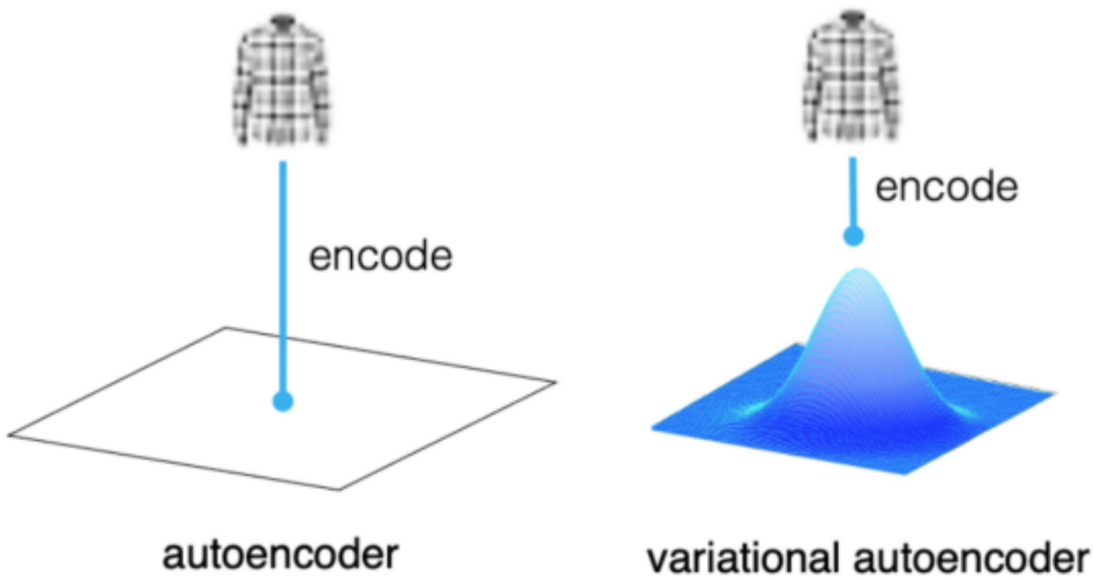
VARIATIONAL AUTOENCODERS

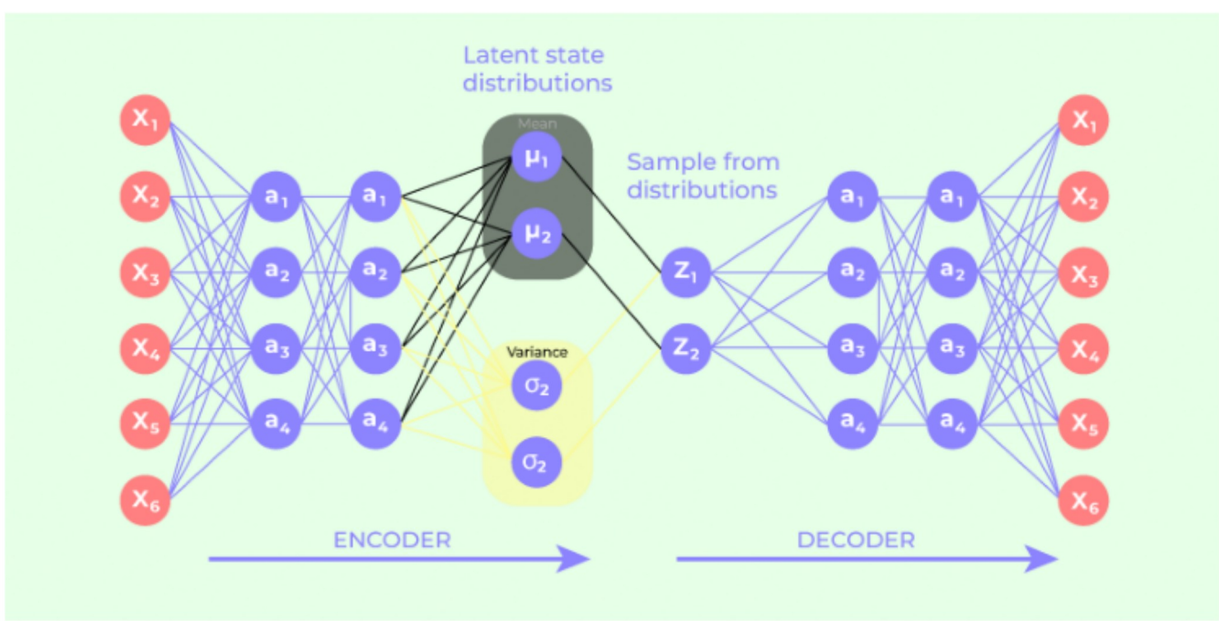
In an autoencoder, each image is mapped directly to one point in the latent space. In a variational autoencoder, each image is instead mapped to a multivariate normal dis tribution around a point in the latent space. This latent space captures the underlying patterns in the data. VAEs are different from regular autoencoders because they also learn the probability distribution of the data in the latent space. This allows them to generate new data samples that are similar to the data they were trained on.

In [5]:
```
img('VAE_abstract.png', 120)
```



In a base Autoenoder there was no requirement for the latent space to be continu ous—even if at a point a well-formed image is decoded, there's no requirement for the neighboring points to look similar. Now, since we are sampling a random point from an area around z_mean, the decoder must ensure that all points in the same neighborhood produce very similar images when decoded, so that the recon struction loss remains small. This is a very nice property that ensures that even when we choose a point in the latent space that has never been seen by the decoder, it is likely to decode to an image that is well formed

In [6]:
```
img('Variational-AutoEncoder.png', 150)
```



In [4]:
```
import matplotlib.pyplot as plt
def img(path,dpi):
    image = plt.imread(path)
    plt.figure(dpi=dpi)
    plt.axis('off')
    plt.imshow(image)
```

In [18]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import tensorflow as tf
import keras
from keras import models, layers, datasets, metrics, optimizers, losses
from keras.layers import Dense, Input, Conv2D, Conv2DTranspose, Reshape, Flatten
```

Dataset

Fashion-MNIST is a popular dataset of grayscale images used for training and evaluating machine learning models, particularly in the realm of image classification. Here it is used for generation task due to the simplicity of data and the sheer quantity and quality of the data. here is an example of the dataset

In [9]:
```
img('fashion_mnist.webp', 100)
```

In [19]:
```
(X_train, y_train), (X_test, y_test) = datasets.fashion_mnist.load_data()
```

Sampling layer

This layer samples a point z from the normal distribution characterized by z_mean and z_log_var, it doesn't have any trainable parameters

In [20]:
```
class Sampling(layers.Layer):
    def call(self, inputs):
        z_mean, z_log_var = inputs
        batch = tf.shape(z_mean)[0]
        dim = tf.shape(z_mean)[1]
        epsilon=tf.keras.backend.random_normal(shape=(batch,dim))
        return z_mean + tf.exp(0.5*z_log_var)*epsilon
```

Encoder of VAE

In [23]:
```
enc_in0 = Input(shape = (32, 32, 1), name="enc_input")
X = Conv2D(filters=32, kernel_size=(3,3), strides=2, activation='relu', padding='sa
X = Conv2D(filters=64, kernel_size=(3,3), strides=2, activation='relu', padding='sa
X = Conv2D(filters=128, kernel_size=(3,3), strides=2, activation='relu', padding='s
```

```
X = Flatten(name="enc_Flatten")(X)

z_mean = Dense(units=2, name="z_mean")(X)
z_log_var = Dense(units=2, name="z_log_var")(X)

z = Sampling()([z_mean, z_log_var])

encoder = models.Model(enc_in0, [z_mean, z_log_var, z], name="Encoder")
encoder.summary()
```

Model: "Encoder"

Layer (type)	Output Shape	Param #	Connected to
enc_input (InputLayer)	(None, 32, 32, 1)	0	-
enc_conv1 (Conv2D)	(None, 16, 16, 32)	320	enc_input[0][0]
enc_conv2 (Conv2D)	(None, 8, 8, 64)	18,496	enc_conv1[0][0]
enc_conv3 (Conv2D)	(None, 4, 4, 128)	73,856	enc_conv2[0][0]
enc_Flatten (Flatten)	(None, 2048)	0	enc_conv3[0][0]
z_mean (Dense)	(None, 2)	4,098	enc_Flatten[0][0]
z_log_var (Dense)	(None, 2)	4,098	enc_Flatten[0][0]
sampling_1 (Sampling)	(None, 2)	0	z_mean[0][0], z_log_var[0][0]

Total params: 100,868 (394.02 KB)

Trainable params: 100,868 (394.02 KB)

Non-trainable params: 0 (0.00 B)

Decoder of VAE

In [24]:
```
dec_in0 = Input(shape=(2,), name='dec_input')
X = Dense(np.prod(enc_out_shape), name='dec_dense')(dec_in0)
X = Reshape(enc_out_shape)(X)
X = Conv2DTranspose(filters=128, kernel_size=(3,3), strides=2, activation='relu', p
X = Conv2DTranspose(filters=64, kernel_size=(3,3), strides=2, activation='relu', pa
X = Conv2DTranspose(filters=32, kernel_size=(3,3), strides=2, activation='relu', pa

dec_out0 = Conv2D(1, kernel_size=(3,3), strides=1, activation='sigmoid', padding='s

decoder = models.Model(dec_in0, dec_out0, name="Decoder")
decoder.summary()
```

Model: "Decoder"

Layer (type)	Output Shape	Param #
dec_input (InputLayer)	(None, 2)	0
dec_dense (Dense)	(None, 2048)	6,144
reshape_3 (Reshape)	(None, 4, 4, 128)	0
dec_convT1 (Conv2DTranspose)	(None, 8, 8, 128)	147,584
dec_convT2 (Conv2DTranspose)	(None, 16, 16, 64)	73,792
dec_convT3 (Conv2DTranspose)	(None, 32, 32, 32)	18,464
dec_conv (Conv2D)	(None, 32, 32, 1)	289

Total params: 246,273 (962.00 KB)

Trainable params: 246,273 (962.00 KB)

Non-trainable params: 0 (0.00 B)

- **tf.reduce_mean** : Calculates the average (mean) of elements across specified dimensions in a tensor.
- **tf.reduce_sum** : Computes the sum of elements across specified dimensions in a tensor.

In [25]:
```
def preprocess(images):
    images = images.astype("float32") / 255.0
    images = np.pad(images, ((0, 0), (2, 2), (2, 2)), constant_values=0.0)
    images = np.expand_dims(images, -1)
    return images
```

In [26]:
```
X_train = preprocess(X_train)
X_test = preprocess(X_test)
```

Additional loss term

KL divergence loss is a key player in VAEs. It acts like a guide, nudging the latent space towards a preferred distribution while preventing it from becoming a blurry mess. This keeps the space informative, allowing the VAE to capture the essence of the data for generating novel and diverse outputs. Explaining the concept of KL divergence loss might be a little too much for the scope of this study and thus is not explained in detail.

Model Training

Note : the model are actually trained over 30 epochs each, 27 epochs first and then 3 extra to improve the document readablility

Base Autoencoder

In [28]:
```
Autoencoder = models.Model(enc_in, decoder0(enc_out), name='autoencoder')
```

In [29]:
```
Autoencoder.compile(optimizer='adam', loss='binary_crossentropy')
```

In [31]:
```
Autoencoder.fit(X_train, X_train, epochs=3, batch_size=10, validation_data=(X_test,
```

```
Epoch 1/3
6000/6000 ━━━━━━━━━━━━━━━━━━━━ 18s 3ms/step - loss: 0.2422 - val_loss: 0.2458
Epoch 2/3
6000/6000 ━━━━━━━━━━━━━━━━━━━━ 18s 3ms/step - loss: 0.2417 - val_loss: 0.2451
Epoch 3/3
6000/6000 ━━━━━━━━━━━━━━━━━━━━ 18s 3ms/step - loss: 0.2420 - val_loss: 0.2448
```

Out[31]: <keras.src.callbacks.history.History at 0x7e0330f49c00>

Variational Autoencoder

In [32]:
```
vae = VAE(encoder, decoder)
```

In [33]:
```
vae.compile(optimizer='adam')
```

In [35]:
```
vae.fit(X_train, epochs=3, batch_size=10, validation_data=(X_test, X_test))
```

```
Epoch 1/3
6000/6000 ━━━━━━━━━━━━━━━━━━━━ 20s 3ms/step - kl_loss: 5.6844 - reconstruction_loss
: 122.9991 - total_loss: 128.6836 - val_kl_loss: 6.7554 - val_loss: 120.2139 - val_
reconstruction_loss: 113.4585
Epoch 2/3
6000/6000 ━━━━━━━━━━━━━━━━━━━━ 20s 3ms/step - kl_loss: 5.6776 - reconstruction_loss
: 123.1029 - total_loss: 128.7807 - val_kl_loss: 6.6026 - val_loss: 122.1582 - val_
reconstruction_loss: 115.5556
Epoch 3/3
6000/6000 ━━━━━━━━━━━━━━━━━━━━ 20s 3ms/step - kl_loss: 5.6743 - reconstruction_loss
: 123.0349 - total_loss: 128.7090 - val_kl_loss: 6.9831 - val_loss: 121.6734 - val_
reconstruction_loss: 114.6903
```

Out[35]: <keras.src.callbacks.history.History at 0x7e0332dcc0a0>

Prediction and Visualizations

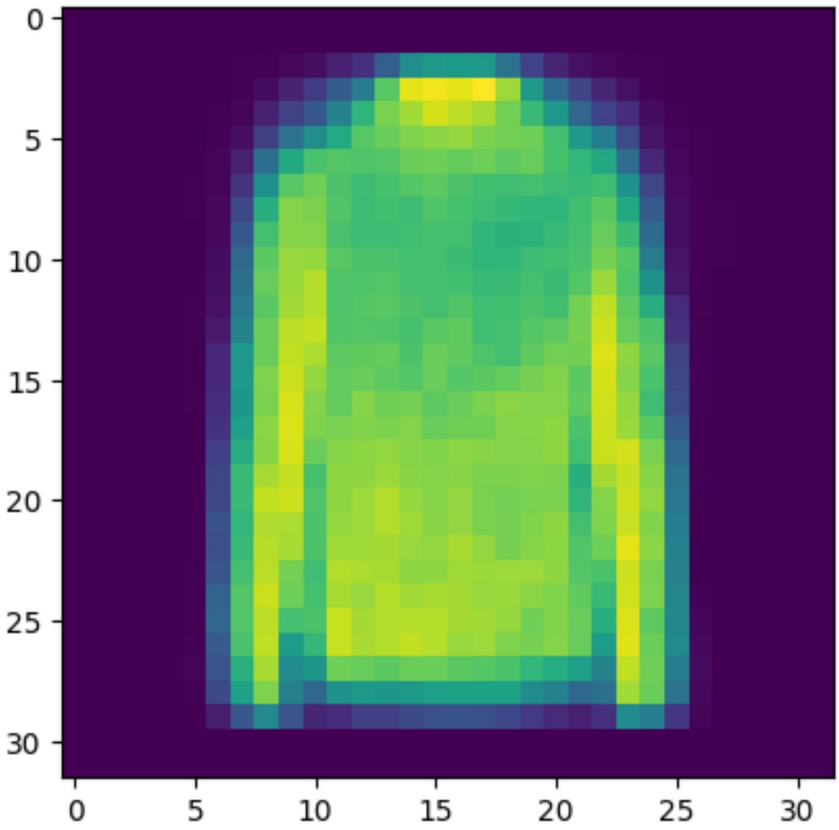
In [37]:
```
size = 100000
eg = X_test[:size]
eg_labels = y_test[:size]
```

Example of an image generated from the VAE model

In [40]:
```
p = decoder.predict(np.array([[-0.5, -0.5]]))
plt.imshow(p[0])
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 368ms/step
```

Out[40]: <matplotlib.image.AxesImage at 0x7e03332bbcd0>



In [60]:
```
z_mean, z_log_var, z = encoder.predict(eg)
pred = encoder0.predict(eg)
```

```
313/313 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step
313/313 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step
```

In [42]:
```
pred = np.array(pred)
```

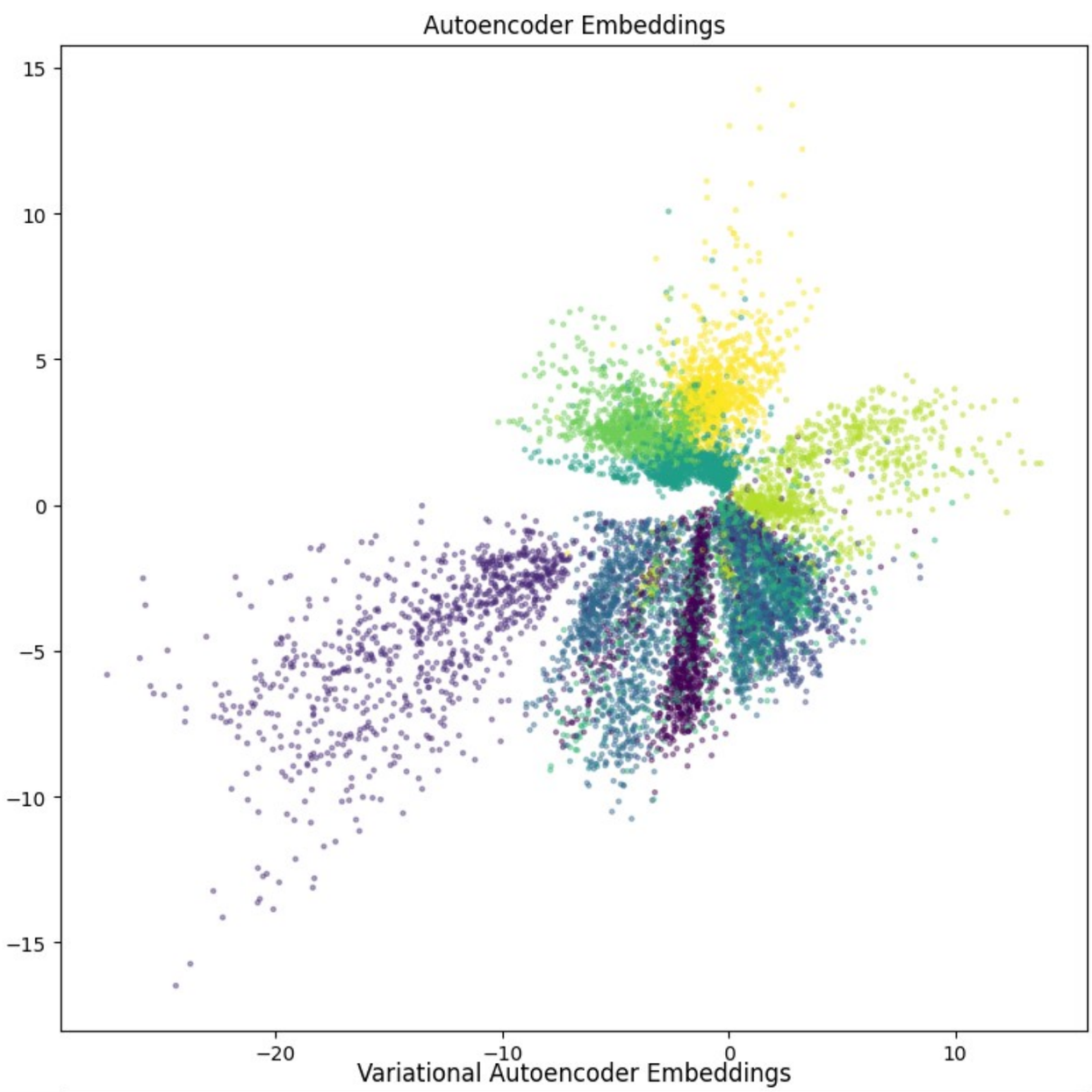
Visualizing the Latent space for both models

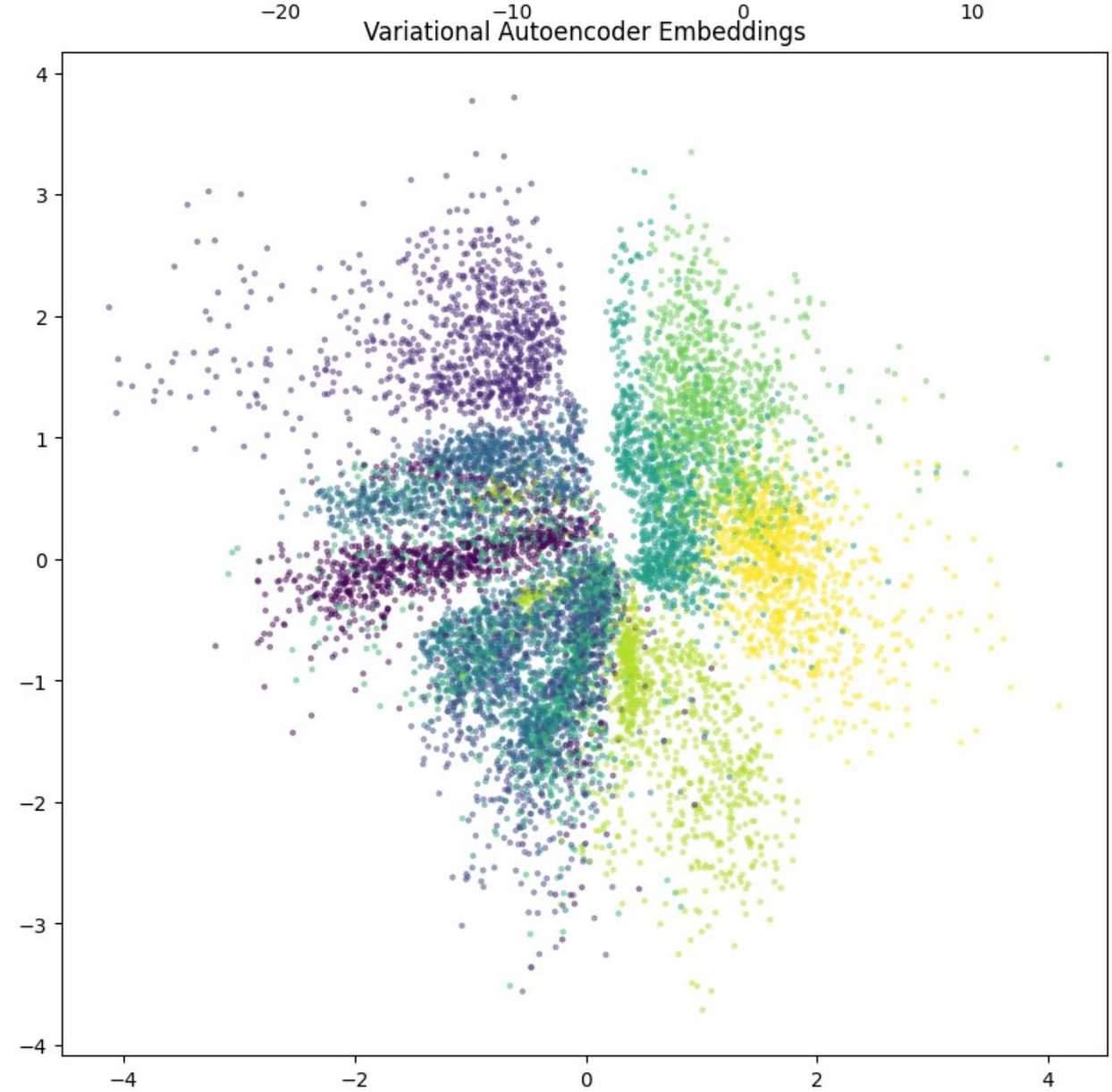
- We can see the range of the latent space in the base Autoencoder is much larger
- In VAE the points tends to spread symmetrically around origin
- The points are distributed in much clearer groups in VAE as compared to the Base Autoencoder
-

In [45]:
```
plt.figure(figsize=(10, 8), dpi=200)
fig, ax = plt.subplots(2, 1, figsize=(8, 15))
fig.tight_layout()

ax[0].scatter(pred[:,0], pred[:,1], alpha=0.4, s=5, c = eg_labels)
ax[1].scatter(z[:,0], z[:,1], alpha=0.4, s=5, c = eg_labels)
ax[0].set_title('Autoencoder Embeddings')
ax[1].set_title('Variational Autoencoder Embeddings')
```

Out[45]:
```
Text(0.5, 1.0, 'Variational Autoencoder Embeddings')
<Figure size 2000x1600 with 0 Axes>
```


Autoencoder Embeddings
15
10
5
0
−5
−10
−15
−20
−10
0
10


Variational Autoencoder Embeddings
4
3
2
1
0
−1
−2
−3
−4
−4
−2
0
2
4

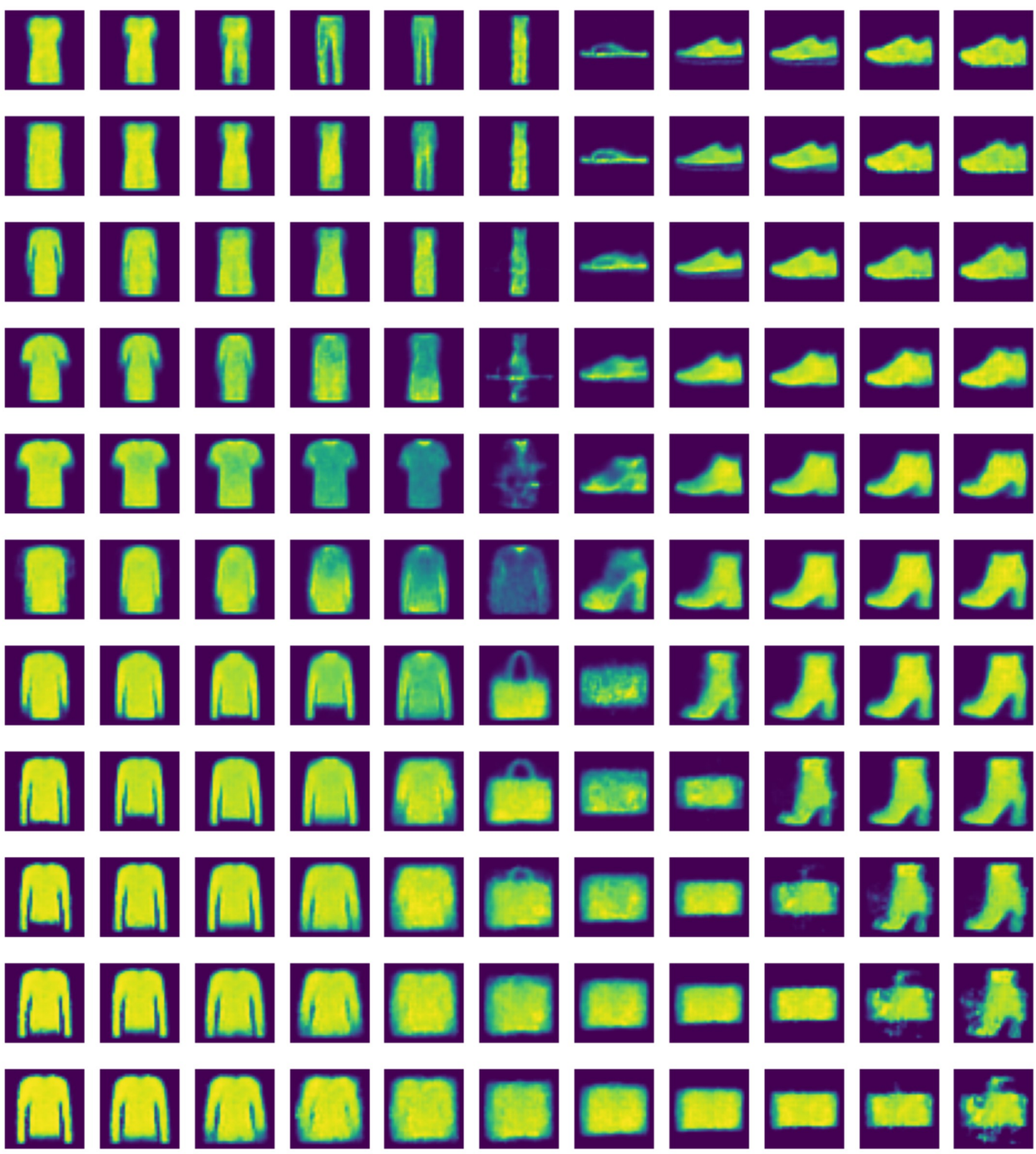
In [46]:
```
def display(arr, dec):
    rec = dec.predict(arr, verbose=False)
    fig, ax = plt.subplots(1, 11,figsize=(12, 3));
    for i in range(11):
        ax[i].axis('off');
        ax[i].imshow(rec[i]);
```

In [47]:
```
def gen(i, j, decoder):
    sam_lis = np.array(range(-5,6))
    sam_lis = sam_lis.astype('float32')
    lis = [[i,j*s] for s in sam_lis]
    display(np.array(lis), decoder)
```

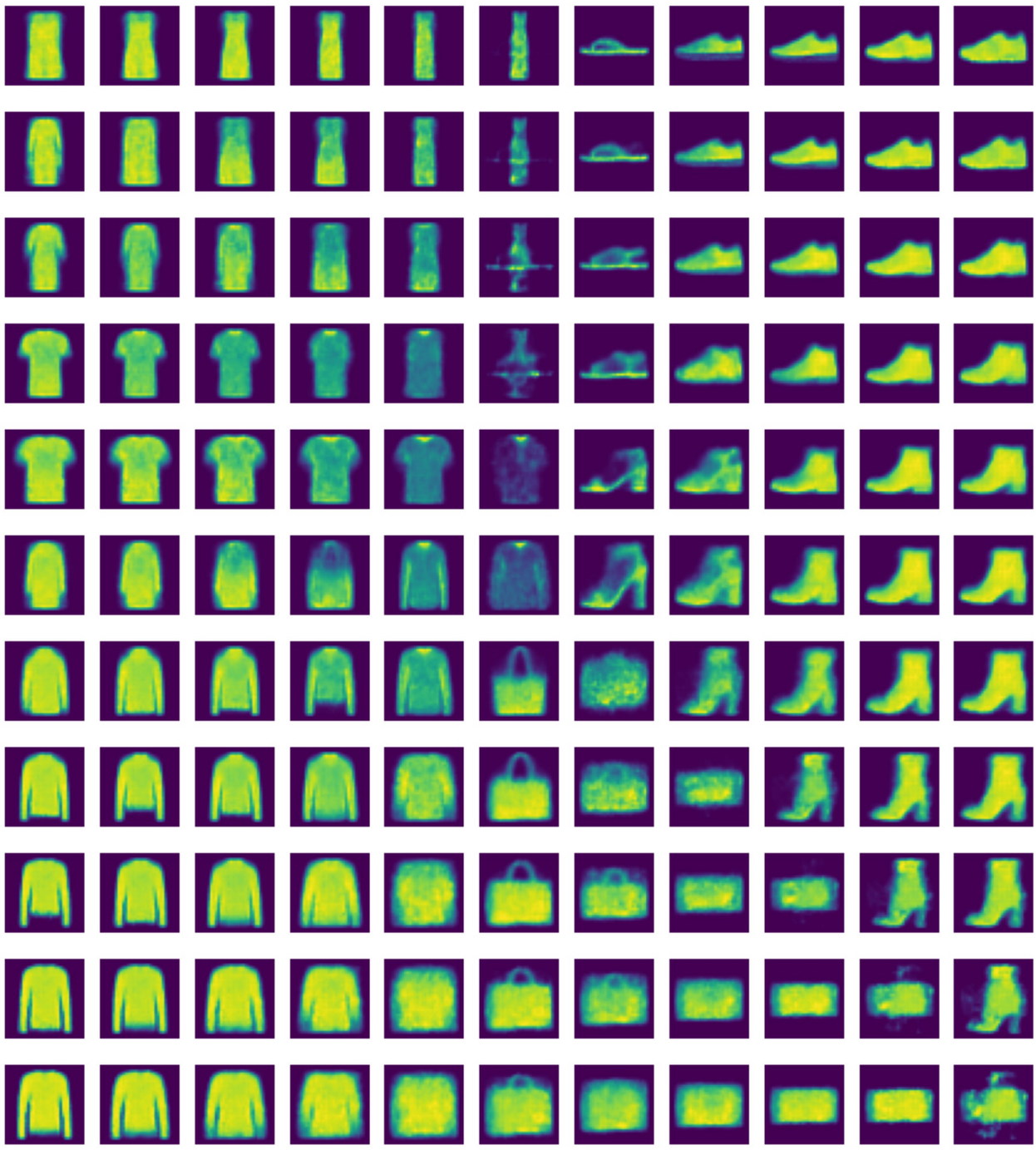
Generating Images from various parts of the latent space

Base Autoencoder's Genrations

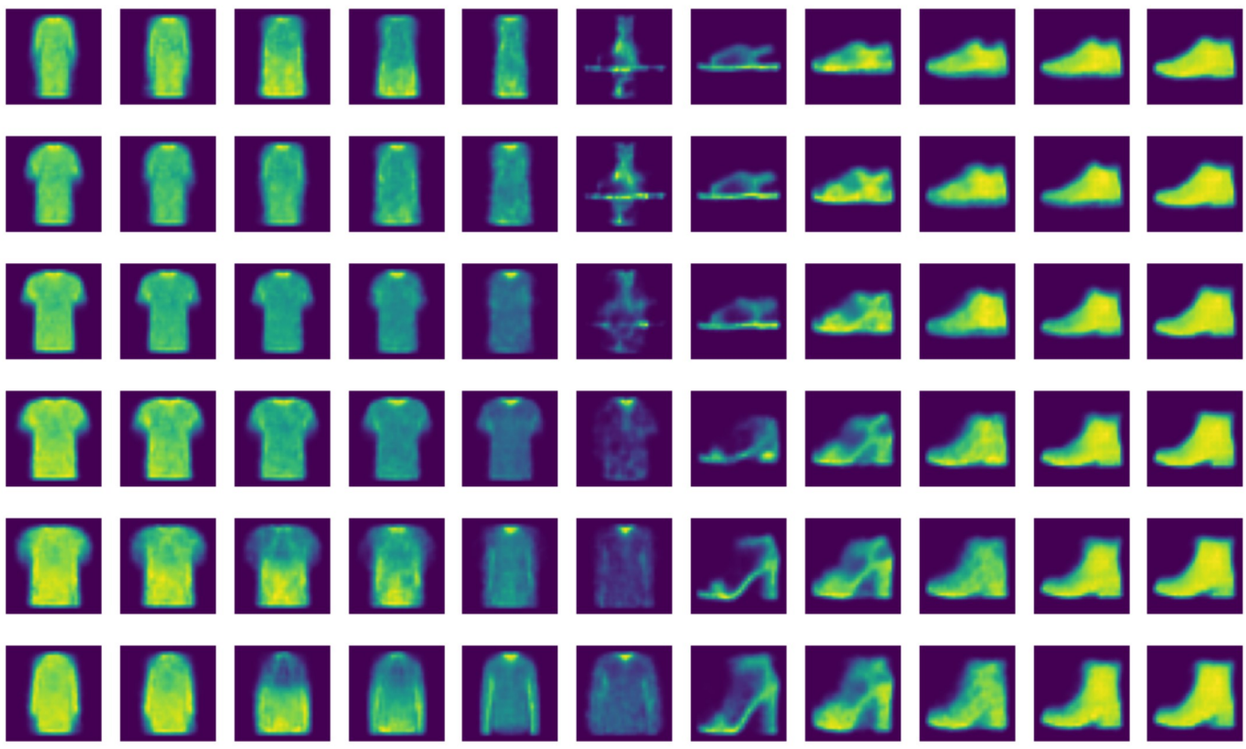
In [54]:
```
for i in np.linspace(-8, 8, 11):
    gen(i, 1.5, decoder0)
```

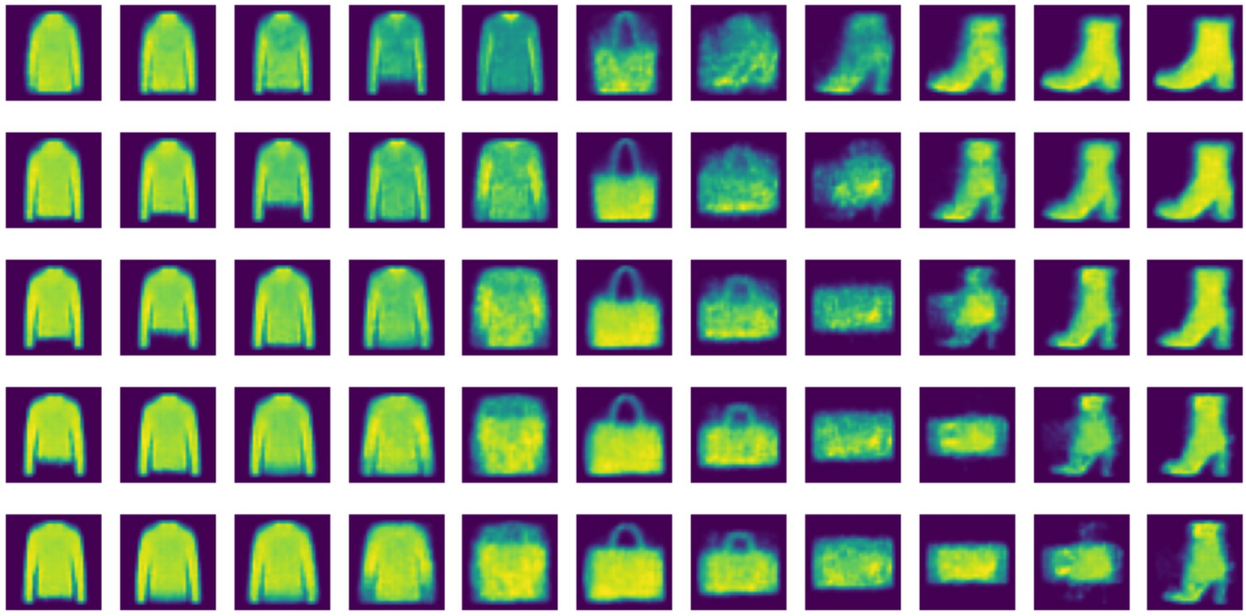


In [55]:
```
for i in np.linspace(-5, 5, 11):
    gen(i, 1, decoder0)
```



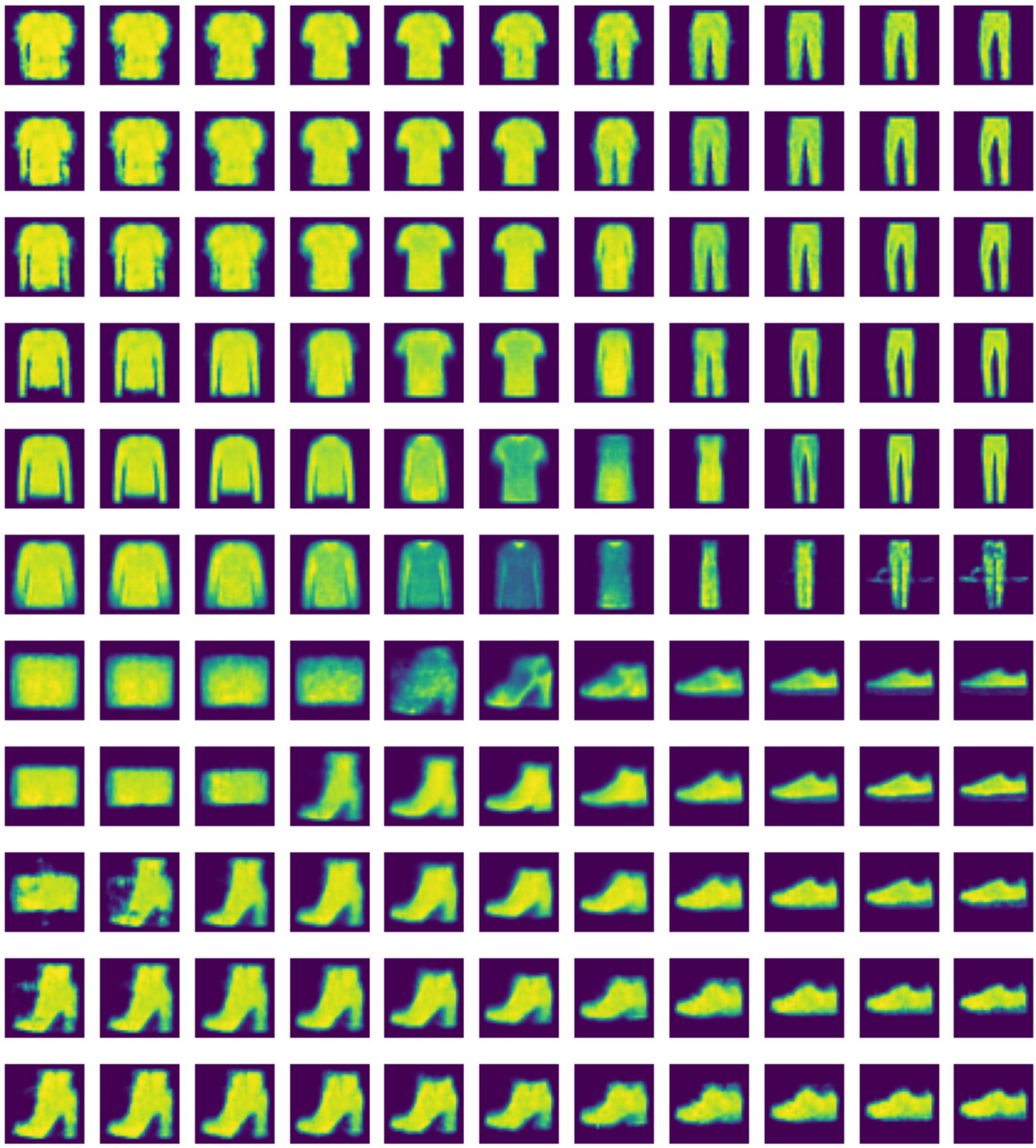
In [56]:
```
for i in np.linspace(-3, 3, 11):
    gen(i, 0.8, decoder0)
```



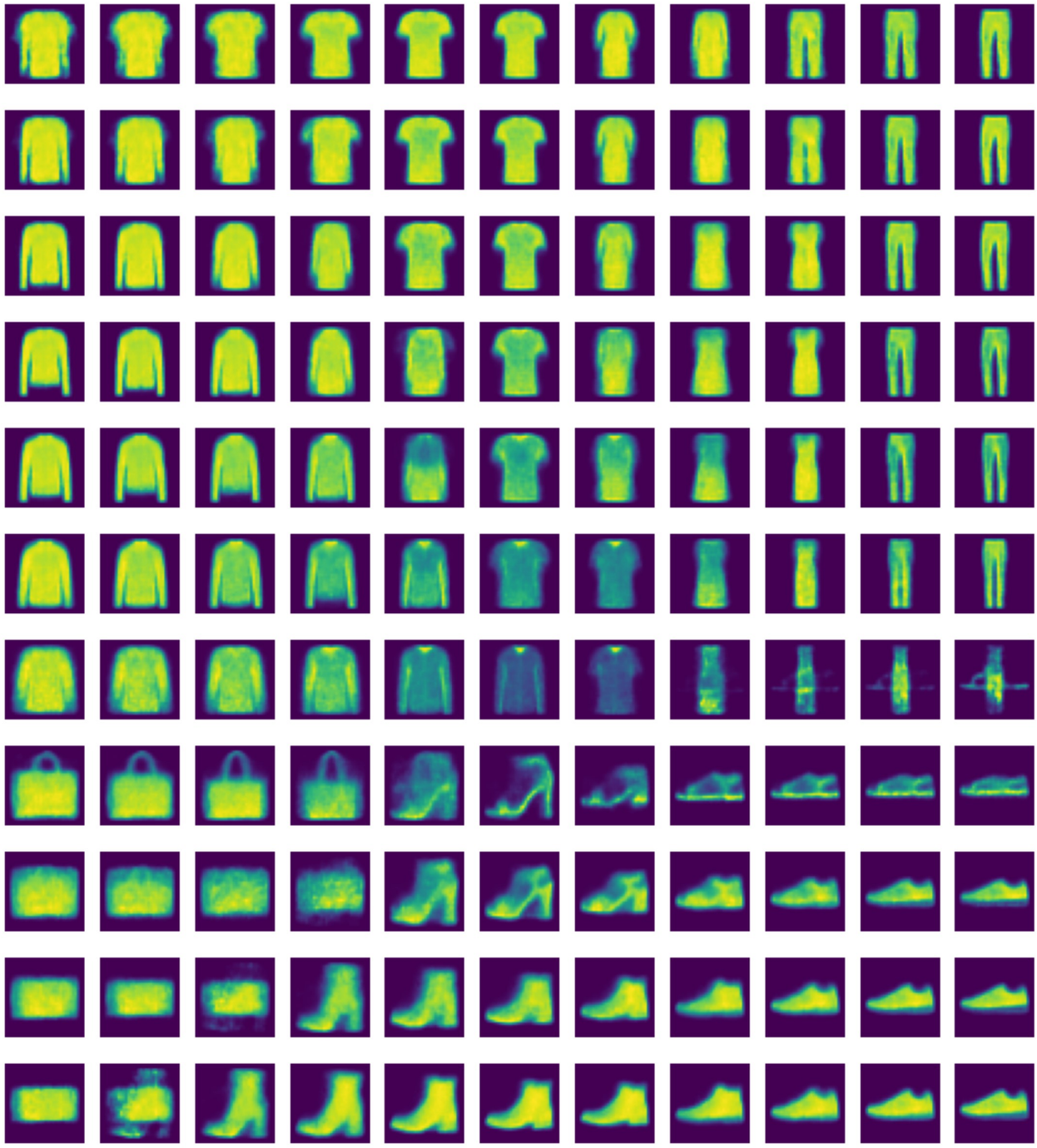


Variatioal Autoencoder's Generations

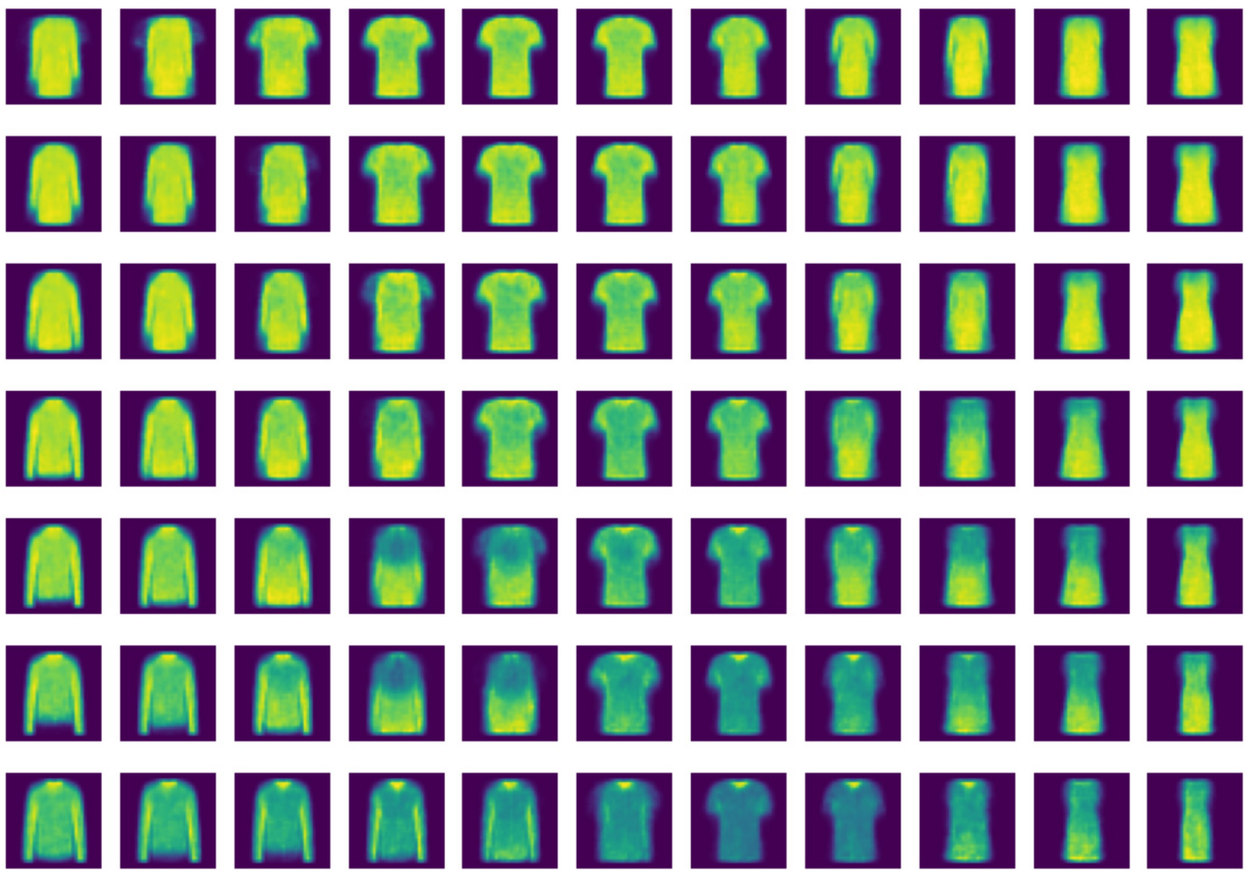
In [57]:
```
for i in np.linspace(-4, 4, 11):
    gen(i, 0.5, decoder)
```

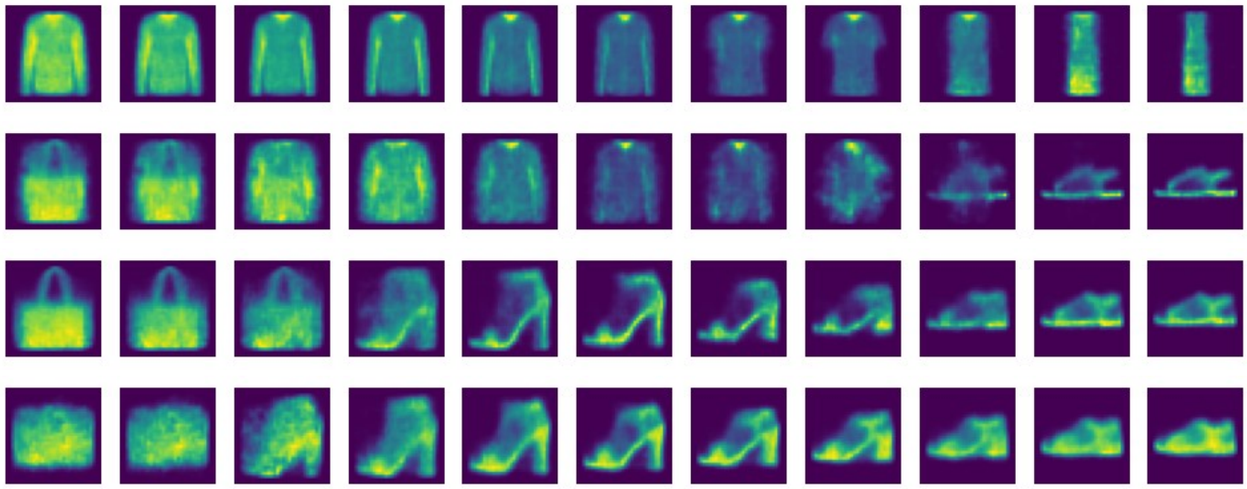


In [58]:
```
for i in np.linspace(-2, 1.5, 11):
    gen(i, 0.3, decoder)
```



In [59]:
```
for i in np.linspace(-1.5, 0.7, 11):
    gen(i, 0.15, decoder)
```





Thank you for your checking out 😄