Deep Learning on Computational Accelerators

(236781)

Variational AutoEncoders

General notes:

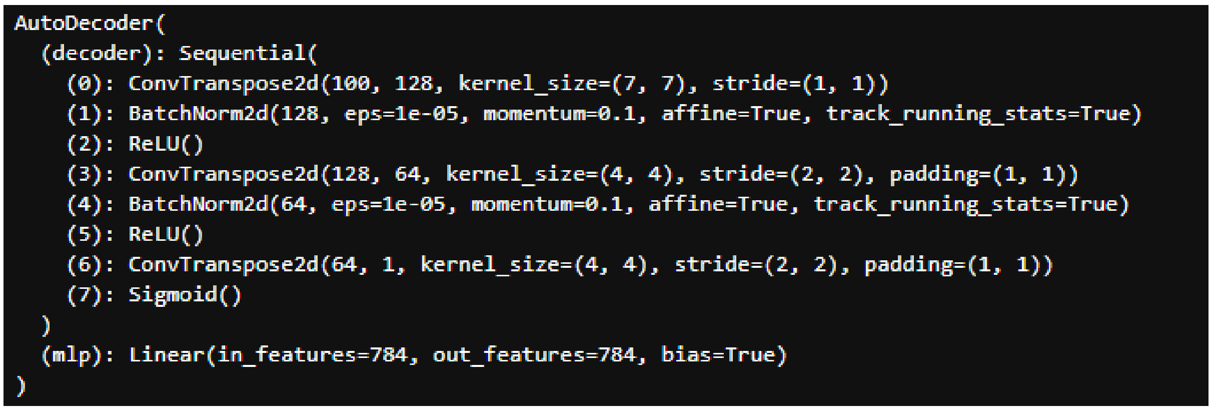
1) When training the VAD model, after a few unsuccessful trials, we have decided to help the model learn latent space by initializing parameter vectors in a specific way. For each class we have randomly sampled parameters (10 sets of parameters, one per each class), and then assigned to each datapoint a set of parameters according to its label. This way each class starts training in a different region in latent space, but also each datapoint still has an ability to learn its unique parameters.

   During the evaluation of both the train and test sets, we assign a randomly sampled unique set of parameters to each datapoint.

   For the test set, we see that the t-SNE is pretty divided, although it looks more natural to the dataset. This tells us that the model has learned the latent space, and not just left the initial train parameters unchanged during training.

Wet Assignment

1.3.1
(1)



```
AutoDecoder(
  (decoder): Sequential(
    (0): ConvTranspose2d(100, 128, kernel_size=(7, 7), stride=(1, 1))
    (1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (2): ReLU()
    (3): ConvTranspose2d(128, 64, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1))
    (4): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (5): ReLU()
    (6): ConvTranspose2d(64, 1, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1))
    (7): Sigmoid()
  )
  (mlp): Linear(in_features=784, out_features=784, bias=True)
)
```

Auto Decoder architecture: It consists of 3 blocks, each containing Transpose Convolution. In the first 2 blocks we also have batch normalisation and the ReLU activation function , and in the last block we only add a Sigmoid activation. After decoding we have a Linear layer, finishing the computation.

We chose to use Convolution, as it is suited for the models that are working with images. The BatchNorm layers help us by stabilising the learning process, fighting exploding and vanishing gradient problems.
Activation functions are there to introduce non-linearity. The Sigmoid activation sets the values of the output to a range that suits grayscale pictures.
The linear layer helps to add some additional extraction abilities that will transform the patterns created by the convolutions, into the image itself.

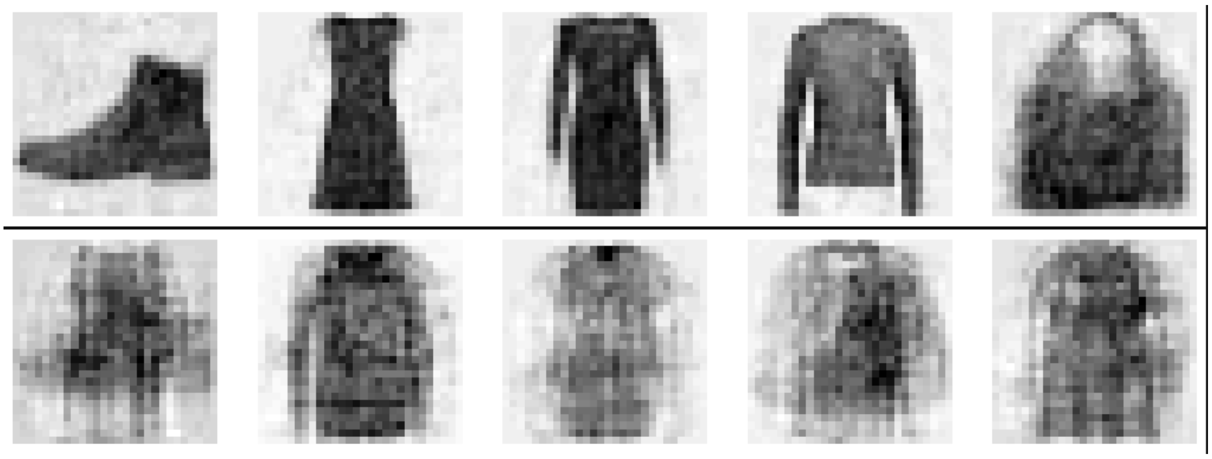
Model Parameters
The latent dimension (z_dim) was chosen after numerous trials so that it remained small enough to act as a bottleneck, but large enough to capture the necessary information for better image restoration.

The kernel sizes and strides in the transposed convolution layers were chosen to progressively increase the spatial resolution of the image while reducing the number of channels. This configuration ensures smooth upsampling of the latent vector back to the original image dimensions.

Training Parameters
The model uses the Adam optimizer with a learning rate of 0.01. Adam was chosen because it adjusts the learning rate adaptively for each parameter, making it suitable for models with high-dimensional data, such as images, and ensuring faster convergence.

(2)
Evaluation loss on train set: 0.025210941152181476
Evaluation loss on test set: 0.11863428447395563

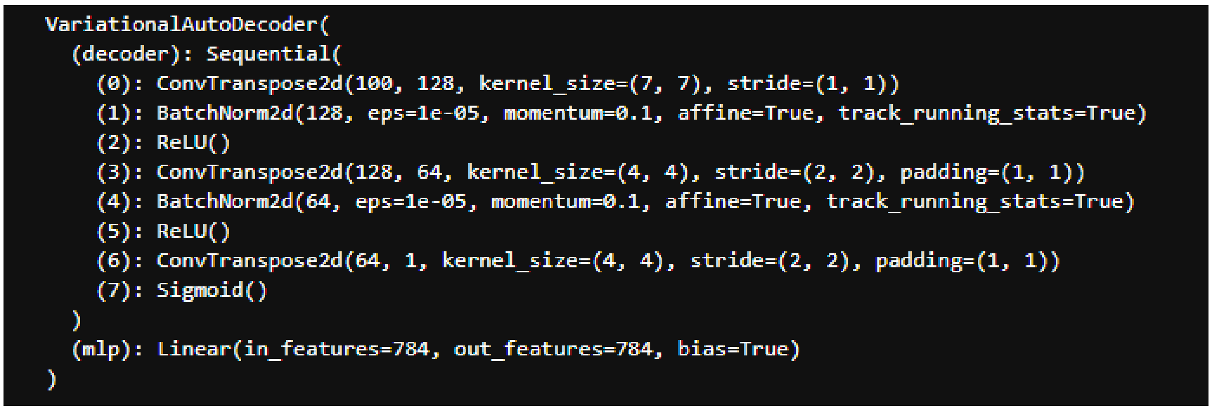
(3)
First row contains 5 images that were decoded from the test set latent vectors.
Second row contains the images that were decoded from the randomly sampled vectors.



We see that the decoded random vector looks like a combination of a lot of different items from different classes. It occurs because the latent space of the model is not divided enough and the vectors that we randomly sampled fall into a poorly ordered latent space region that can be decoded as different items.

1.3.2

(1)



```
VariationalAutoDecoder(
  (decoder): Sequential(
    (0): ConvTranspose2d(100, 128, kernel_size=(7, 7), stride=(1, 1))
    (1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (2): ReLU()
    (3): ConvTranspose2d(128, 64, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1))
    (4): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (5): ReLU()
    (6): ConvTranspose2d(64, 1, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1))
    (7): Sigmoid()
  )
  (mlp): Linear(in_features=784, out_features=784, bias=True)
)
```

Our implementation of a Variational Auto Decoder uses a combination of a Convolutional Neural Network (which is well-suited for image data due to its ability to capture spatial relationships within feature maps) in the decoder and a fully connected layer (MLP) to tweak 28x28 grayscale images after the CNN's computations.

Model Architecture

The architecture uses transposed convolution layers to upsample the latent vector back to the image size.

- The first layer transforms the latent vector from z_dim(100) to a feature map with 128 channels. This layer captures high-level features from the latent space.

- The second layer reduces the channel size to 64, thus refining the feature maps.

- The final layer brings the output to 28x28 (the target image size), with 1 output channel to match the grayscale image format.

BatchNorm is applied after each convolution to stabilize training and help the model converge faster. The ReLU function is applied after the batch normalization to introduce non-linearity and prevent the vanishing gradient problem. A sigmoid activation function is used at the output to constrain the pixel values in the range of [0, 1], in order to then normalize the output to the desired [0,255] range.

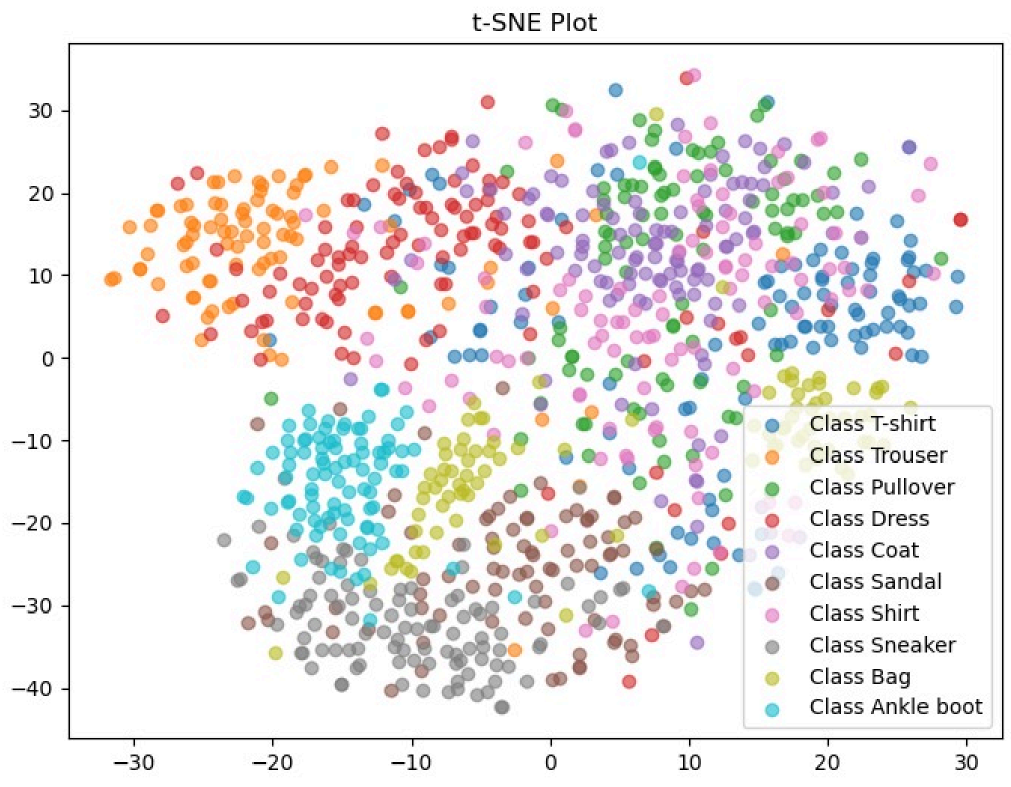
The MLP (a single linear layer) follows the decoder. After the image has been upscaled to its original dimensions (28x28), it is passed through the fully connected layer. It helps the model further learn patterns or correct any errors in the output that may not have been fully addressed by the transposed convolutions.

Model Parameters
The latent dimension (z_dim) was chosen after numerous trials so that it remained small enough to act as a bottleneck, but large enough to capture the necessary information for better image restoration.

The kernel sizes and strides in the transposed convolution layers were chosen to progressively increase the spatial resolution of the image while reducing the number of channels. This configuration ensures smooth upsampling of the latent vector back to the original image dimensions.

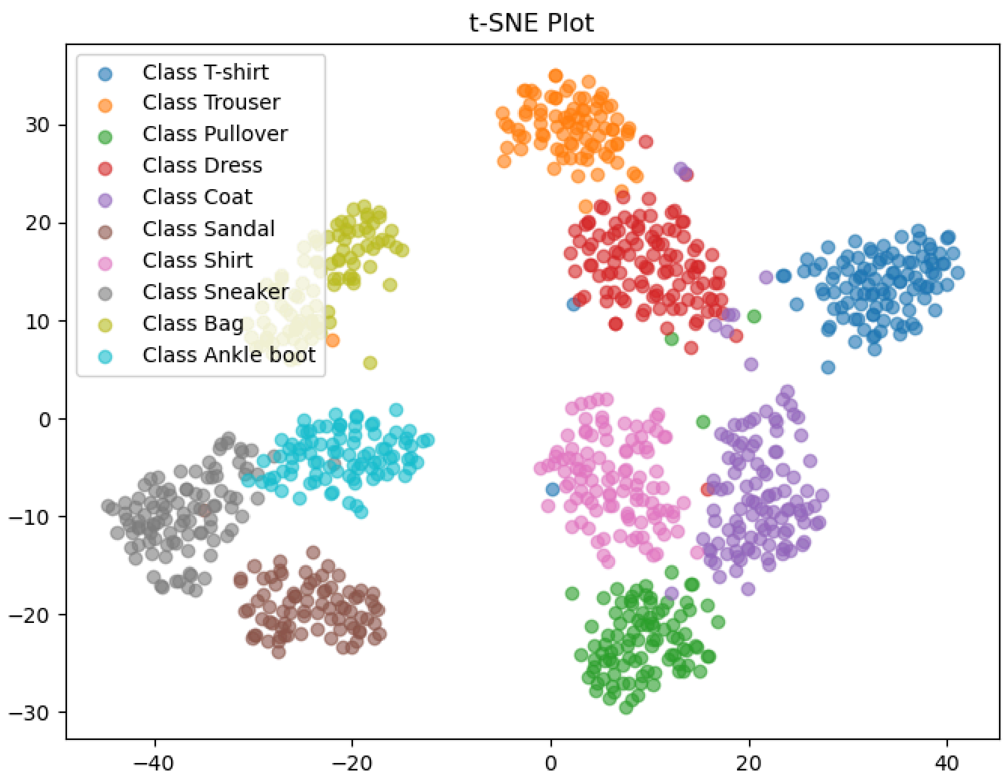
Training Parameters
The model uses the Adam optimizer with a learning rate of 0.01. Adam was chosen because it adjusts the learning rate adaptively for each parameter, making it suitable for models with high-dimensional data, such as images, and ensuring faster convergence.

(2) AD model t-SNE:



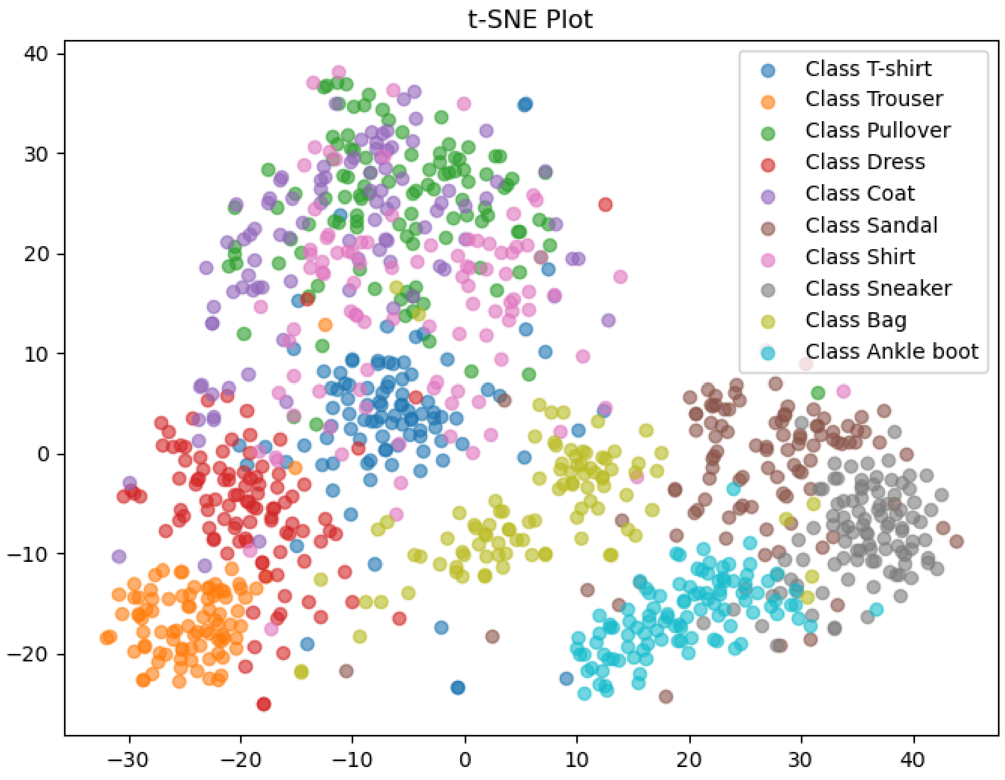
(3) Train set evaluation

$$\mathit{Train\ evaluation\ loss}\ \approx\ 0.0186$$

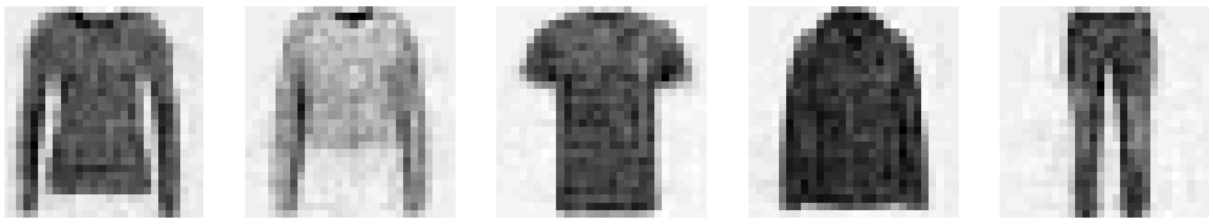


<u>Test set evaluation</u>

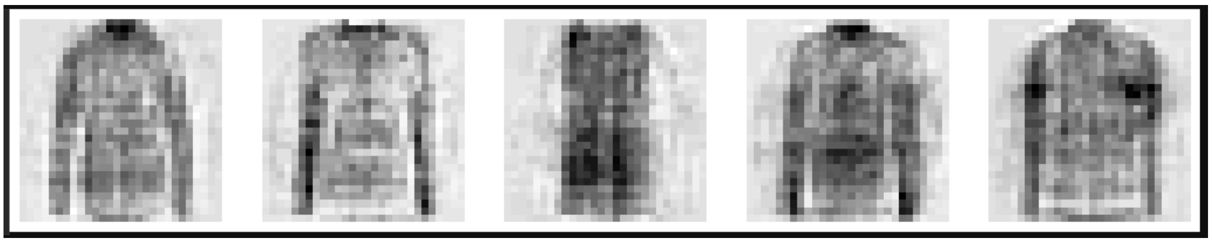
$$Test\ \ evaluation\, loss\ \approx 0.1112$$



Decoded test set:



Decoded random:



The difference in quality between images restored from the test set and those generated from random latent vectors (bottom row) arises from how the learned latent space is structured during training and how well it reflects the underlying data distribution.

Even though latent vectors for test images were randomly initialized, they were optimized during model evaluation to align with the corresponding images in the test set.

While the model has learned a rough distribution of the latent space, it does not guarantee that every possible latent vector will correspond to a high-quality image. If the latent vector happens to fall in a well-organized region of the space, the generated image might be good. However, if it falls in a disorganized or less-optimized region, the decoder might produce poor-quality images. Thus, the randomly generated images are of pure quality.

*We can see on the TSNE graph, that while some labels are well segregated, a big mass of data points remain disorganised and thus classes like Pullover or Shirt cannot be well generated.
This makes sense since images with these labels look alike and therefore the model cannot distinguish between them well enough.

(4) <u>Uniform distribution</u>

$$Uniform(x|a,b) = \frac{1}{b-a}, \ where\ a \leq x \leq b$$

Uniform distribution refers to a type of probability distribution in which all outcomes are equally likely. The reparametrization is given by:

$$Uniform(0,1) \cdot (b - a) + a, \ where\ a, b\ are\ learned\ parameters$$

<u>Laplace distribution</u>

$$CDF:\ Laplace(x|\mu, b) = \frac{1}{2b} \cdot exp\{\frac{-|x-\mu|}{b}\}$$

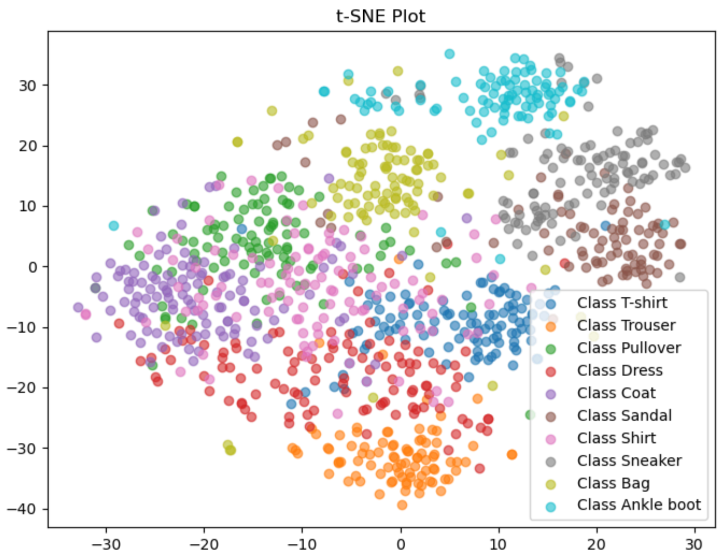
Where μ is the location parameter (the peak of the distribution), b is the scale parameter, controlling the "spread" of the distribution.

The Laplace distribution is symmetric around μ and is more robust to outliers. This makes it useful for tasks where we want to capture more variance or noise in the data compared to a Gaussian distribution. The reparametrization is given by:

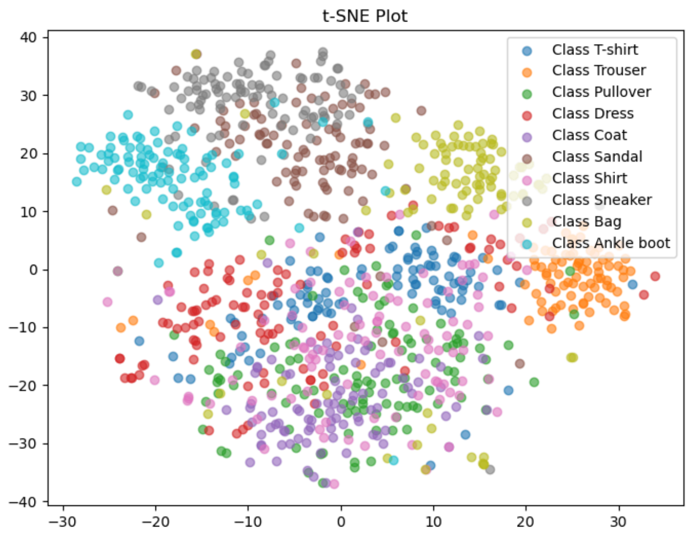
$$\mu - b \cdot sign(v) \cdot ln(1 - 2|v|), \ where\ v \sim Uniform(-\ 0.5, 0.5)$$

(5) Uniform distribution

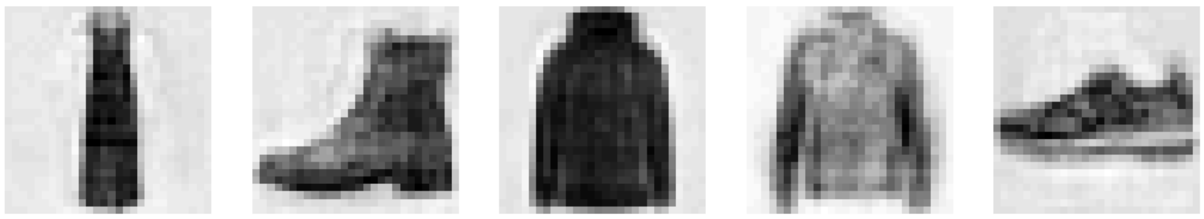
$$Train\ evaluation\ loss\ \approx 0.1457$$



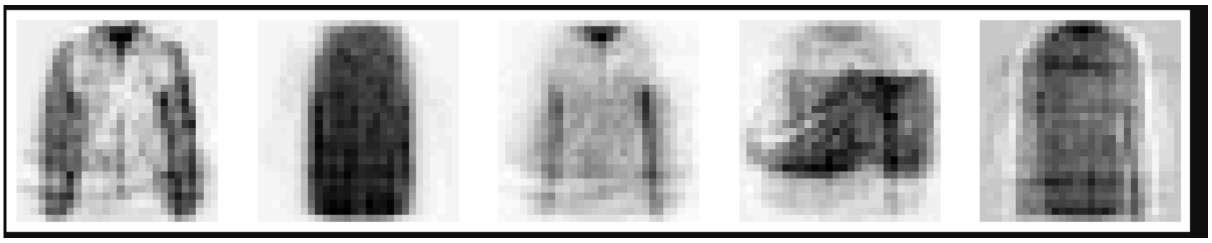
$$Test\ evaluation\ loss\ \approx 0.1644$$



Decoded test:

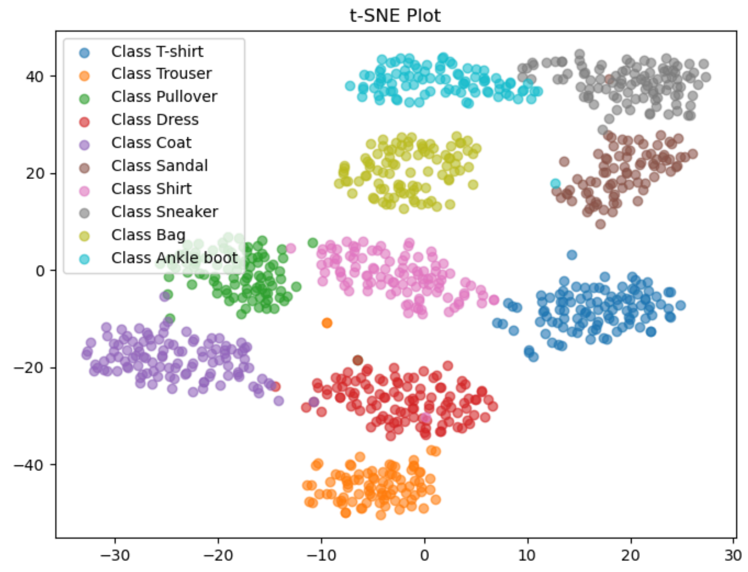


Decoded random:

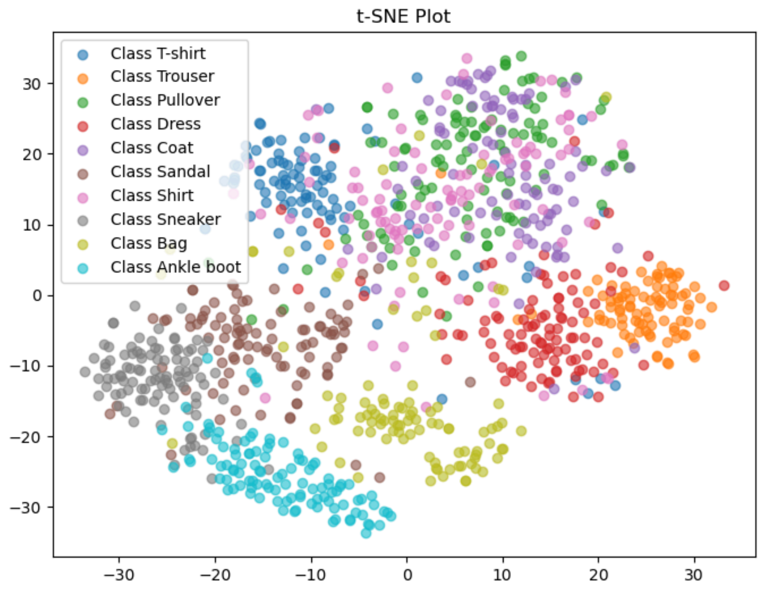


Laplace distribution

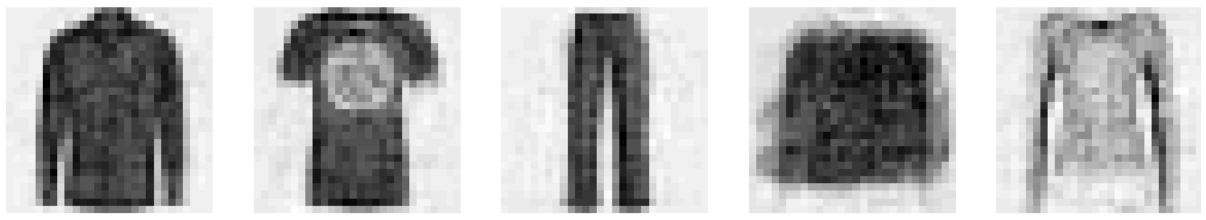
$$Train\ evaluation\ loss\ \approx\ 0.0204$$



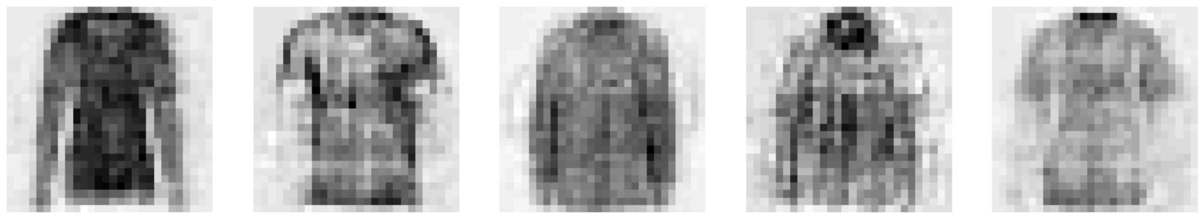
$$Test\ evaluation\ loss\ \approx\ 0.1118$$



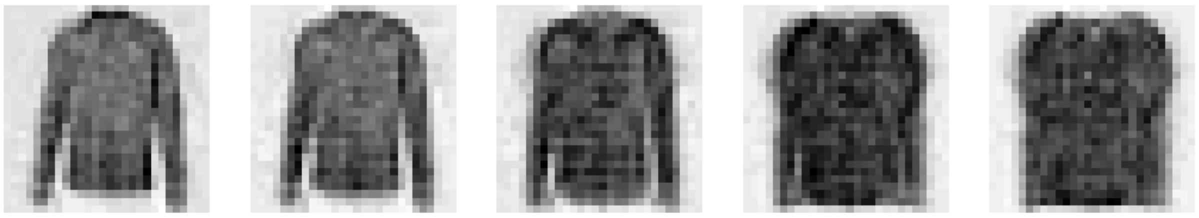
Decoded test set:



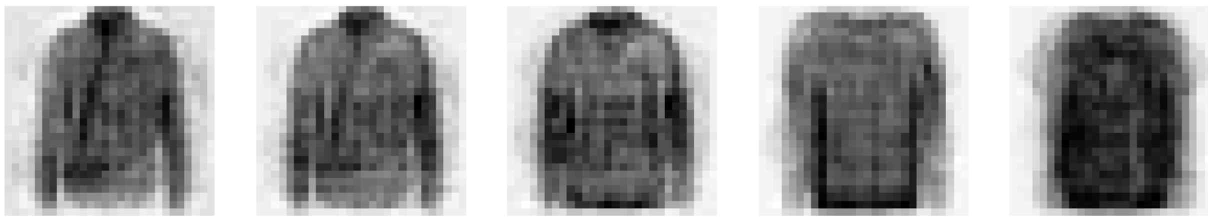
Decoded random:



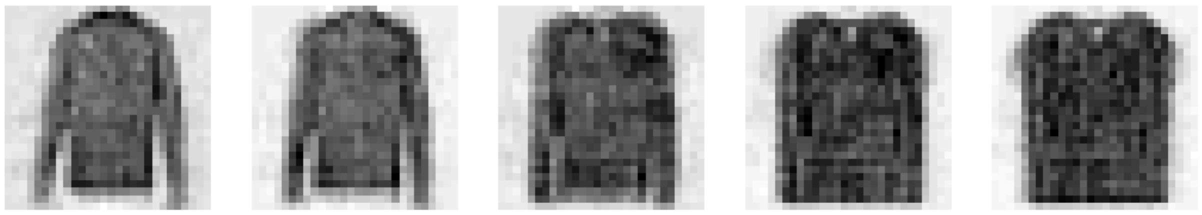
(6) <u>Normal distribution</u>



<u>Uniform distribution</u>



<u>Laplace distribution</u>



** All the images were generated after the test evaluation (evaluate_model run)