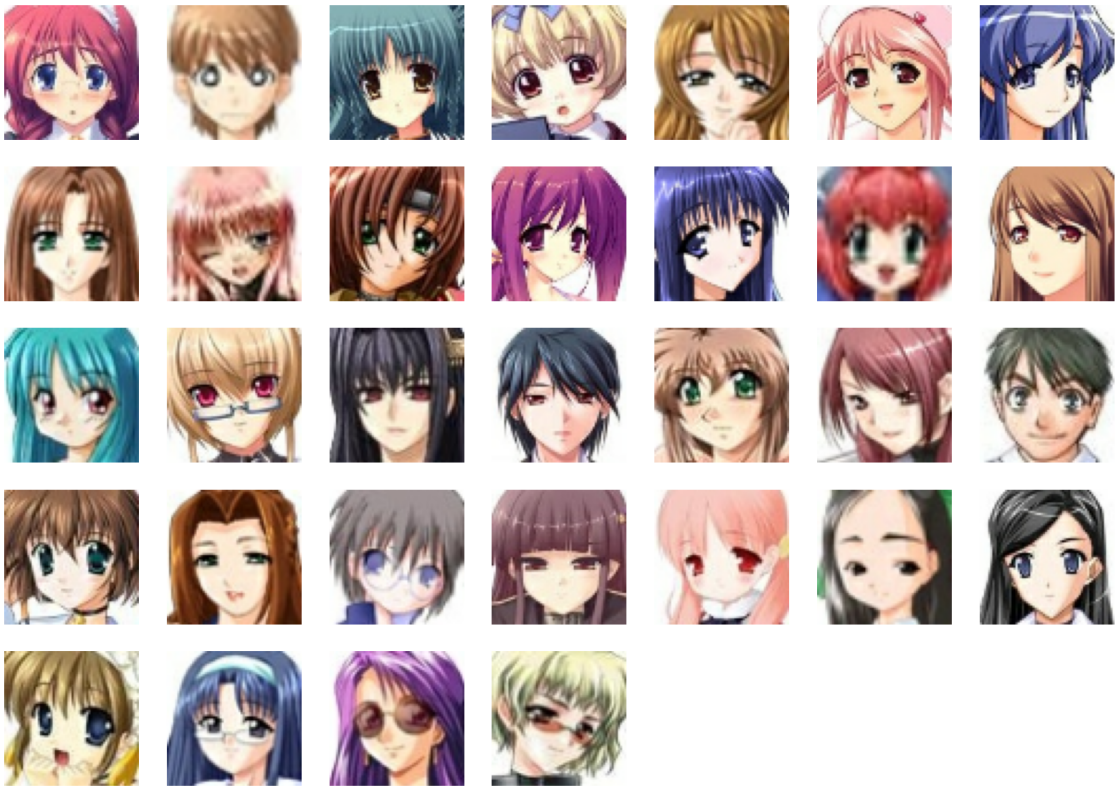
[13]:
```
# Load data from the directory
dataset = tf.keras.utils.image_dataset_from_directory(
    BASE_DIR,
    label_mode=None,  # No labels
    image_size=TARGET_SIZE,
    batch_size=BATCH_SIZE,
    shuffle=True
)
```

```
Found 21527 files.
```

[15]:
```
# Normalize the images
normalization_layer = tf.keras.layers.Rescaling(1./127.5, offset=-1)
```

```
dataset = dataset.map(lambda x: normalization_layer(x))
```

[17]:
```
# Get a batch of images
for batch in dataset.take(1):
    sample_images = batch.numpy()

plt.figure(figsize=(20, 20))
for i in range(min(len(sample_images), 49)):
    plt.subplot(7, 7, i+1)
    plt.imshow((sample_images[i] + 1) * 127.5 / 255)
    plt.axis('off')
plt.show()
```



[19]:
```
LATENT_DIM = 100
WEIGHT_INIT = keras.initializers.RandomNormal(mean=0.0, stddev=0.02)

# Generator
def build_generator():
    model = Sequential(name='generator')
    model.add(layers.Dense(8 * 8 * 512, input_dim=LATENT_DIM))
    model.add(layers.ReLU())
    model.add(layers.Reshape((8, 8, 512)))
```

```
    model.add(layers.Conv2DTranspose(256, (4, 4), strides=(2, 2),␣
 ↪padding='same', kernel_initializer=WEIGHT_INIT))
    model.add(layers.ReLU())
    model.add(layers.Conv2DTranspose(128, (4, 4), strides=(2, 2),␣
 ↪padding='same', kernel_initializer=WEIGHT_INIT))
    model.add(layers.ReLU())
    model.add(layers.Conv2DTranspose(64, (4, 4), strides=(2, 2),␣
 ↪padding='same', kernel_initializer=WEIGHT_INIT))
    model.add(layers.ReLU())
    model.add(layers.Conv2D(3, (4, 4), padding='same', activation='tanh'))
    return model

# Discriminator
def build_discriminator(input_shape):
    model = Sequential(name='discriminator')
    model.add(layers.Conv2D(64, (4, 4), strides=(2, 2), padding='same',␣
 ↪input_shape=input_shape))
    model.add(layers.LeakyReLU(alpha=0.2))
    model.add(layers.Conv2D(128, (4, 4), strides=(2, 2), padding='same'))
    model.add(layers.LeakyReLU(alpha=0.2))
    model.add(layers.Conv2D(256, (4, 4), strides=(2, 2), padding='same'))
    model.add(layers.LeakyReLU(alpha=0.2))
    model.add(layers.Flatten())
    model.add(layers.Dropout(0.3))
    model.add(layers.Dense(1, activation='sigmoid'))
    return model

generator = build_generator()
discriminator = build_discriminator((64, 64, 3))

generator.summary()
discriminator.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 (Conv2DTranspose)   (None, 16, 16, 256)                ␣
↪2,097,408

 re_lu_1 (ReLU)                       (None, 16, 16, 256)                ␣
↪  0

 conv2d_transpose_1 (Conv2DTranspose) (None, 32, 32, 128)                ␣
↪524,416

 re_lu_2 (ReLU)                       (None, 32, 32, 128)                ␣
↪  0

 conv2d_transpose_2 (Conv2DTranspose) (None, 64, 64, 64)                 ␣
↪131,136

 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)

Model: "discriminator"


 Layer (type)                         Output Shape                       ␣
↪Param #

 conv2d_1 (Conv2D)                    (None, 32, 32, 64)                 ␣
↪3,136

 leaky_re_lu (LeakyReLU)              (None, 32, 32, 64)                 ␣
↪  0
```

```
conv2d_2 (Conv2D)                    (None, 16, 16, 128)                ␣
↪131,200

leaky_re_lu_1 (LeakyReLU)            (None, 16, 16, 128)                ␣
↪  0

conv2d_3 (Conv2D)                    (None, 8, 8, 256)                  ␣
↪524,544

leaky_re_lu_2 (LeakyReLU)            (None, 8, 8, 256)                  ␣
↪  0

flatten (Flatten)                    (None, 16384)                      ␣
↪  0

dropout (Dropout)                    (None, 16384)                      ␣
↪  0

dense_1 (Dense)                      (None, 1)                          ␣
↪16,385
```

Total params: 675,265 (2.58 MB)

Trainable params: 675,265 (2.58 MB)

Non-trainable params: 0 (0.00 B)

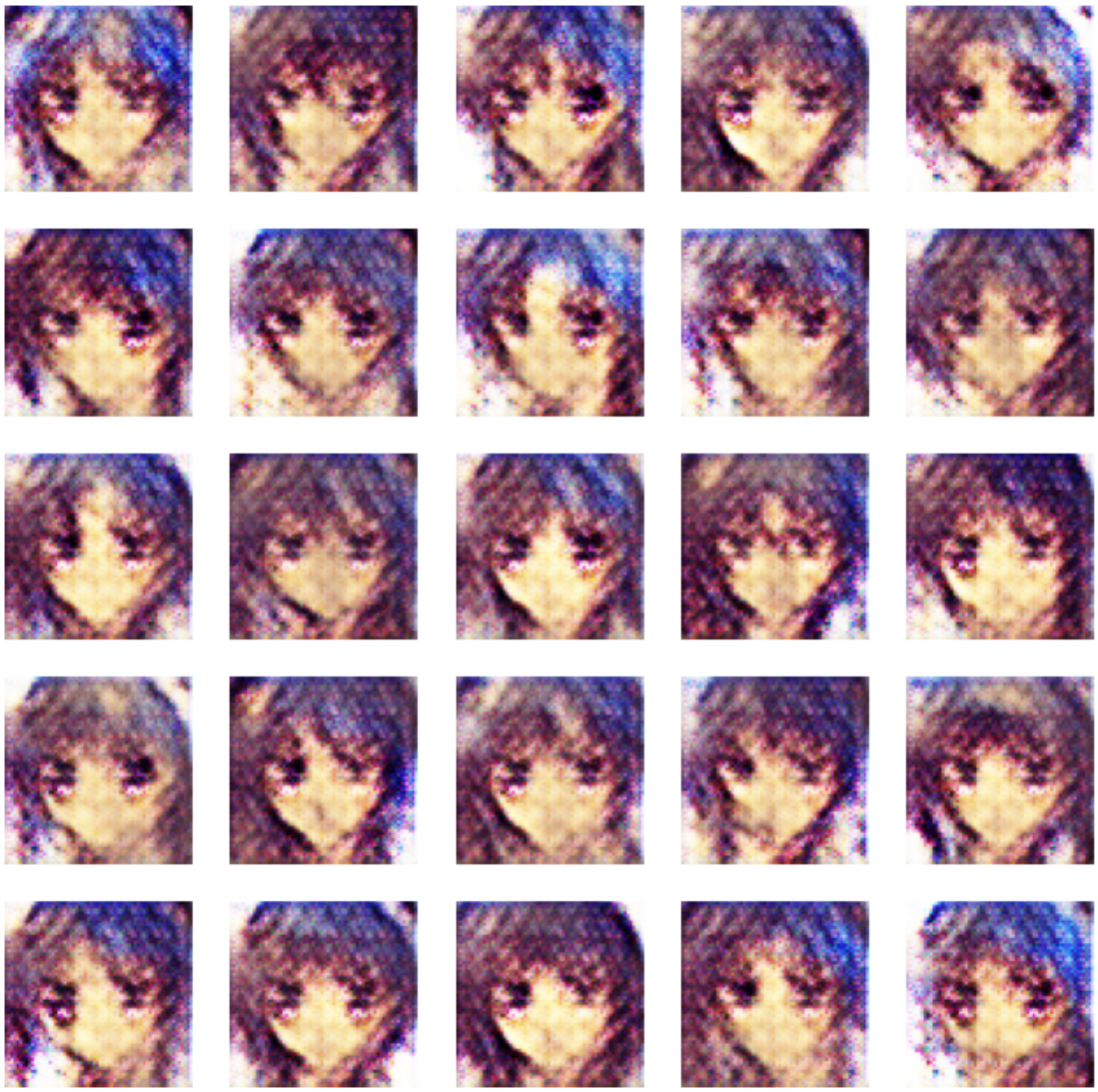
[21]:
```python
class DCGAN(keras.Model):
    def __init__(self, generator, discriminator, latent_dim):
        super().__init__()
        self.generator = generator
        self.discriminator = discriminator
        self.latent_dim = latent_dim
        self.g_loss_metric = keras.metrics.Mean(name='g_loss')
        self.d_loss_metric = keras.metrics.Mean(name='d_loss')

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

    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

    def train_step(self, real_images):
        batch_size = tf.shape(real_images)[0]
        random_noise = tf.random.normal(shape=(batch_size, self.latent_dim))

        with tf.GradientTape() as tape:
            pred_real = self.discriminator(real_images, training=True)
            real_labels = tf.ones((batch_size, 1)) * 0.9
            d_loss_real = self.loss_fn(real_labels, pred_real)

            fake_images = self.generator(random_noise)
            pred_fake = self.discriminator(fake_images, training=True)
            fake_labels = tf.zeros((batch_size, 1))
            d_loss_fake = self.loss_fn(fake_labels, pred_fake)

            d_loss = (d_loss_real + d_loss_fake) / 2

        gradients = tape.gradient(d_loss, self.discriminator.
↪trainable_variables)
        self.d_optimizer.apply_gradients(zip(gradients, self.discriminator.
↪trainable_variables))

        with tf.GradientTape() as tape:
            fake_images = self.generator(random_noise, training=True)
            pred_fake = self.discriminator(fake_images, training=True)
            g_loss = self.loss_fn(tf.ones((batch_size, 1)), pred_fake)

        gradients = tape.gradient(g_loss, self.generator.trainable_variables)
        self.g_optimizer.apply_gradients(zip(gradients, self.generator.
↪trainable_variables))

        self.d_loss_metric.update_state(d_loss)
        self.g_loss_metric.update_state(g_loss)

        return {'d_loss': self.d_loss_metric.result(), 'g_loss': self.
↪g_loss_metric.result()}
```

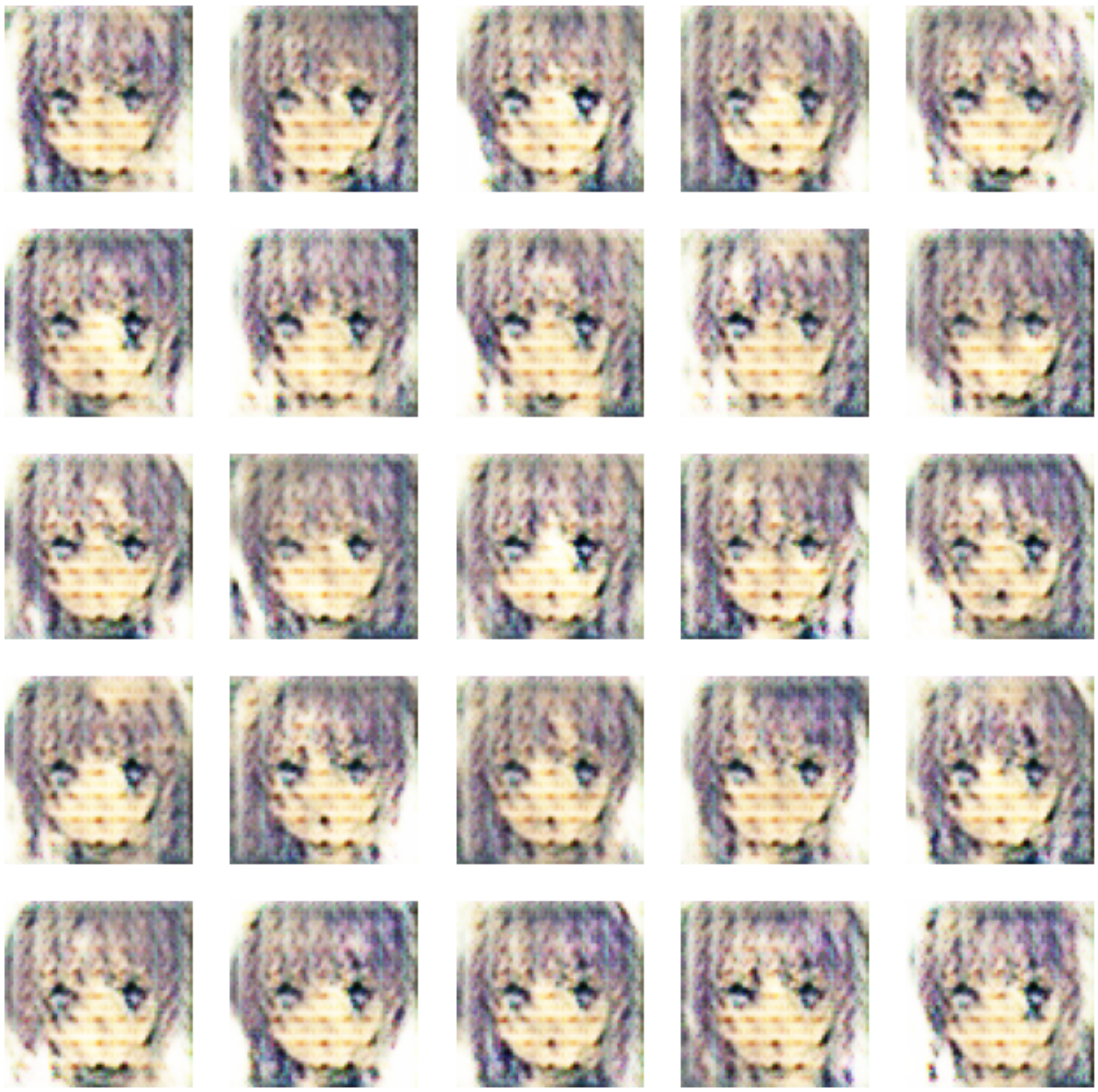
[25]:
```
class DCGANMonitor(keras.callbacks.Callback):
    def __init__(self, num_imgs=25, latent_dim=100):
        self.num_imgs = num_imgs
        self.latent_dim = latent_dim
        self.noise = tf.random.normal([25, latent_dim])

    def on_epoch_end(self, epoch, logs=None):
        g_img = self.model.generator(self.noise)
```

```
        g_img = (g_img * 127.5) + 127.5
        fig = plt.figure(figsize=(8, 8))
        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.show()

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

dcgan = DCGAN(generator=generator, discriminator=discriminator,␣
 ↪latent_dim=LATENT_DIM)

D_LR = 0.0001
G_LR = 0.0003
dcgan.compile(g_optimizer=Adam(learning_rate=G_LR, beta_1=0.5),␣
 ↪d_optimizer=Adam(learning_rate=D_LR, beta_1=0.5),␣
 ↪loss_fn=BinaryCrossentropy())

N_EPOCHS = 25
dcgan.fit(dataset, epochs=N_EPOCHS, callbacks=[DCGANMonitor()])
```

```
Epoch 1/25
673/673            0s 821ms/step -
d_loss: 0.6929 - g_loss: 0.8388
```



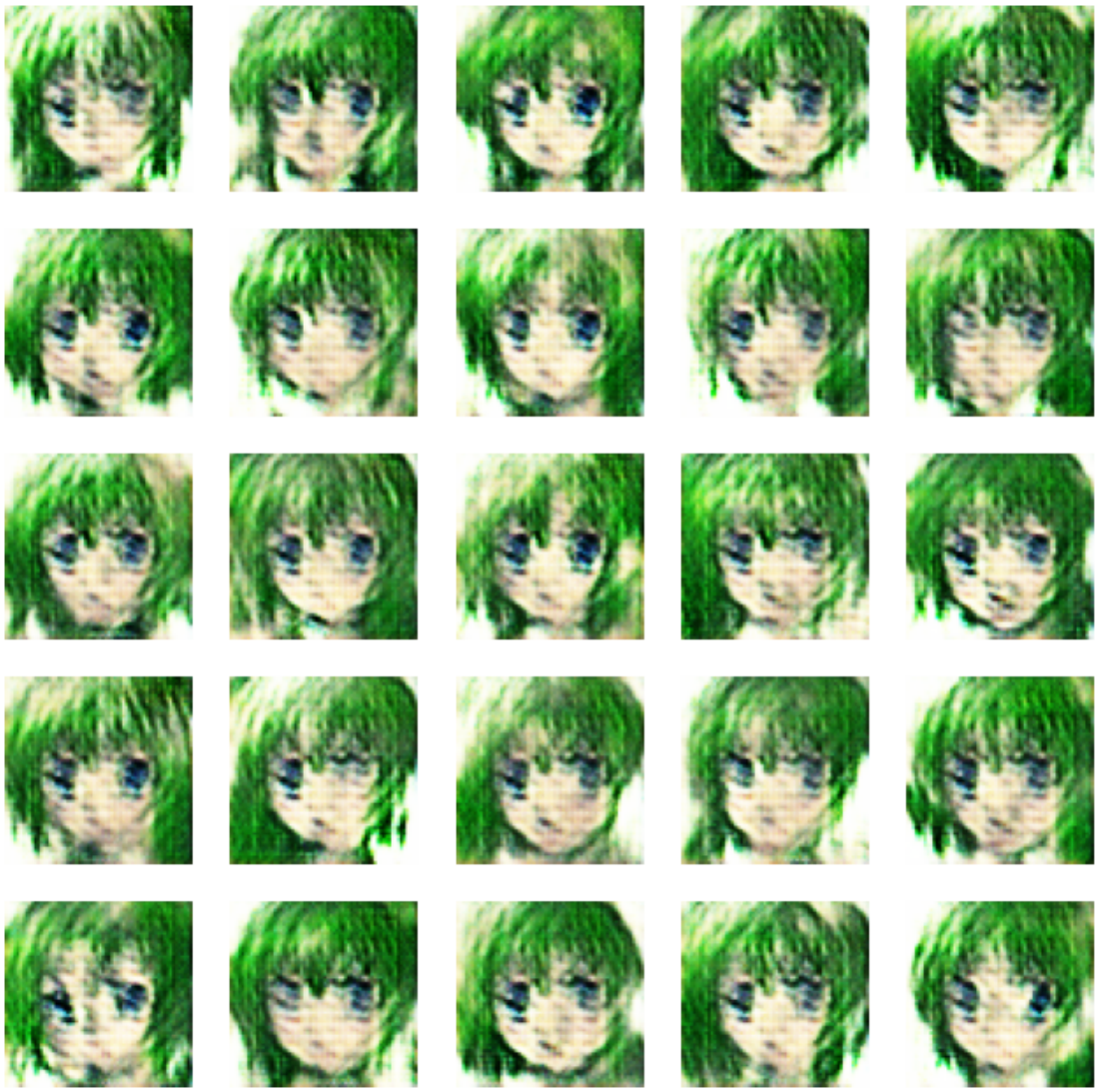
```
673/673            558s 822ms/step -
d_loss: 0.6929 - g_loss: 0.8388
Epoch 2/25
673/673            0s 872ms/step -
d_loss: 0.6891 - g_loss: 0.8621
```



```
673/673             587s 873ms/step -
d_loss: 0.6891 - g_loss: 0.8621
Epoch 3/25
673/673             0s 804ms/step -
d_loss: 0.6883 - g_loss: 0.8743
```

```
673/673             542s 805ms/step -
d_loss: 0.6883 - g_loss: 0.8743
Epoch 4/25
673/673             0s 816ms/step -
d_loss: 0.6893 - g_loss: 0.8570
```

```
673/673             550s 817ms/step -
d_loss: 0.6893 - g_loss: 0.8570
Epoch 5/25
673/673             0s 802ms/step -
d_loss: 0.6845 - g_loss: 0.8622
```



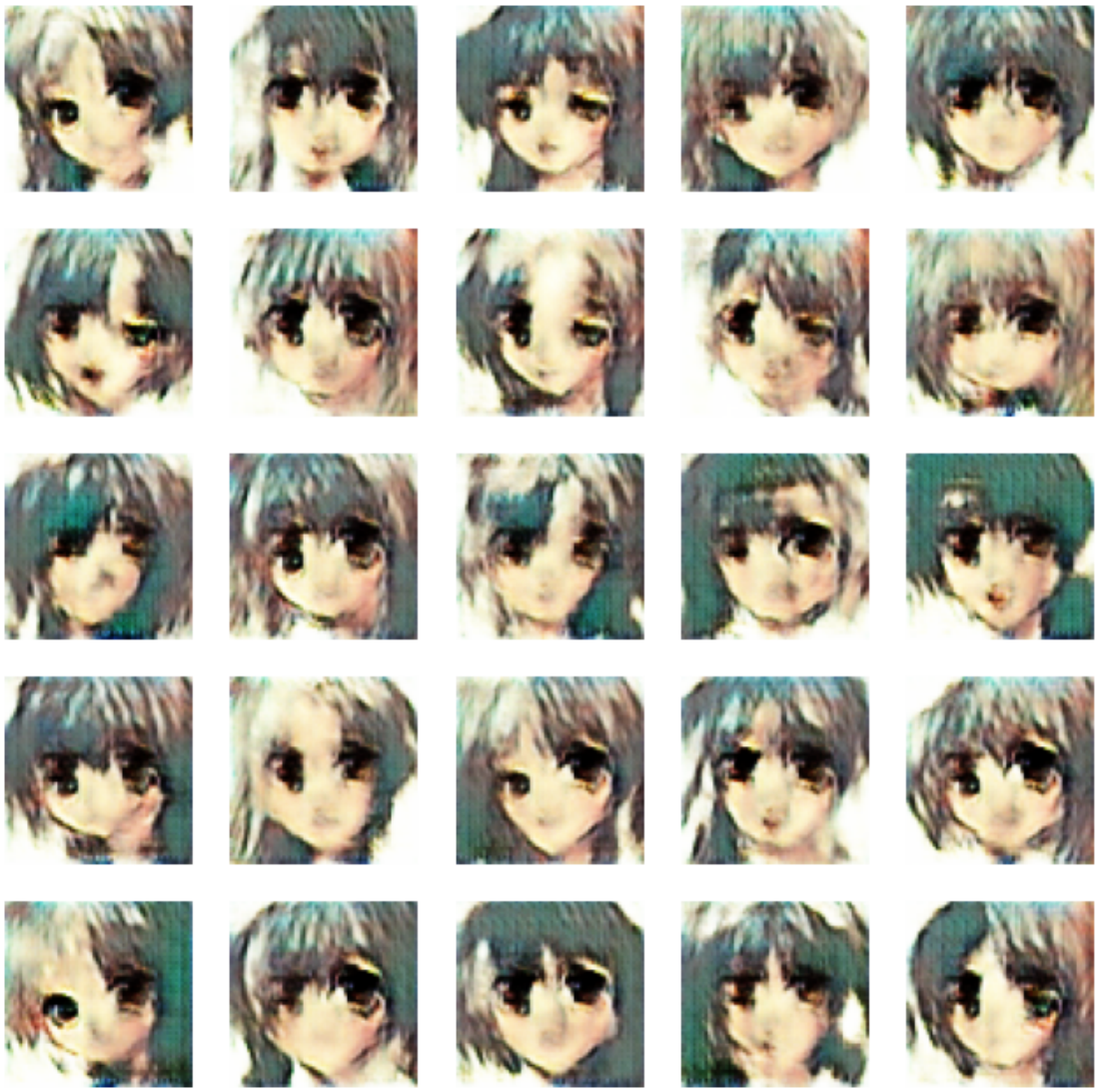
```
673/673             542s 806ms/step -
d_loss: 0.6845 - g_loss: 0.8622
Epoch 6/25
673/673             0s 801ms/step -
d_loss: 0.6789 - g_loss: 0.8996
```

```
673/673             539s 801ms/step -
d_loss: 0.6789 - g_loss: 0.8996
Epoch 7/25
673/673             0s 791ms/step -
d_loss: 0.6827 - g_loss: 0.9067
```

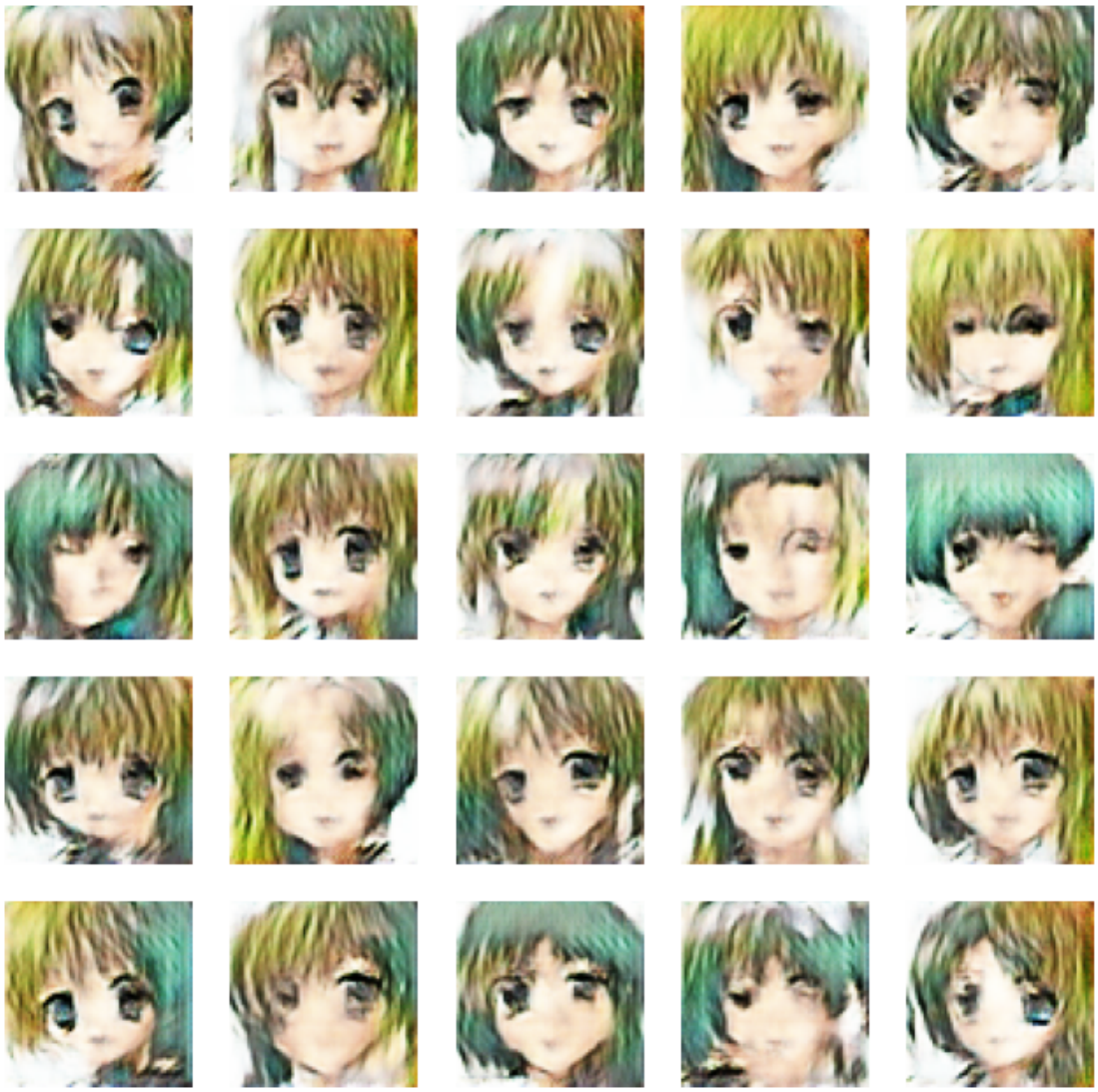
```
673/673             533s 792ms/step -
d_loss: 0.6827 - g_loss: 0.9067
Epoch 8/25
673/673             0s 775ms/step -
d_loss: 0.6800 - g_loss: 0.8994
```

```
673/673             522s 776ms/step -
d_loss: 0.6800 - g_loss: 0.8994
Epoch 9/25
673/673             0s 783ms/step -
d_loss: 0.6750 - g_loss: 0.9359
```

```
673/673             528s 784ms/step -
d_loss: 0.6750 - g_loss: 0.9359
Epoch 10/25
673/673             0s 810ms/step -
d_loss: 0.6624 - g_loss: 0.9789
```

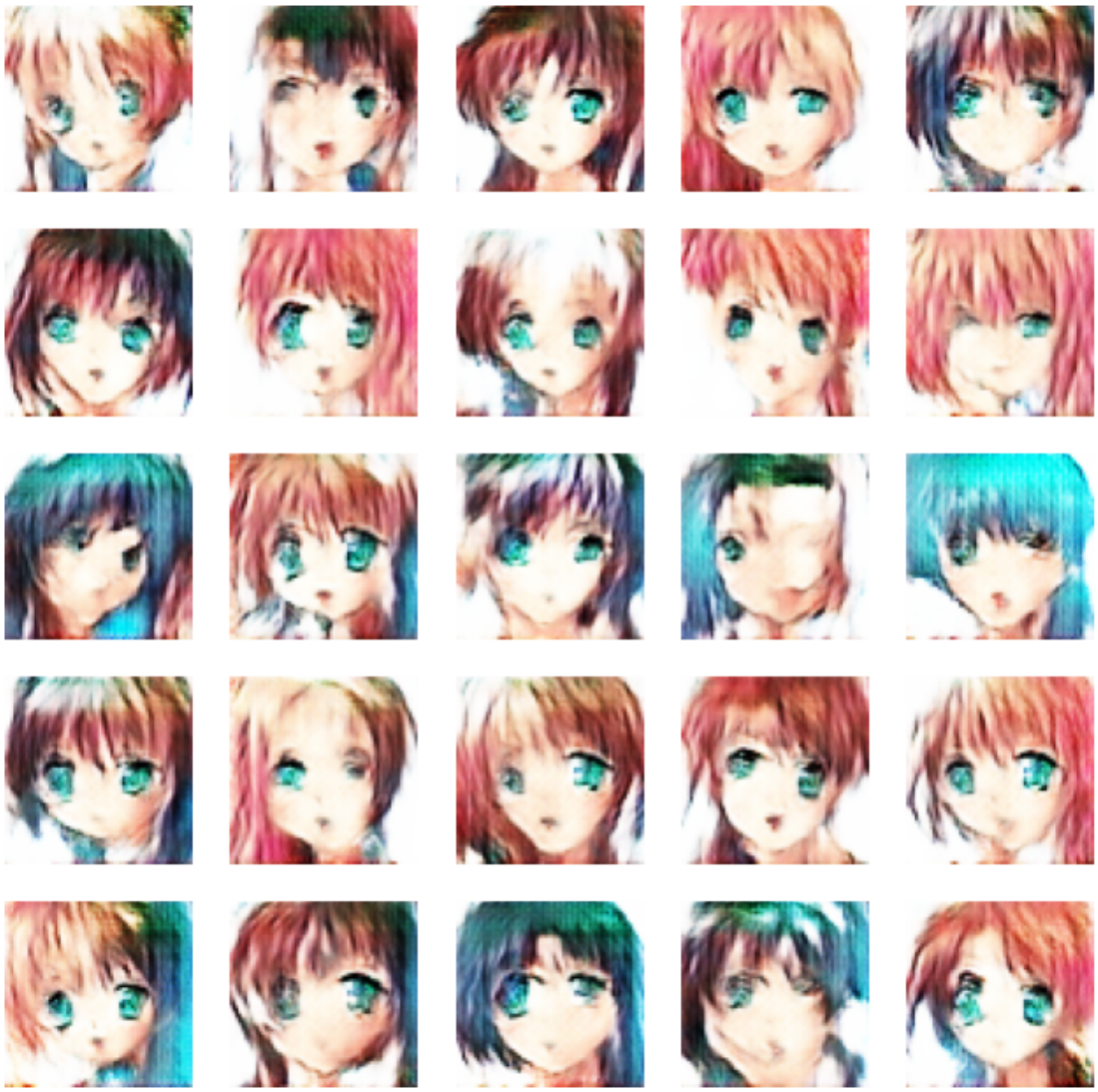


```
673/673             546s 812ms/step -
d_loss: 0.6624 - g_loss: 0.9789
Epoch 11/25
673/673             0s 808ms/step -
d_loss: 0.6664 - g_loss: 0.9581
```

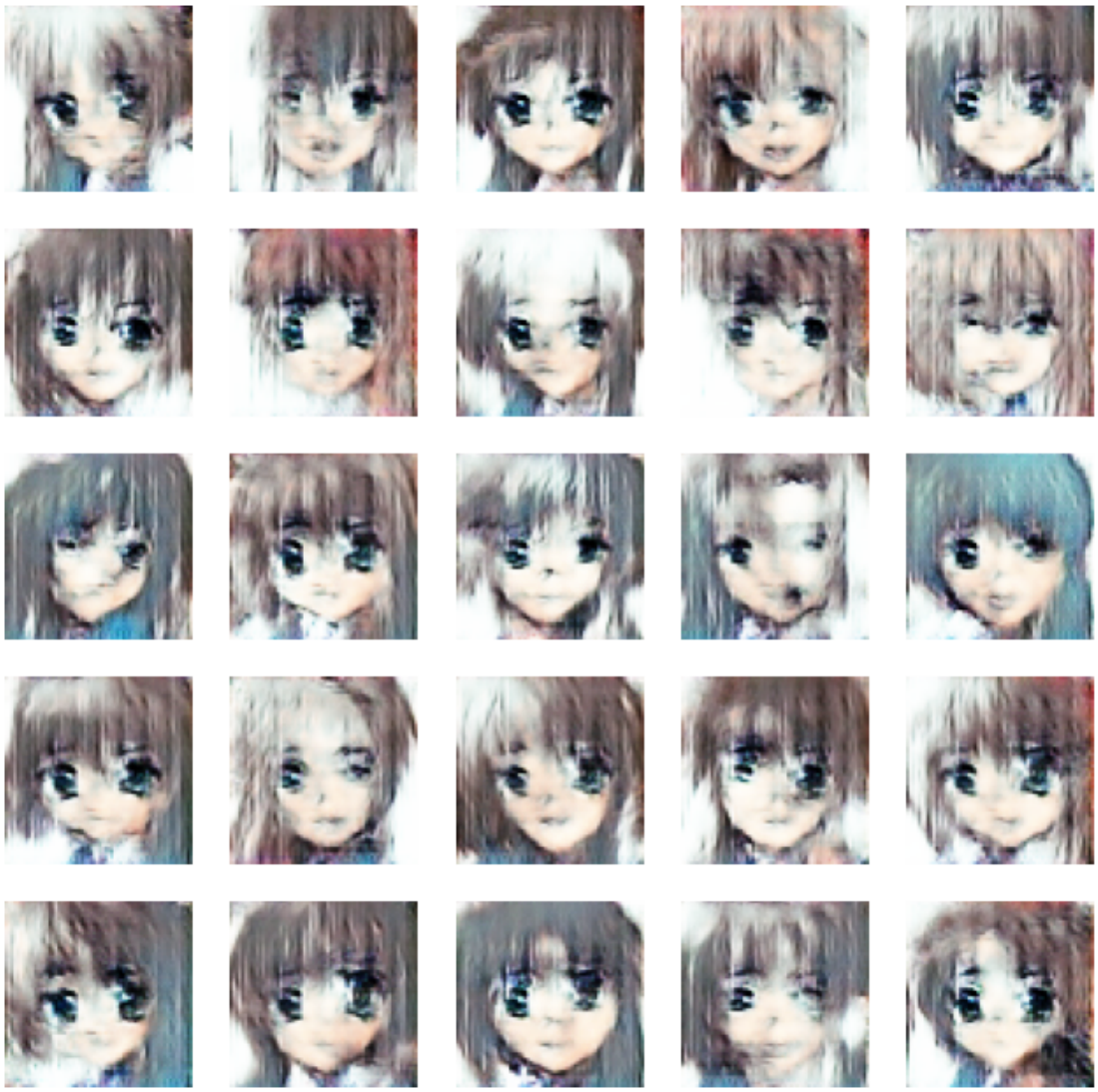


```
673/673             545s 809ms/step -
d_loss: 0.6664 - g_loss: 0.9581
Epoch 12/25
673/673             0s 816ms/step -
d_loss: 0.6717 - g_loss: 0.9402
```

```
673/673             550s 818ms/step -
d_loss: 0.6717 - g_loss: 0.9402
Epoch 13/25
673/673             0s 810ms/step -
d_loss: 0.6753 - g_loss: 0.9134
```



```
673/673              546s 811ms/step -
d_loss: 0.6753 - g_loss: 0.9134
Epoch 14/25
673/673              0s 809ms/step -
d_loss: 0.6761 - g_loss: 0.9063
```



```
673/673             547s 813ms/step -
d_loss: 0.6761 - g_loss: 0.9063
Epoch 15/25
673/673             0s 809ms/step -
d_loss: 0.6673 - g_loss: 0.9376
```

```
673/673             546s 810ms/step -
d_loss: 0.6673 - g_loss: 0.9376
Epoch 16/25
673/673             0s 796ms/step -
d_loss: 0.6683 - g_loss: 0.9356
```

```
673/673             537s 797ms/step -
d_loss: 0.6683 - g_loss: 0.9357
Epoch 17/25
673/673             0s 795ms/step -
d_loss: 0.6656 - g_loss: 0.9408
```

```
673/673              536s 796ms/step -
d_loss: 0.6656 - g_loss: 0.9408
Epoch 18/25
673/673              0s 805ms/step -
d_loss: 0.6666 - g_loss: 0.9380
```

```
673/673             543s 806ms/step -
d_loss: 0.6666 - g_loss: 0.9381
Epoch 19/25
673/673             0s 816ms/step -
d_loss: 0.6663 - g_loss: 0.9235
```

```
673/673             551s 819ms/step -
d_loss: 0.6663 - g_loss: 0.9235
Epoch 20/25
673/673             0s 805ms/step -
d_loss: 0.6674 - g_loss: 0.9313
```

```
673/673             543s 806ms/step -
d_loss: 0.6674 - g_loss: 0.9313
Epoch 21/25
673/673             0s 807ms/step -
d_loss: 0.6686 - g_loss: 0.9288
```

```
673/673             544s 808ms/step -
d_loss: 0.6686 - g_loss: 0.9288
Epoch 22/25
673/673             0s 813ms/step -
d_loss: 0.6687 - g_loss: 0.9344
```

```
673/673             548s 814ms/step -
d_loss: 0.6687 - g_loss: 0.9344
Epoch 23/25
673/673             0s 806ms/step -
d_loss: 0.6687 - g_loss: 0.9276
```

```
673/673             543s 807ms/step -
d_loss: 0.6687 - g_loss: 0.9276
Epoch 24/25
673/673             0s 812ms/step -
d_loss: 0.6676 - g_loss: 0.9338
```

```
673/673             547s 813ms/step -
d_loss: 0.6676 - g_loss: 0.9338
Epoch 25/25
673/673             0s 821ms/step -
d_loss: 0.6658 - g_loss: 0.9372
```

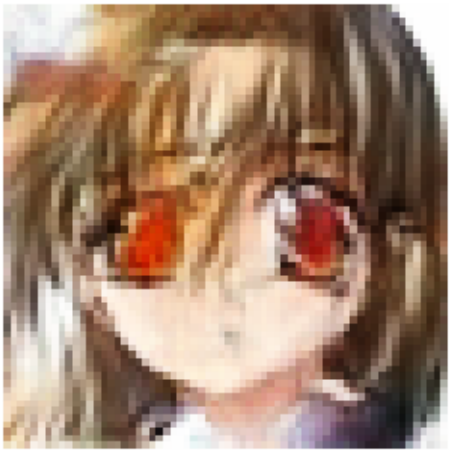
```
673/673              554s 822ms/step -
d_loss: 0.6658 - g_loss: 0.9372
```

```
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or
`keras.saving.save_model(model)`. This file format is considered legacy. We
recommend using instead the native Keras format, e.g.
`model.save('my_model.keras')` or `keras.saving.save_model(model,
'my_model.keras')`.
```

[25]:
```
<keras.src.callbacks.history.History at 0x27dcb553170>
```

[31]:
```
noise = tf.random.normal([1, 100])
fig = plt.figure(figsize=(3, 3))
g_img = dcgan.generator(noise)
```

```
g_img = (g_img * 127.5) + 127.5
img = array_to_img(g_img[0])
plt.imshow(img)
plt.axis('off')
plt.show()
```



[ ]: