# 👖 Autoencoders on Fashion MNIST

## Source notes

Sources adapated from:

Notebook adapated for Google Colab from the codebase (https://github.com/davidADSP/Generative_Deep_Learning_2nd_Edition) for the book:

**Generative Deep Learning**, 2nd Edition by *David Foster* Released April 2023 Publisher(s): O'Reilly Media, Inc. ISBN: 9781098134181

cf: https://www.oreilly.com/library/view/generative-deep-learning/9781098134174/

Copyright notice:

https://github.com/davidADSP/Generative_Deep_Learning_2nd_Edition#Apache-2.0-1-ov-file

## Preparation

```
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()
```

In this notebook, we'll walk through the steps required to train your own autoencoder on the fashion MNIST dataset.

```
%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
```

## 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-da
29515/29515 ━━━━━━━━━━━━━━━━━━━━ 0s 0us/step
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-da
26421880/26421880 ━━━━━━━━━━━━━━━━━━━━ 0s 0us/step
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-da
5148/5148 ━━━━━━━━━━━━━━━━━━━━ 0s 0us/step
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-da
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)
    imgs = np.expand_dims(imgs, -1)
    return imgs


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
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: "functional"**

| Layer (type) | Output Shape | |
|---|---|---|
| encoder_input (InputLayer) | (None, 32, 32, 1) | |
| conv2d (Conv2D) | (None, 16, 16, 32) | |
| conv2d_1 (Conv2D) | (None, 8, 8, 64) | |
| conv2d_2 (Conv2D) | (None, 4, 4, 128) | |
| flatten (Flatten) | (None, 2048) | |
| encoder_output (Dense) | (None, 2) | |

**Total params:** 96,770 (378.01 KB)
**Trainable params:** 96,770 (378.01 KB)
**Non-trainable params:** 0 (0.00 B)

```
# Decoder
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: "functional_1"**

| Layer (type) | Output Shape | |
|---|---|---|
| decoder_input (InputLayer) | (None, 2) | |
| dense (Dense) | (None, 2048) | |
| reshape (Reshape) | (None, 4, 4, 128) | |
| conv2d_transpose (Conv2DTranspose) | (None, 8, 8, 128) | |
| conv2d_transpose_1 (Conv2DTranspose) | (None, 16, 16, 64) | |
| conv2d_transpose_2 (Conv2DTranspose) | (None, 32, 32, 32) | |
| decoder_output (Conv2D) | (None, 32, 32, 1) | |

**Total params:** 246,273 (962.00 KB)
**Trainable params:** 246,273 (962.00 KB)
**Non-trainable params:** 0 (0.00 B)

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

**Model: "functional_2"**

| Layer (type) | Output Shape | |
|---|---|---|
| encoder_input (InputLayer) | (None, 32, 32, 1) | |
| conv2d (Conv2D) | (None, 16, 16, 32) | |
| conv2d_1 (Conv2D) | (None, 8, 8, 64) | |
| conv2d_2 (Conv2D) | (None, 4, 4, 128) | |
| flatten (Flatten) | (None, 2048) | |
| encoder_output (Dense) | (None, 2) | |
| functional_1 (Functional) | (None, 32, 32, 1) | |

**Total params:** 343,043 (1.31 MB)
**Trainable params:** 343,043 (1.31 MB)
**Non-trainable params:** 0 (0.00 B)

## 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.keras",
    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 ━━━━━━━━━━━━━━━━━━━━ 13s 10ms/step - loss: 0.3657 - val_loss: 0.263
Epoch 2/3
600/600 ━━━━━━━━━━━━━━━━━━━━ 5s 8ms/step - loss: 0.2599 - val_loss: 0.2576
Epoch 3/3
600/600 ━━━━━━━━━━━━━━━━━━━━ 10s 9ms/step - loss: 0.2561 - val_loss: 0.2556
<keras.src.callbacks.history.History at 0x7b9b86aacdf0>
```

```
# Save the final models
from pathlib import Path
Path("./models").mkdir(parents=True, exist_ok=True)
autoencoder.save("./models/autoencoder.keras")
encoder.save("./models/encoder.keras")
decoder.save("./models/decoder.keras")
```

## 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 ━━━━━━━━━━━━━━━━━━━━ 2s 6ms/step
Example real clothing items
```

```
Reconstructions
```

## 5. Embed using the encoder

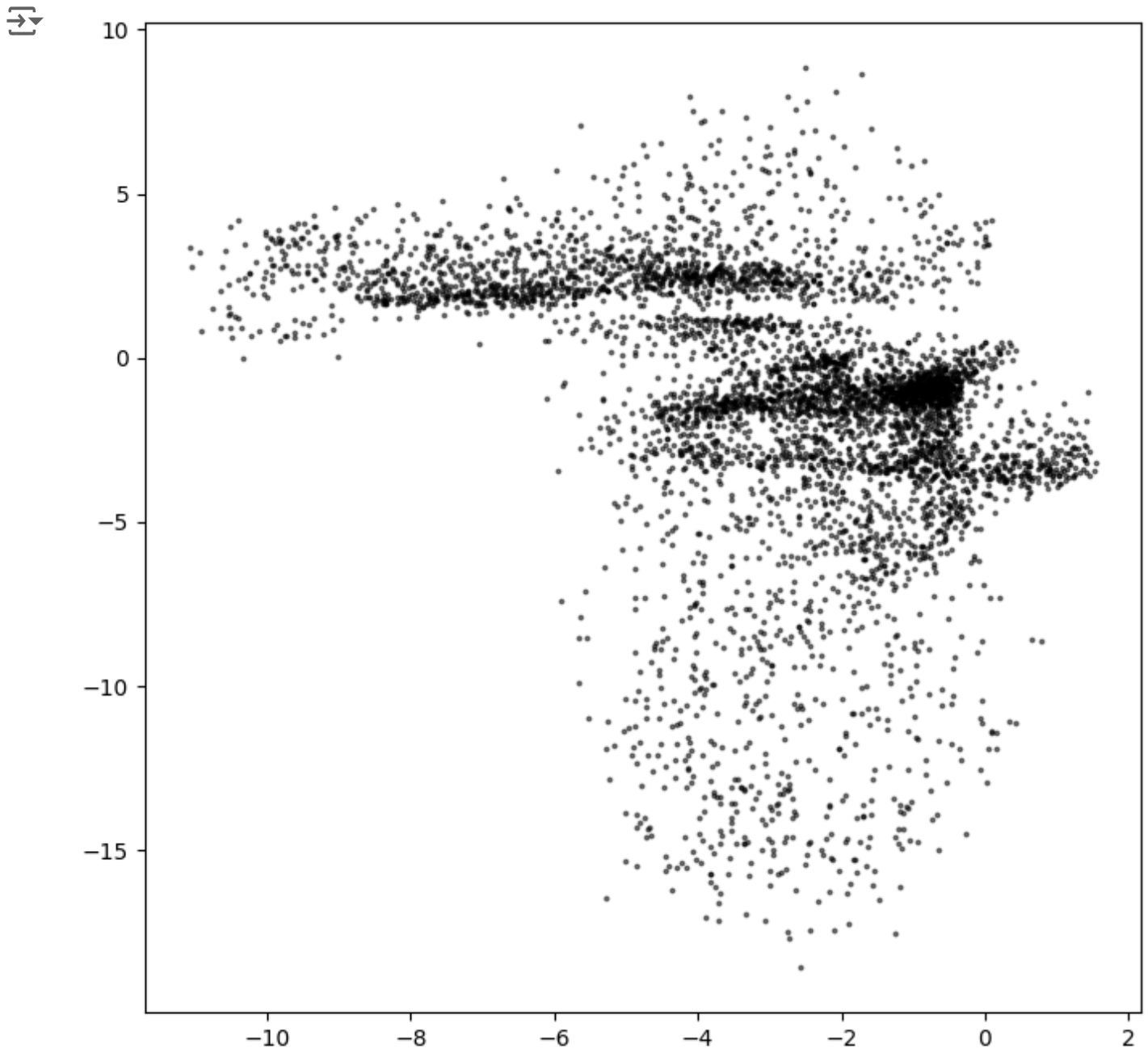
```
# Encode the example images
embeddings = encoder.predict(example_images)
```

```
157/157 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step
```

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

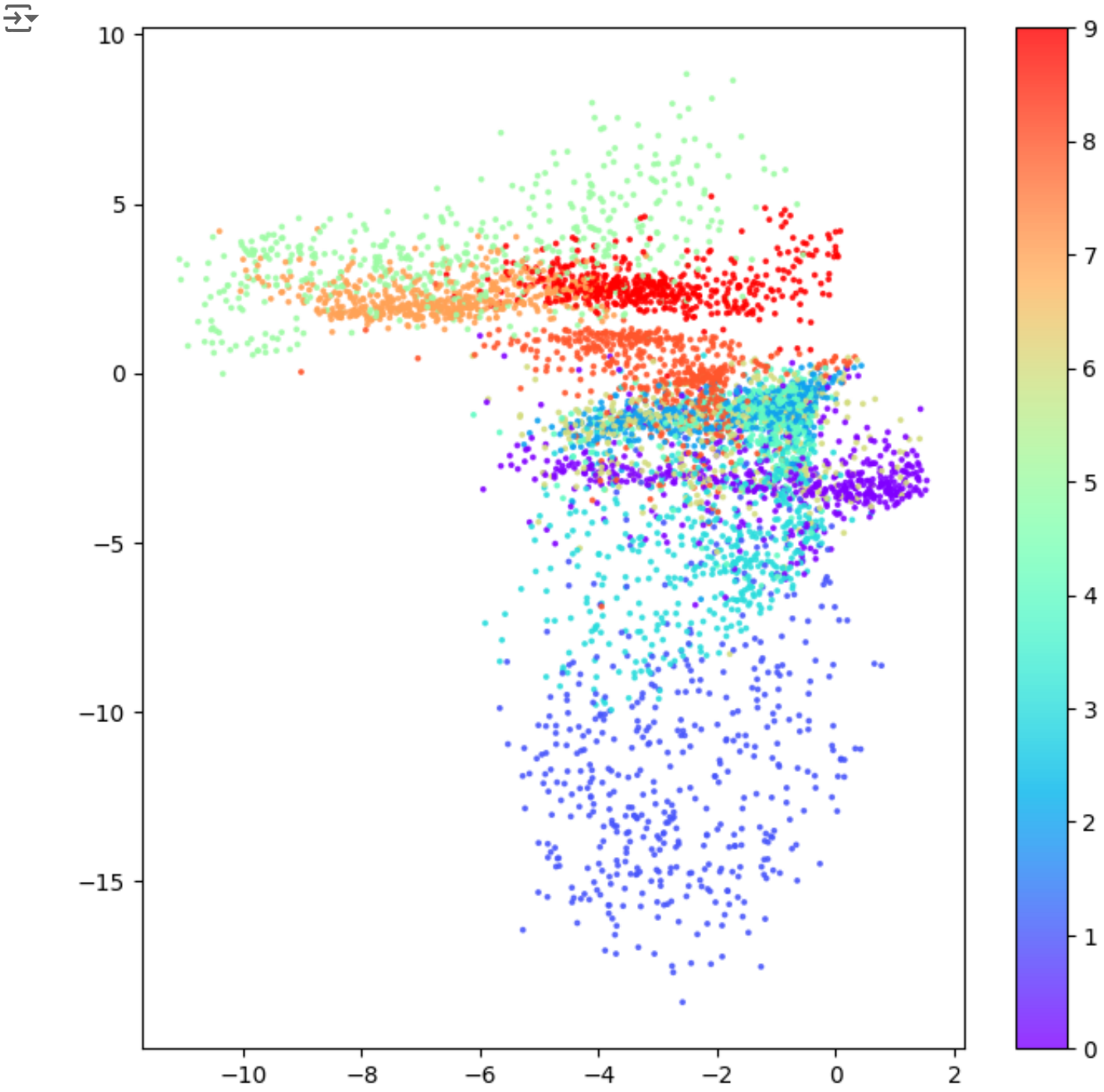
```
[[ -5.4338603    3.2548223 ]
 [ -0.6418047   -0.84171236]
 [ -0.26298317 -14.493086  ]
 [ -3.729753   -13.378001  ]
 [ -2.6105247   -0.9483488 ]
 [ -0.8168253   -6.581604  ]
 [ -4.0922318   -2.3349204 ]
 [ -3.2285955   -1.5143324 ]
 [-10.532016     1.2840981 ]
 [ -7.994115     1.957331  ]]
```

```
# 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 (clothing type - see table)
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()
```

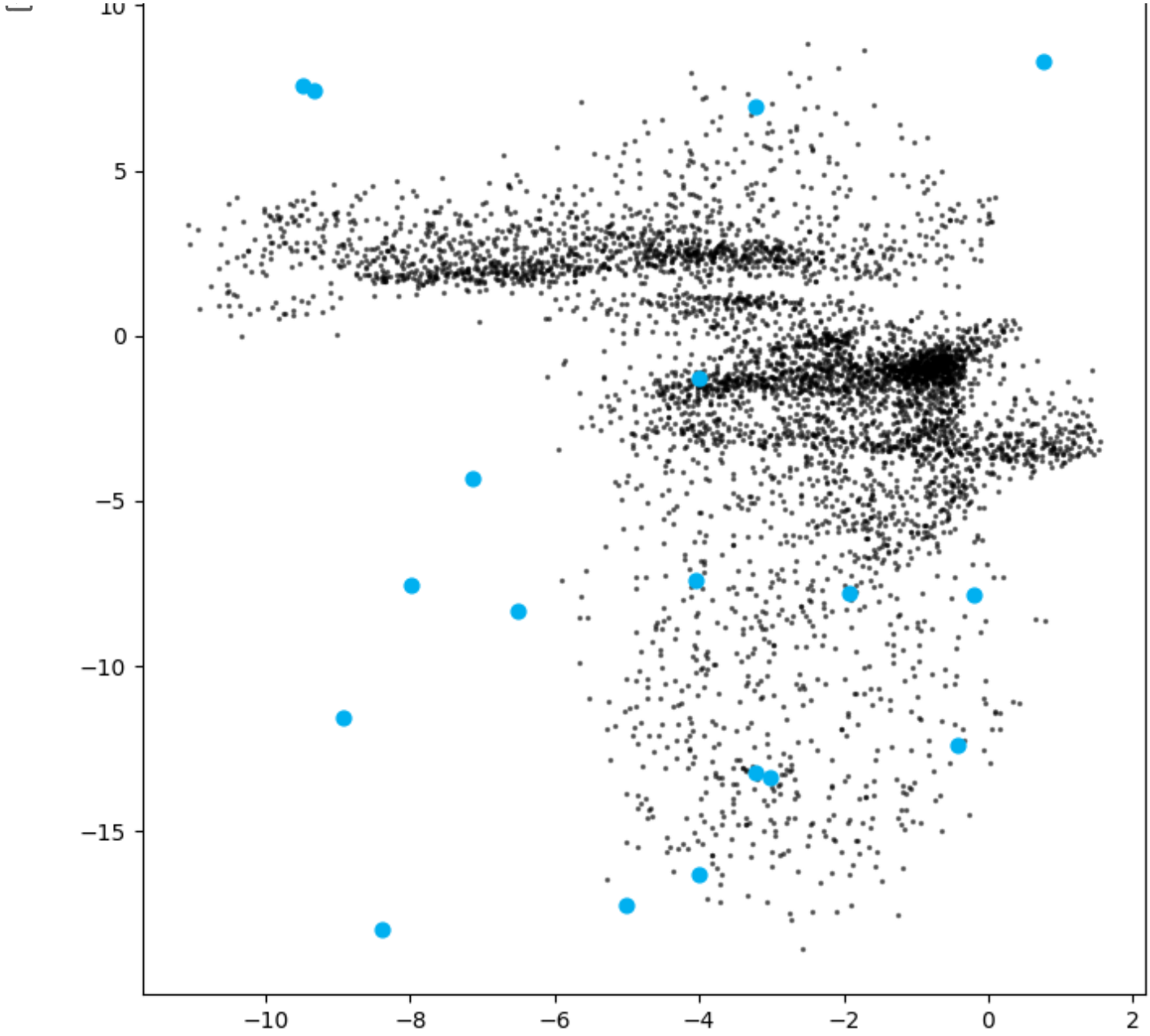


## 6. Generate using the decoder

```
# 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 473ms/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")
```



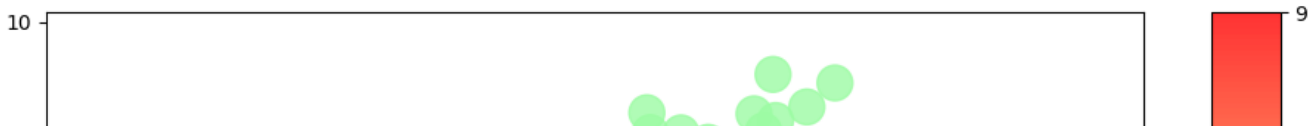

[-9.5 7.6] [ -8.9 -11.6] [ -3. -13.4] [-0.2 -7.8] [-6.5 -8.3] [ -0.4 -12.4]

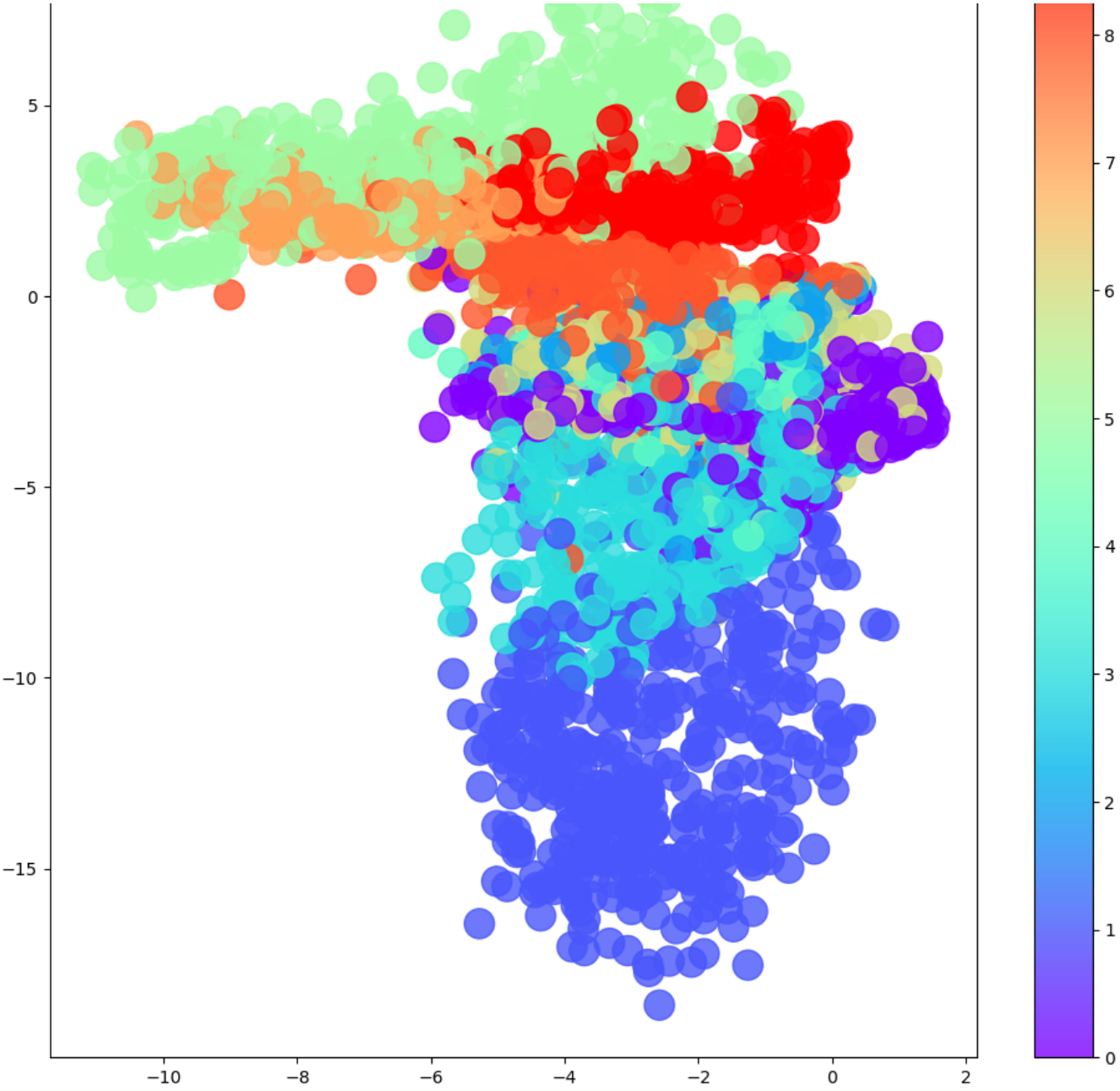
[-4. -1.3] [-7.1 -4.3] [-9.3 7.4] [-8. -7.6] [-1.9 -7.8] [-4. -7.4]

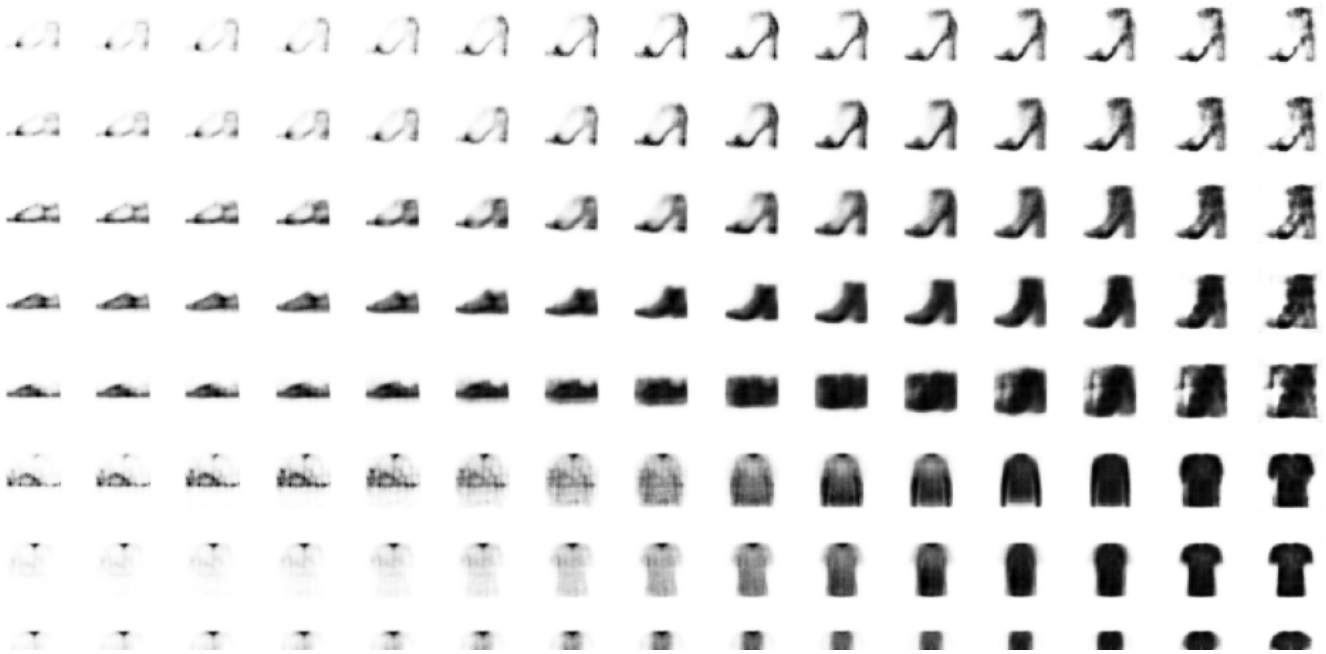
[ -3.2 -13.2] [0.8 8.3] [ -4. -16.3] [ -5. -17.3] [ -8.4 -18. ] [-3.2 6.9]

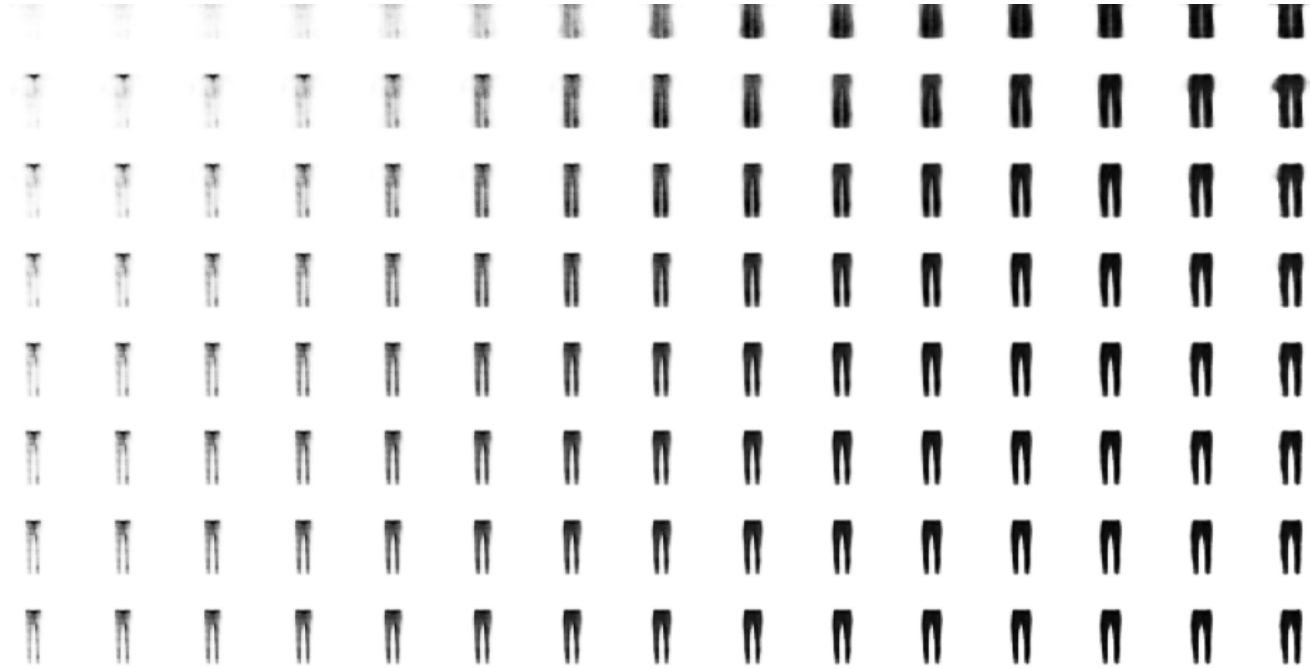
```
# Colour the embeddings by their label (clothing type - see table)
figsize = 12
grid_size = 15
plt.figure(figsize=(figsize, figsize))
plt.scatter(
    embeddings[:, 0],
    embeddings[:, 1],
    cmap="rainbow",
    c=example_labels,
    alpha=0.8,
    s=300,
)
plt.colorbar()

x = np.linspace(min(embeddings[:, 0]), max(embeddings[:, 0]), grid_size)
y = np.linspace(max(embeddings[:, 1]), min(embeddings[:, 1]), grid_size)
xv, yv = np.meshgrid(x, y)
xv = xv.flatten()
yv = yv.flatten()
grid = np.array(list(zip(xv, yv)))

reconstructions = decoder.predict(grid)
# plt.scatter(grid[:, 0], grid[:, 1], c="black", alpha=1, s=10)
plt.show()

fig = plt.figure(figsize=(figsize, figsize))
fig.subplots_adjust(hspace=0.4, wspace=0.4)
for i in range(grid_size**2):
    ax = fig.add_subplot(grid_size, grid_size, i + 1)
    ax.axis("off")
    ax.imshow(reconstructions[i, :, :], cmap="Greys")
```

```
8/8 ━━━━━━━━━━━━━━━━━━━━ 1s 47ms/step
```




5
0
−5
−10
−15
−10
−8
−6
−4
−2
0
2
8
7
6
5
4
3
2
1
0





| 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 |