GROUP-3

MASKANI NAVEEN YADAV - 20201CEI0025

KATIPALLY YASHWANTH REDDY - 20201CEI0039

PERAM MAHENDRA REDDY - 20201CEI0112

INDLA MADHANKUMAR - 20201CEI0136

Import libraries

```
%load_ext autoreload
%autoreload 2

import numpy as np
import matplotlib.pyplot as plt

from tensorflow.keras import layers, models, datasets, callbacks
import tensorflow.keras.backend as K


import matplotlib.pyplot as plt


def sample_batch(dataset):
    batch = dataset.take(1).get_single_element()
    if isinstance(batch, tuple):
        batch = batch[0]
    return batch.numpy()


def display(
    images, n=10, size=(20, 3), cmap="gray_r", as_type="float32", save_to=None
):
    """
    Displays n random images from each one of the supplied arrays.
    """
    if images.max() > 1.0:
        images = images / 255.0
    elif images.min() < 0.0:
        images = (images + 1.0) / 2.0

    plt.figure(figsize=size)
    for i in range(n):
        _ = plt.subplot(1, n, i + 1)
        plt.imshow(images[i].astype(as_type), cmap=cmap)
        plt.axis("off")

    if save_to:
        plt.savefig(save_to)
        print(f"\nSaved to {save_to}")

    plt.show()
```

0. Parameters

```
IMAGE_SIZE = 32
CHANNELS = 1
BATCH_SIZE = 100
BUFFER_SIZE = 1000
VALIDATION_SPLIT = 0.2
EMBEDDING_DIM = 2
EPOCHS = 3
```

1. Prepare the data

```
# Load the data
(x_train, y_train), (x_test, y_test) = datasets.fashion_mnist.load_data()
```

```
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
```

```
# Preprocess the data


def preprocess(imgs):
    """
    Normalize and reshape the images
    """
    imgs = imgs.astype("float32") / 255.0
    imgs = np.pad(imgs, ((0, 0), (2, 2), (2, 2)), constant_values=0.0) # amount of padding to be applied to each dimension of the array. In
    imgs = np.expand_dims(imgs, -1)
    return imgs

'''
(0, 0): No padding is added along the first dimension (batch size). The 0 on both sides means no additional rows are added at the beginning
(2, 2): Padding of 2 pixels is added along the second dimension (height). This means 2 rows of zeros are added both at the top and bottom of
(2, 2): Padding of 2 pixels is added along the third dimension (width). This means 2 columns of zeros are added both on the left and right c
'''

x_train = preprocess(x_train)
x_test = preprocess(x_test)
```

```
# Show some items of clothing from the training set
display(x_train)
```

2. Build the autoencoder

```
# Encoder
# In an autoencoder, the encoder's job is to take the input image and map it to an
# embedding vector in the latent space.

encoder_input = layers.Input(
    shape=(IMAGE_SIZE, IMAGE_SIZE, CHANNELS), name="encoder_input"
)
x = layers.Conv2D(32, (3, 3), strides=2, activation="relu", padding="same")(
    encoder_input
)
x = layers.Conv2D(64, (3, 3), strides=2, activation="relu", padding="same")(x)
x = layers.Conv2D(128, (3, 3), strides=2, activation="relu", padding="same")(x)
shape_before_flattening = K.int_shape(x)[1:]  # the decoder will need this!

x = layers.Flatten()(x)
encoder_output = layers.Dense(EMBEDDING_DIM, name="encoder_output")(x)

encoder = models.Model(encoder_input, encoder_output)
encoder.summary()
```

```
Model: "model"
```

```
 Layer (type)                Output Shape              Param #   
=================================================================
 encoder_input (InputLayer)  [(None, 32, 32, 1)]       0         
                                                                 
 conv2d (Conv2D)             (None, 16, 16, 32)        320       
                                                                 
 conv2d_1 (Conv2D)           (None, 8, 8, 64)          18496     
                                                                 
 conv2d_2 (Conv2D)           (None, 4, 4, 128)         73856     
                                                                 
 flatten (Flatten)           (None, 2048)              0         
                                                                 
 encoder_output (Dense)      (None, 2)                 4098      
                                                                 
=================================================================
Total params: 96770 (378.01 KB)
Trainable params: 96770 (378.01 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

```
# Decoder
# The decoder is a mirror image of the encoder—instead of convolutional layers,
# we use convolutional transpose layers

decoder_input = layers.Input(shape=(EMBEDDING_DIM,), name="decoder_input")

x = layers.Dense(np.prod(shape_before_flattening))(decoder_input)
x = layers.Reshape(shape_before_flattening)(x)

x = layers.Conv2DTranspose(128, (3, 3), strides=2, activation="relu", padding="same")(x)
x = layers.Conv2DTranspose(64, (3, 3), strides=2, activation="relu", padding="same")(x)
x = layers.Conv2DTranspose(32, (3, 3), strides=2, activation="relu", padding="same")(x)

decoder_output = layers.Conv2D(
    CHANNELS,
    (3, 3),
    strides=1,
    activation="sigmoid",
    padding="same",
    name="decoder_output",
)(x)

decoder = models.Model(decoder_input, decoder_output)
decoder.summary()
```

```
Model: "model_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 decoder_input (InputLayer)  [(None, 2)]               0         
                                                                 
 dense (Dense)               (None, 2048)              6144      
                                                                 
 reshape (Reshape)           (None, 4, 4, 128)         0         
                                                                 
 conv2d_transpose (Conv2DTr  (None, 8, 8, 128)         147584    
 anspose)                                                        
                                                                 
 conv2d_transpose_1 (Conv2D  (None, 16, 16, 64)        73792     
 Transpose)                                                      
                                                                 
 conv2d_transpose_2 (Conv2D  (None, 32, 32, 32)        18464     
 Transpose)                                                      
                                                                 
 decoder_output (Conv2D)     (None, 32, 32, 1)         289       
                                                                 
=================================================================
Total params: 246273 (962.00 KB)
Trainable params: 246273 (962.00 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

```
# Joining the Encoder to the Decoder

# To train the encoder and decoder simultaneously, we need to define a model that will
# represent the flow of an image through the encoder and back out through the decoder.

autoencoder = models.Model(
    encoder_input, decoder(encoder_output)
)  # decoder(encoder_output)
autoencoder.summary()
```

```
Model: "model_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 encoder_input (InputLayer)  [(None, 32, 32, 1)]       0         
                                                                 
 conv2d (Conv2D)             (None, 16, 16, 32)        320       
                                                                 
 conv2d_1 (Conv2D)           (None, 8, 8, 64)          18496     
                                                                 
 conv2d_2 (Conv2D)           (None, 4, 4, 128)         73856     
                                                                 
 flatten (Flatten)           (None, 2048)              0         
                                                                 
 encoder_output (Dense)      (None, 2)                 4098      
                                                                 
 model_1 (Functional)        (None, 32, 32, 1)         246273    
                                                                 
=================================================================
Total params: 343043 (1.31 MB)
Trainable params: 343043 (1.31 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

3. Train the autoencoder

```
# Compile the autoencoder
autoencoder.compile(optimizer="adam", loss="binary_crossentropy")
```

```
# Create a model save checkpoint
model_checkpoint_callback = callbacks.ModelCheckpoint(
    filepath="./checkpoint",
    save_weights_only=False,
    save_freq="epoch",
    monitor="loss",
    mode="min",
    save_best_only=True,
    verbose=0,
)
tensorboard_callback = callbacks.TensorBoard(log_dir="./logs")
```

```
autoencoder.fit(
    x_train,
    x_train,
    epochs=EPOCHS,
    batch_size=BATCH_SIZE,
    shuffle=True,
    validation_data=(x_test, x_test),
    callbacks=[model_checkpoint_callback, tensorboard_callback],
)
```

```
Epoch 1/3
600/600 [==============================] - 236s 391ms/step - loss: 0.2939 - val_loss: 0.2618
Epoch 2/3
600/600 [==============================] - 230s 384ms/step - loss: 0.2569 - val_loss: 0.2559
Epoch 3/3
600/600 [==============================] - 220s 367ms/step - loss: 0.2533 - val_loss: 0.2535
<keras.src.callbacks.History at 0x7e980e8a1900>
```

```
# Save the final models
autoencoder.save("./models/autoencoder")
encoder.save("./models/encoder")
decoder.save("./models/decoder")
```

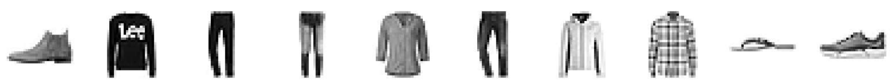
```
WARNING:tensorflow:Compiled the loaded model, but the compiled metrics have yet to be built. `model.compile_metrics` will be empty until
WARNING:tensorflow:Compiled the loaded model, but the compiled metrics have yet to be built. `model.compile_metrics` will be empty until
```

4. Reconstruct using the autoencoder

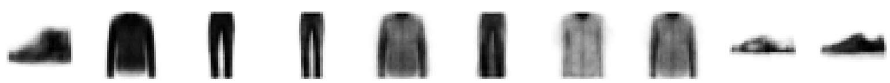
```
n_to_predict = 5000
example_images = x_test[:n_to_predict]
example_labels = y_test[:n_to_predict]
```

```
predictions = autoencoder.predict(example_images)

print("Example real clothing items")
display(example_images)
print("Reconstructions")
display(predictions)
```

```
157/157 [==============================] - 6s 35ms/step
Example real clothing items
```



```
Reconstructions
```



Notice how the reconstruction isn't perfect—there are still some details of the original images that aren't captured by the decoding process, such as logos. This is because by reducing each image to just two numbers, we naturally lose some information.

5. Embed using the encoder

Let's now investigate how the encoder is representing images in the latent space.

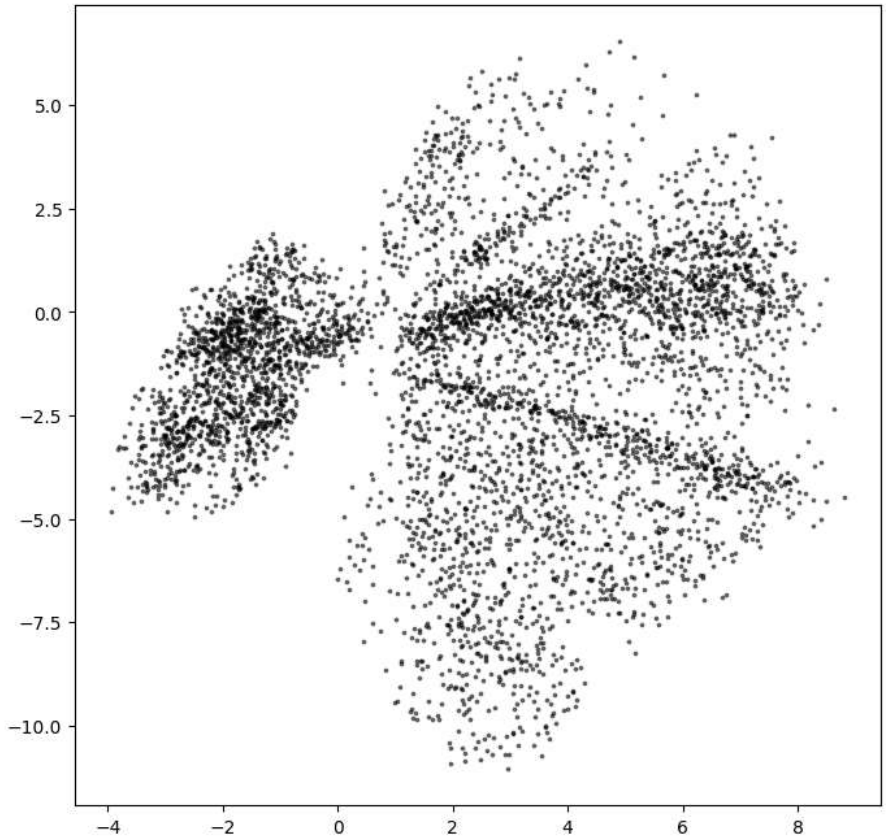
```
# Encode the example images
embeddings = encoder.predict(example_images)
```

```
157/157 [==============================] - 1s 6ms/step
```

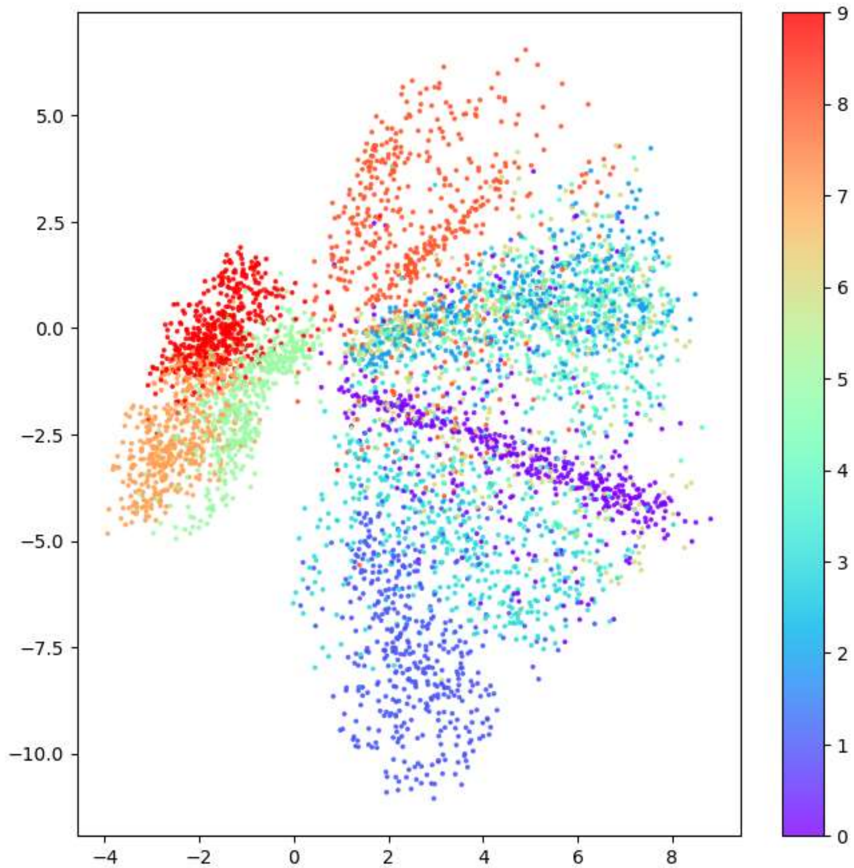
```
# Some examples of the embeddings
print(embeddings[:10])
```

```
[[-1.4910505  -1.2249507 ]
 [ 7.0651135   1.1740873 ]
 [ 3.7713332  -9.269059  ]
 [ 1.9740249  -6.6853676 ]
 [ 3.0130956   0.5014184 ]
 [ 3.4576645  -5.4743385 ]
 [ 1.8377662  -0.91126204]
 [ 2.6585374   0.02292454]
 [-1.7201378  -4.2824516 ]
 [-2.7758434  -3.5814784 ]]
```

```
# Show the encoded points in 2D space
figsize = 8

plt.figure(figsize=(figsize, figsize))
plt.scatter(embeddings[:, 0], embeddings[:, 1], c="black", alpha=0.5, s=3)
plt.show()
```




```
# Colour the embeddings by their label
example_labels = y_test[:n_to_predict]

figsize = 8
plt.figure(figsize=(figsize, figsize))
plt.scatter(
    embeddings[:, 0],
    embeddings[:, 1],
    cmap="rainbow",
    c=example_labels,
    alpha=0.8,
    s=3,
)
plt.colorbar()
plt.show()

'''
ID Clothing label
0 T-shirt/top
1 Trouser
2 Pullover
3 Dress
4 Coat
5 Sandal
6 Shirt
7 Sneaker
8 Bag
9 Ankle boot

'''
```



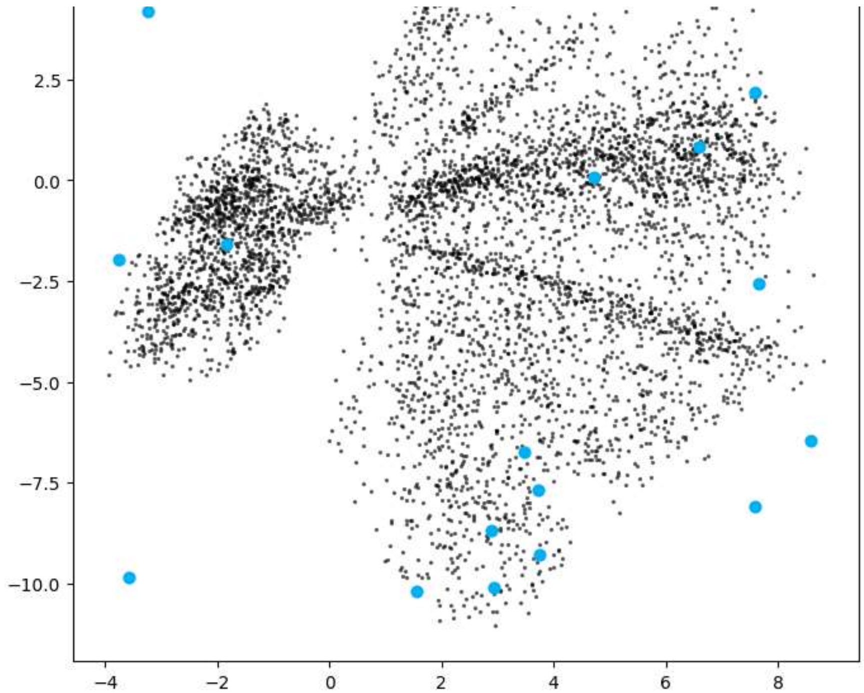
6. Generate using the decoder

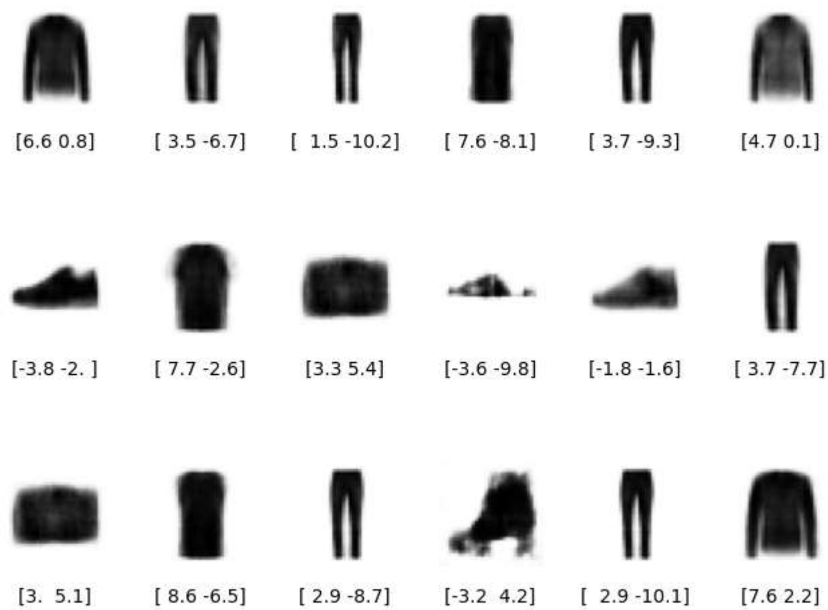
Generating novel images using the decoder

```
# Get the range of the existing embeddings
mins, maxs = np.min(embeddings, axis=0), np.max(embeddings, axis=0)

# Sample some points in the latent space
grid_width, grid_height = (6, 3)
sample = np.random.uniform(
    mins, maxs, size=(grid_width * grid_height, EMBEDDING_DIM)
)


# Decode the sampled points
reconstructions = decoder.predict(sample)
```

```
1/1 [==============================] - 0s 146ms/step
```

```
# Draw a plot of...
figsize = 8
plt.figure(figsize=(figsize, figsize))

# ... the original embeddings ...
plt.scatter(embeddings[:, 0], embeddings[:, 1], c="black", alpha=0.5, s=2)

# ... and the newly generated points in the latent space
plt.scatter(sample[:, 0], sample[:, 1], c="#00B0F0", alpha=1, s=40)
plt.show()

# Add underneath a grid of the decoded images
fig = plt.figure(figsize=(figsize, grid_height * 2))
fig.subplots_adjust(hspace=0.4, wspace=0.4)

for i in range(grid_width * grid_height):
    ax = fig.add_subplot(grid_height, grid_width, i + 1)
    ax.axis("off")
    ax.text(
        0.5,
        -0.35,
        str(np.round(sample[i, :], 1)),
        fontsize=10,
        ha="center",
        transform=ax.transAxes,
    )
    ax.imshow(reconstructions[i, :, :], cmap="Greys")
```


2.5
0.0
−2.5
−5.0
−7.5
−10.0
−4
−2
0
2
4
6
8


[6.6 0.8]
[3.5 -6.7]
[1.5 -10.2]
[7.6 -8.1]
[3.7 -9.3]
[4.7 0.1]
[-3.8 -2.]
[7.7 -2.6]
[3.3 5.4]
[-3.6 -9.8]
[-1.8 -1.6]
[3.7 -7.7]
[3. 5.1]
[8.6 -6.5]
[2.9 -8.7]
[-3.2 4.2]
[2.9 -10.1]
[7.6 2.2]