```
!mkdir ~/.kaggle
```

```
!cp /content/kaggle.json ~/.kaggle/
```

```
!kaggle datasets download -d joosthazelzet/lego-brick-images
```

```
Warning: Your Kaggle API key is readable by other users on this system! To fix this, you can run 'chmod 600 /root/.kaggle/kaggle.json'
Downloading lego-brick-images.zip to /content
100% 1.00G/1.00G [00:38<00:00, 36.6MB/s]
100% 1.00G/1.00G [00:38<00:00, 27.7MB/s]
```

```
!unzip /content/lego-brick-images.zip
```

```
# %load_ext autoreload
# %autoreload 2
import numpy as np
import matplotlib.pyplot as plt

import tensorflow as tf
from tensorflow.keras import (
    layers,
    models,
    callbacks,
    losses,
    utils,
    metrics,
    optimizers,
)

from drive.MyDrive.GAN_Datasets.utils import display, sample_batch
```

```
IMAGE_SIZE = 64
CHANNELS = 1
BATCH_SIZE = 128
Z_DIM = 100
EPOCHS = 300
LOAD_MODEL = False
ADAM_BETA_1 = 0.5
ADAM_BETA_2 = 0.999
LEARNING_RATE = 0.0002
NOISE_PARAM = 0.1
```

```
#  Prepare the data
train_data = utils.image_dataset_from_directory(
    "/content/LEGO brick images v1",
    labels=None,
    color_mode="grayscale",
    image_size=(IMAGE_SIZE, IMAGE_SIZE),
    batch_size=BATCH_SIZE,
    shuffle=True,
    seed=42,
    interpolation="bilinear",
)
```

```
Found 6379 files belonging to 1 classes.
```

```
def preprocess(img):
  img = (tf.cast(img, "float32")- 127.5)/127.5
  return img

train = train_data.map(lambda x:preprocess(x))
```

```
train_sample = sample_batch(train)
display(train_sample)
```

Build the GAN

```
discriminator_input  = layers.Input(shape=(IMAGE_SIZE, IMAGE_SIZE, CHANNELS))
x = layers.Conv2D(64, kernel_size=4, strides=2, padding='same', use_bias=False)(discriminator_input)
x = layers.LeakyReLU(0.2)(x)
x = layers.Dropout(0.3)(x)
x = layers.Conv2D(128, kernel_size=4, strides=2, padding='same', use_bias=False)(x)
x = layers.BatchNormalization(momentum=0.9)(x)
x = layers.LeakyReLU(0.2)(x)
x = layers.Dropout(0.3)(x)
x = layers.Conv2D(256, kernel_size=4, strides=2, padding='same', use_bias=False)(x)
x  = layers.BatchNormalization(momentum=0.9)(x)
x = layers.LeakyReLU(0.2)(x)
x = layers.Dropout(0.3)(x)
x = layers.Conv2D(512, kernel_size=4, strides=2, padding='same', use_bias=False)(x)
x  = layers.BatchNormalization(momentum=0.9)(x)
x = layers.LeakyReLU(0.2)(x)
x = layers.Dropout(0.3)(x)
x = layers.Conv2D(1, kernel_size=4, strides=1, padding='valid', use_bias=False, activation='sigmoid',)(x)
discriminator_output = layers.Flatten()(x)
discriminator = models.Model(discriminator_input, discriminator_output)

discriminator.summary()
```

```
Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_1 (InputLayer)        [(None, 64, 64, 1)]       0

 conv2d (Conv2D)             (None, 32, 32, 64)        1024

 leaky_re_lu (LeakyReLU)     (None, 32, 32, 64)        0

 dropout (Dropout)           (None, 32, 32, 64)        0

 conv2d_1 (Conv2D)           (None, 16, 16, 128)       131072

 batch_normalization (BatchN  (None, 16, 16, 128)      512
 ormalization)

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

 dropout_1 (Dropout)         (None, 16, 16, 128)       0

 conv2d_2 (Conv2D)           (None, 8, 8, 256)         524288

 batch_normalization_1 (Batc  (None, 8, 8, 256)        1024
 hNormalization)

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

 dropout_2 (Dropout)         (None, 8, 8, 256)         0

 conv2d_3 (Conv2D)           (None, 4, 4, 512)         2097152

 batch_normalization_2 (Batc  (None, 4, 4, 512)        2048
 hNormalization)

 leaky_re_lu_3 (LeakyReLU)   (None, 4, 4, 512)         0

 dropout_3 (Dropout)         (None, 4, 4, 512)         0

 conv2d_4 (Conv2D)           (None, 1, 1, 1)           8192

 flatten (Flatten)           (None, 1)                 0

=================================================================
Total params: 2,765,312
Trainable params: 2,763,520
```

```
Non-trainable params: 1,792
_________________________________________________________________
```

```
generator_input = layers.Input(shape=(Z_DIM,))
x = layers.Reshape((1,1,Z_DIM))(generator_input)
x = layers.Conv2DTranspose(512, kernel_size=4, strides=1, padding='valid', use_bias=False)(x)
x = layers.BatchNormalization(momentum=0.9)(x)
x = layers.LeakyReLU(0.2)(x)
x = layers.Conv2DTranspose(256, kernel_size=4, strides=2, padding='same', use_bias=False)(x)
x = layers.BatchNormalization(momentum=0.9)(x)
x = layers.LeakyReLU(0.2)(x)
x = layers.Conv2DTranspose(128, kernel_size=4, strides=2, padding='same', use_bias=False)(x)
x = layers.BatchNormalization(momentum=0.9)(x)
x = layers.LeakyReLU(0.2)(x)
x = layers.Conv2DTranspose(64, kernel_size=4, strides=2, padding='same', use_bias=False)(x)
x = layers.BatchNormalization(momentum=0.9)(x)
x = layers.LeakyReLU(0.2)(x)
generator_output = layers.Conv2DTranspose(CHANNELS, kernel_size=4, strides=2, padding='same', use_bias=False, activation='tanh')(x)
generator = models.Model(generator_input, generator_output)
generator.summary()
```

```
Model: "model_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_2 (InputLayer)        [(None, 100)]             0

 reshape (Reshape)           (None, 1, 1, 100)         0

 conv2d_transpose (Conv2DTra  (None, 4, 4, 512)        819200
 nspose)

 batch_normalization_3 (Batc  (None, 4, 4, 512)        2048
 hNormalization)

 leaky_re_lu_4 (LeakyReLU)   (None, 4, 4, 512)         0

 conv2d_transpose_1 (Conv2DT  (None, 8, 8, 256)        2097152
 ranspose)

 batch_normalization_4 (Batc  (None, 8, 8, 256)        1024
 hNormalization)

 leaky_re_lu_5 (LeakyReLU)   (None, 8, 8, 256)         0

 conv2d_transpose_2 (Conv2DT  (None, 16, 16, 128)      524288
 ranspose)

 batch_normalization_5 (Batc  (None, 16, 16, 128)      512
 hNormalization)

 leaky_re_lu_6 (LeakyReLU)   (None, 16, 16, 128)       0

 conv2d_transpose_3 (Conv2DT  (None, 32, 32, 64)       131072
 ranspose)

 batch_normalization_6 (Batc  (None, 32, 32, 64)       256
 hNormalization)

 leaky_re_lu_7 (LeakyReLU)   (None, 32, 32, 64)        0

 conv2d_transpose_4 (Conv2DT  (None, 64, 64, 1)        1024
 ranspose)

=================================================================
Total params: 3,576,576
Trainable params: 3,574,656
Non-trainable params: 1,920
_________________________________________________________________
```

```
class DCGAN(models.Model):
    def __init__(self, discriminator, generator, latent_dim):
        super(DCGAN, self).__init__()
        self.discriminator = discriminator
        self.generator = generator
        self.latent_dim = latent_dim

    def compile(self, d_optimizer, g_optimizer):
        super(DCGAN, self).compile()
        self.loss_fn = losses.BinaryCrossentropy()
        self.d_optimizer = d_optimizer
        self.g_optimizer = g_optimizer
        self.d_loss_metric = metrics.Mean(name="d_loss")
        self.d_real_acc_metric = metrics.BinaryAccuracy(name="d_real_acc")
        self.d_fake_acc_metric = metrics.BinaryAccuracy(name="d_fake_acc")
        self.d_acc_metric = metrics.BinaryAccuracy(name="d_acc")
        self.g_loss_metric = metrics.Mean(name="g_loss")
        self.g_acc_metric = metrics.BinaryAccuracy(name="g_acc")

    @property
    def metrics(self):
        return [
            self.d_loss_metric,
            self.d_real_acc_metric,
            self.d_fake_acc_metric,
            self.d_acc_metric,
            self.g_loss_metric,
            self.g_acc_metric,
        ]

    def train_step(self, real_images):
        # Sample random points in the latent space
        batch_size = tf.shape(real_images)[0]
        random_latent_vectors = tf.random.normal(
            shape=(batch_size, self.latent_dim)
        )

        # Train the discriminator on fake images
        with tf.GradientTape() as gen_tape, tf.GradientTape() as disc_tape:
            generated_images = self.generator(
                random_latent_vectors, training=True
            )
            real_predictions = self.discriminator(real_images, training=True)
            fake_predictions = self.discriminator(
                generated_images, training=True
            )

            real_labels = tf.ones_like(real_predictions)
            real_noisy_labels = real_labels + NOISE_PARAM * tf.random.uniform(
                tf.shape(real_predictions)
            )
            fake_labels = tf.zeros_like(fake_predictions)
            fake_noisy_labels = fake_labels - NOISE_PARAM * tf.random.uniform(
                tf.shape(fake_predictions)
            )

            d_real_loss = self.loss_fn(real_noisy_labels, real_predictions)
            d_fake_loss = self.loss_fn(fake_noisy_labels, fake_predictions)
            d_loss = (d_real_loss + d_fake_loss) / 2.0

            g_loss = self.loss_fn(real_labels, fake_predictions)

        gradients_of_discriminator = disc_tape.gradient(
            d_loss, self.discriminator.trainable_variables
        )
        gradients_of_generator = gen_tape.gradient(
            g_loss, self.generator.trainable_variables
        )

        self.d_optimizer.apply_gradients(
            zip(gradients_of_discriminator, discriminator.trainable_variables)
        )
        self.g_optimizer.apply_gradients(
            zip(gradients_of_generator, generator.trainable_variables)
        )

        # Update metrics
```

```
        self.d_loss_metric.update_state(d_loss)
        self.d_real_acc_metric.update_state(real_labels, real_predictions)
        self.d_fake_acc_metric.update_state(fake_labels, fake_predictions)
        self.d_acc_metric.update_state(
            [real_labels, fake_labels], [real_predictions, fake_predictions]
        )
        self.g_loss_metric.update_state(g_loss)
        self.g_acc_metric.update_state(real_labels, fake_predictions)

        return {m.name: m.result() for m in self.metrics}
```

```
# create a DCGAN
dcgan = DCGAN(discriminator=discriminator, generator=generator , latent_dim=Z_DIM)
```

```
if LOAD_MODEL:
  dcgan.load_weights("./checkpoint/checkpoint.ckpt")
```

Train the GAN

```
dcgan.compile(
    d_optimizer=optimizers.Adam(
        learning_rate=LEARNING_RATE, beta_1=ADAM_BETA_1, beta_2=ADAM_BETA_2
    ),
    g_optimizer=optimizers.Adam(
        learning_rate=LEARNING_RATE, beta_1=ADAM_BETA_1, beta_2=ADAM_BETA_2
    ),
)
```

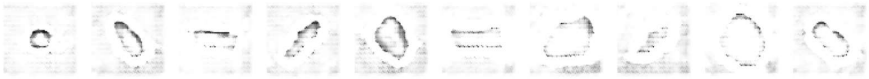
```
model_checkpoint_callback = callbacks.ModelCheckpoint(
    filepath="./checkpoint/checkpoint.ckpt",
    save_weights_only=True,
    save_freq="epoch",
    verbose=0,
)

tensorboard_callback = callbacks.TensorBoard(log_dir="./logs")

class ImageGenerator(callbacks.Callback):
    def __init__(self, num_img, latent_dim):
        self.num_img = num_img
        self.latent_dim = latent_dim

    def on_epoch_end(self, epoch, logs=None):
        random_latent_vectors = tf.random.normal(
            shape=(self.num_img, self.latent_dim)
        )
        generated_images = self.model.generator(random_latent_vectors)
        generated_images = generated_images * 127.5 + 127.5
        generated_images = generated_images.numpy()
        display(
            generated_images,
            save_to="/content/output/generated_img_%03d.png" % (epoch),
        )
```

```
dcgan.fit(
    train,
    epochs=EPOCHS,
    callbacks=[
        model_checkpoint_callback,
        tensorboard_callback,
        ImageGenerator(num_img=10, latent_dim=Z_DIM),
    ],
)
```

```
50/50 [==============================] - 12s 210ms/step - d_loss: -0.7727 - d_real_ac
Epoch 286/300
50/50 [==============================] - ETA: 0s - d_loss: -0.7655 - d_real_acc: 1.00
Saved to /content/output/generated_img_285.png
```

```
50/50 [==============================] - 12s 211ms/step - d_loss: -0.7655 - d_real_ac
Epoch 287/300
50/50 [==============================] - ETA: 0s - d_loss: -0.7652 - d_real_acc: 1.00
Saved to /content/output/generated_img_286.png
```

```
50/50 [==============================] - 12s 211ms/step - d_loss: -0.7652 - d_real_ac
Epoch 288/300
50/50 [==============================] - ETA: 0s - d_loss: -0.7698 - d_real_acc: 1.00
Saved to /content/output/generated_img_287.png
```



```
50/50 [==============================] - 12s 211ms/step - d_loss: -0.7698 - d_real_ac
Epoch 289/300
50/50 [==============================] - ETA: 0s - d_loss: -0.7651 - d_real_acc: 1.00
Saved to /content/output/generated_img_288.png
```

```
50/50 [==============================] - 12s 216ms/step - d_loss: -0.7651 - d_real_ac
Epoch 290/300
50/50 [==============================] - ETA: 0s - d_loss: -0.7713 - d_real_acc: 1.00
Saved to /content/output/generated_img_289.png
```

```
50/50 [==============================] - 12s 209ms/step - d_loss: -0.7713 - d_real_ac
Epoch 291/300
50/50 [==============================] - ETA: 0s - d_loss: -0.7660 - d_real_acc: 1.00
Saved to /content/output/generated_img_290.png
```

```
50/50 [==============================] - 12s 221ms/step - d_loss: -0.7660 - d_real_ac
Epoch 292/300
50/50 [==============================] - ETA: 0s - d_loss: -0.7726 - d_real_acc: 1.00
Saved to /content/output/generated_img_291.png
```

```
50/50 [==============================] - 12s 214ms/step - d_loss: -0.7726 - d_real_ac
Epoch 293/300
```