

▼ Ungraded Lab: First DCGAN

In this lab, you will see a demo of a Deep Convolutional GAN (DCGAN) trained on Fashion MNIST. You'll see architectural differences from the GAN in the first lab and also see the best practices when building this network.

▼ Imports

```
import tensorflow as tf
import tensorflow.keras as keras
```

```
import numpy as np
import matplotlib.pyplot as plt
from IPython import display
```

▼ Utilities

```
def plot_results(images, n_cols=None):
    '''visualizes fake images'''
    display.clear_output(wait=False)

    n_cols = n_cols or len(images)
    n_rows = (len(images) - 1) // n_cols + 1

    if images.shape[-1] == 1:
        images = np.squeeze(images, axis=-1)

    plt.figure(figsize=(n_cols, n_rows))

    for index, image in enumerate(images):
        plt.subplot(n_rows, n_cols, index + 1)
        plt.imshow(image, cmap="binary")
        plt.axis("off")
```

▼ Download and Prepare the Dataset

You will use the [Fashion MNIST](#) dataset for this exercise. As before, you will only need to create batches of the training images. The preprocessing steps are also shown below.

```
# download the training images
(X_train, _), _ = keras.datasets.fashion_mnist.load_data()

# normalize pixel values
X_train = X_train.astype(np.float32) / 255

# reshape and rescale
X_train = X_train.reshape(-1, 28, 28, 1) * 2. - 1.

BATCH_SIZE = 128

# create batches of tensors to be fed into the model
dataset = tf.data.Dataset.from_tensor_slices(X_train)
dataset = dataset.shuffle(1000)
dataset = dataset.batch(BATCH_SIZE, drop_remainder=True).prefetch(1)
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/train-labels-idx1-ubyte.gz
29515/29515 [=====] - 0s 0us/step
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/train-images-idx3-ubyte.gz
26421880/26421880 [=====] - 0s 0us/step
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/t10k-labels-idx1-ubyte.gz
5148/5148 [=====] - 0s 0us/step
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/t10k-images-idx3-ubyte.gz
4422102/4422102 [=====] - 0s 0us/step
```

▼ Build the Model

In DCGANs, convolutional layers are predominantly used to build the generator and discriminator. You will see how the layers are stacked as well as the best practices shown below.

▼ Generator

For the generator, we take in random noise and eventually transform it to the shape of the Fashion MNIST images. The general steps are:

- Feed the input noise to a dense layer.
- Reshape the output to have three dimensions. This stands for the (length, width, number of filters).
- Perform a deconvolution (with [Conv2DTranspose](#)), reducing the number of filters by half and using a stride of 2.
- The final layer upsamples the features to the size of the training images. In this case 28 x 28 x 1.

Notice that batch normalization is performed except for the final deconvolution layer. As best practice, `selu` is the activation used for the intermediate deconvolution while `tanh` is for the output. We printed the model summary so you can see the shapes at each layer.

```
codings_size = 32

generator = keras.models.Sequential([
    keras.layers.Dense(7 * 7 * 128, input_shape=[codings_size]),
    keras.layers.Reshape([7, 7, 128]),
    keras.layers.BatchNormalization(),
    keras.layers.Conv2DTranspose(64, kernel_size=5, strides=2, padding="SAME",
                                activation="selu"),
    keras.layers.BatchNormalization(),
    keras.layers.Conv2DTranspose(1, kernel_size=5, strides=2, padding="SAME",
                                activation="tanh"),
])

generator.summary()
```

🔗 Model: "sequential"

Layer (type)	Output Shape	Param #
=====		
dense (Dense)	(None, 6272)	206976
reshape (Reshape)	(None, 7, 7, 128)	0
batch_normalization (Batch Normalization)	(None, 7, 7, 128)	512
conv2d_transpose (Conv2DTranspose)	(None, 14, 14, 64)	204864
batch_normalization_1 (Batch Normalization)	(None, 14, 14, 64)	256
conv2d_transpose_1 (Conv2DTranspose)	(None, 28, 28, 1)	1601
=====		
Total params: 414,209		
Trainable params: 413,825		
Non-trainable params: 384		

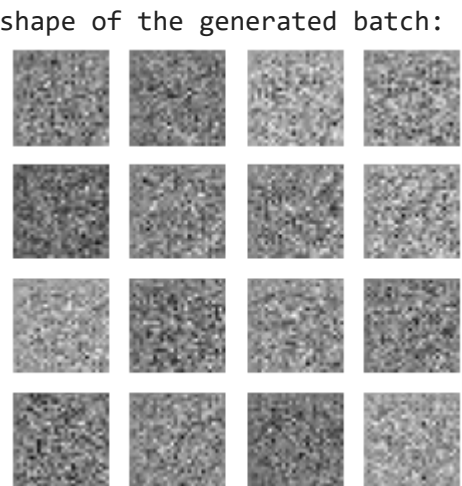
As a sanity check, let's see the fake images generated by the untrained generator and see the dimensions of the output.

```
# generate a batch of noise input (batch size = 16)
test_noise = tf.random.normal([16, codings_size])

# feed the batch to the untrained generator
test_image = generator(test_noise)

# visualize sample output
plot_results(test_image, n_cols=4)

print(f'shape of the generated batch: {test_image.shape}')
```



▼ Discriminator

The discriminator will use strided convolutions to reduce the dimensionality of the input images. As best practice, these are activated by [LeakyRELU](#). The output features will be flattened and fed to a 1-unit dense layer activated by `sigmoid`.

```
discriminator = keras.models.Sequential([
    keras.layers.Conv2D(64, kernel_size=5, strides=2, padding="SAME",
                        activation=keras.layers.LeakyReLU(0.2),
                        input_shape=[28, 28, 1]),
    keras.layers.Dropout(0.4),
    keras.layers.Conv2D(128, kernel_size=5, strides=2, padding="SAME",
                        activation=keras.layers.LeakyReLU(0.2)),
    keras.layers.Dropout(0.4),
    keras.layers.Flatten(),
    keras.layers.Dense(1, activation="sigmoid")
])
```

discriminator.summary()

Model: "sequential_1"		
Layer (type)	Output Shape	Param #
=====		
conv2d (Conv2D)	(None, 14, 14, 64)	1664
dropout (Dropout)	(None, 14, 14, 64)	0
conv2d_1 (Conv2D)	(None, 7, 7, 128)	204928
dropout_1 (Dropout)	(None, 7, 7, 128)	0
flatten (Flatten)	(None, 6272)	0
dense_1 (Dense)	(None, 1)	6273
=====		
Total params: 212,865		
Trainable params: 212,865		
Non-trainable params: 0		

As before, you will append these two subnetwork to build the complete GAN.

```
gan = keras.models.Sequential([generator, discriminator])
```

▼ Configure the Model for training

The discriminator and GAN will still be classifying fake and real images so you will use the same settings as before.

```
discriminator.compile(loss="binary_crossentropy", optimizer="rmsprop")
discriminator.trainable = False
gan.compile(loss="binary_crossentropy", optimizer="rmsprop")
```

▼ Train the Model

The training loop will also be identical to the previous one you built. Run the cells below and observe how the fake images become more convincing as the training progresses.

```
def train_gan(gan, dataset, random_normal_dimensions, n_epochs=50):
    """ Defines the two-phase training loop of the GAN
    Args:
        gan -- the GAN model which has the generator and discriminator
        dataset -- the training set of real images
        random_normal_dimensions -- dimensionality of the input to the generator
        n_epochs -- number of epochs
    """
    generator, discriminator = gan.layers
    for epoch in range(n_epochs):
        print("Epoch {}/{}".format(epoch + 1, n_epochs))
        for real_images in dataset:
            # infer batch size from the training batch
            batch_size = real_images.shape[0]

            # Train the discriminator - PHASE 1
            # create the noise
            noise = tf.random.normal(shape=[batch_size, random_normal_dimensions])

            # use the noise to generate fake images
```

```

fake_images = generator(noise)

# create a list by concatenating the fake images with the real ones
mixed_images = tf.concat([fake_images, real_images], axis=0)

# Create the labels for the discriminator
# 0 for the fake images
# 1 for the real images
discriminator_labels = tf.constant([[0.]] * batch_size + [[1.]] * batch_size)

# ensure that the discriminator is trainable
discriminator.trainable = True

# use train_on_batch to train the discriminator with the mixed images and the discriminator labels
discriminator.train_on_batch(mixed_images, discriminator_labels)

# Train the generator - PHASE 2
# create a batch of noise input to feed to the GAN
noise = tf.random.normal(shape=[batch_size, random_normal_dimensions])

# label all generated images to be "real"
generator_labels = tf.constant([[1.]] * batch_size)

# freeze the discriminator
discriminator.trainable = False

# train the GAN on the noise with the labels all set to be true
gan.train_on_batch(noise, generator_labels)

# plot the fake images used to train the discriminator
plot_results(fake_images, 16)
plt.show()

```

```

train_gan(gan, dataset, codings_size, 100)

```

