Deep Learning VAD Project Wet Part

Authors: Nativ Maor & Matan Kutz

October 27, 2024

1.3

1.3.1 Training an Auto-Decoder

1. An Overview of Auto-Decoder implementation

For the Auto-Decoder implementation we used an architecture based on the DecoderCNN of the GAN we used in HW3 part 2. The DecoderCNN was useful in generating images after it was trained as part of a GAN, so we believe the architecture has the capacity to extract useful information for classification and image generation tasks for the Fashion-MNIST data-set. The DecoderCNN architecture was based on the Decoder from the original DCGAN paper (by Alec Radford, Luke Mets, Soumith Chintala) and has proved its ability to generate realistic looking outputs from latent representations in various scenarios. It was published in a peer reviewed paper and had significant influence and reach. We took inspiration for our model parameters (like layer parameters, model depth, use of batch normalization ,activation function choices) and training parameters (optimization) and tuned around them for our cases goals.

The latent space dimension we chose was 128 - not too large so it would be a trivial problem, but not too small so it would still have the capacity to hold the information for the images reconstruction, it is close to an order of magnitude less than the full image number of pixels as 28x28=784 (which would be trivial).

The first layer of the network is a FC layer from the latent space to a much larger space of size 8192 (with a bias). The increase to high-dimensional latent space is typical to auto-decoder as it allows for richer construction.

The second component is the DecoderCNN which is composed of 3 components of {Deconvolution layer, Batch normalization and a Relu} followed by 2 additional Deconvolution layers to fix the output size with a Sigmoid multiplied by 255 at the end of the network to make all the values of the reconstructed image to be between 0 and 255 like the expected pixel value range.

The kernel size used in the 3 convolution components is 4x4 with stride 2x2 and padding 1x1 with no bias - from 512 to 256 to 128 channels. The depth and parameter choices for the kernel and batch normalization, and the activation function to be relu were inspired by the parameters in the DCGAN paper. The reason for batch normalization is to stabilize the training and is common practice

in training ANNs (the reasons for its effectiveness are in debate but the effect of it is evident).

After these 3 Deconvolution components there is a deconvolution layer from 128 to 1 (1 being the channel of the generated output), followed by a 1x1 deconvolution layer with kernel size 1x1, stride 1x1 and padding 2x2 to fit the generated image size to 28x28.

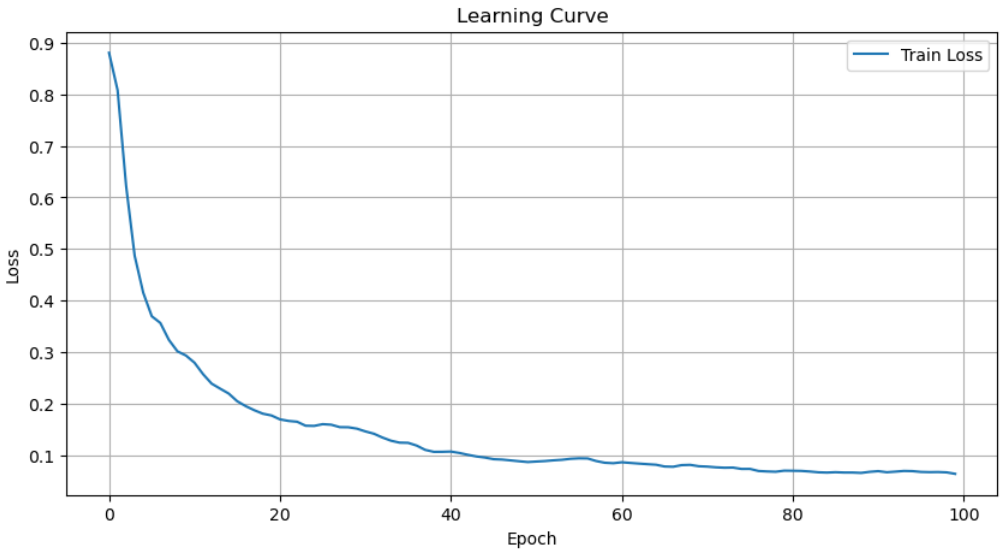
Finally, we use a sigmoid layer to fit the values of each cell of the 28x28 output to be between 0 and 1 and multiply by 255 to make the range of each cell between 0 and 255.

For the choices of the training parameters, we chose ADAM as our optimizer and used the default parameters with a learning rate of 10^{-3} (we experimented with various learning rates and found it to be the best in our search space).

To train the model we set the parameters to be optimized to be the train_latents together with the model parameters. We used the reconstruction loss provided in evaluate.py to as the loss function to be optimized.

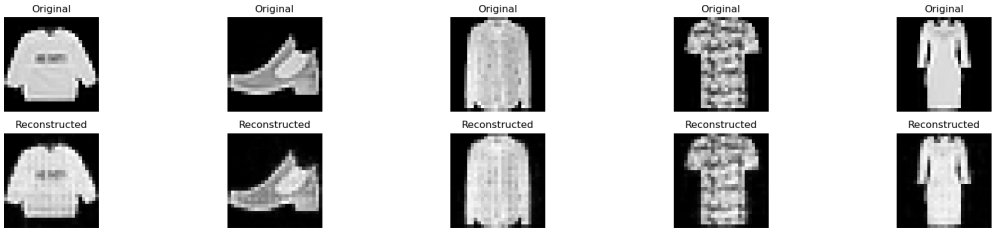
2. Evaluation of the AD model

The model training loss graph over 100 epochs:



After 100 epochs of training the reconstruction loss got down from a value around 0.9 to 0.06.

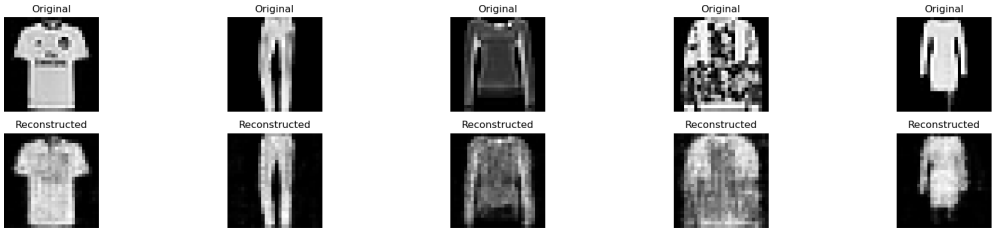
Visualization of the reconstructed images from the learned latent vectors for the train-set:



We see that the reconstruction is quite good for the training set and captures main and small features of the original images in the reconstruction.

As for the test-set, running for 100 epochs an ADAM optimizer to optimize only the test_latents with a lr of 10^-2 resulted with a reconstruction loss of 0.16.

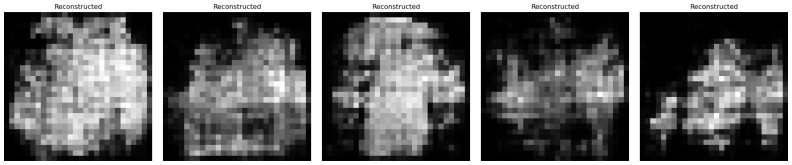
Example for reconstructed images:



We can see that the reconstruction is decent, it captures the general shape of the clothing objects. Although it does not capture small features of the images. But it was expected for it to perform less well than the train set.

3. Sampling from $\mathcal{N}(0, I)$:

Let's observe what sampling the latent vector from a standard normal distribution will decode into,



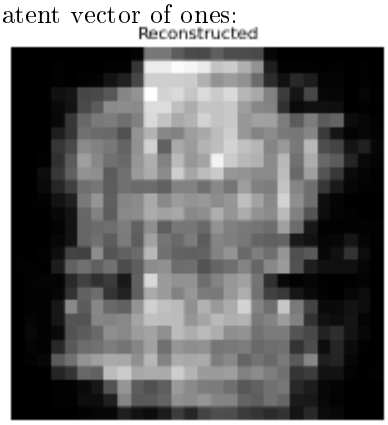
Some people might say they can recognize some similarity to clothing items in these decoding, like a heel show on the right and to its left a sneaker and further left a coat. But clearly these reconstructions are of very poor quality and are probably just wishful thinking and a fruit of our imagination knowing what we want to expect. It is not surprising that there will be some similarity to clothing items in the decoding, as the network was trained on clothing items, and it learned to fit vectors to it.

The better quality of the test cases reconstructions is resulted from the optimization on the test_latents to direct them in the direction of reconstructing something meaningful, they went through 100 epochs of training to fit as best as possible to some original image. That is while the totally random latents

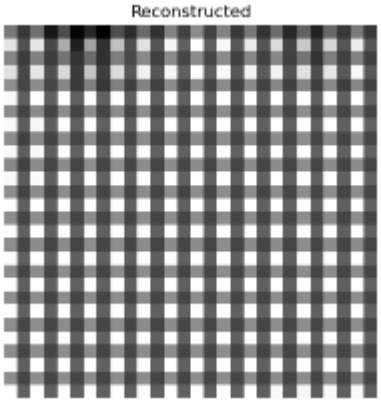
have nothing to train towards, there is nothing we can compare them to.

Addition, Just for fun, I tried to see what simple vectors and modification would result with:

A latent vector of ones:

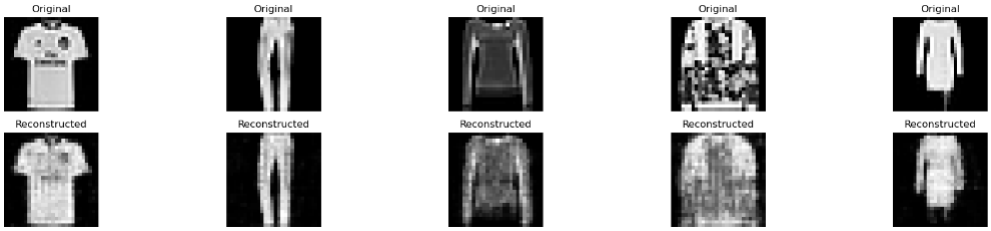


A latent vector of zeros:

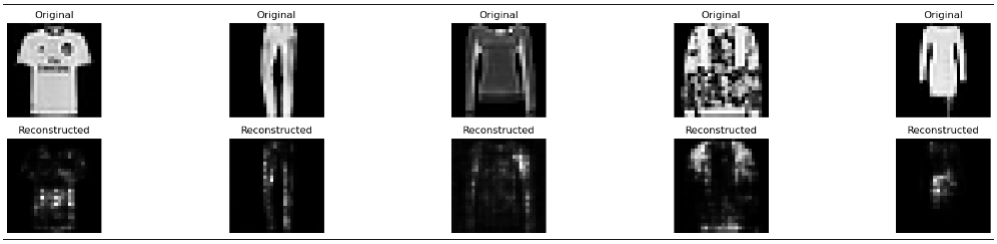


Also, while experimenting with it some more it seemed that changing only the magnitude of the latent vector seemed to have an effect on the clarity of the results:

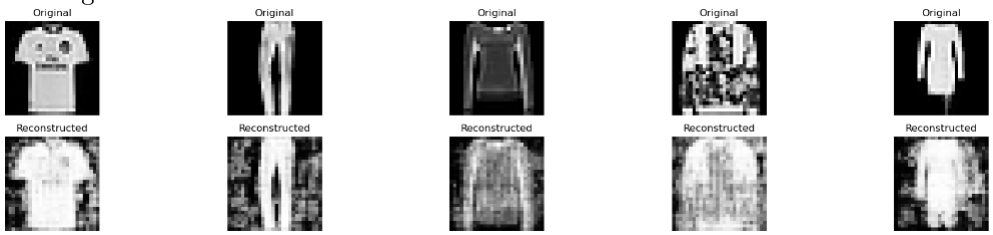
For example for the following 5 test_latents:



Half magnitude (multiplied by 0.5) results:



1.5 magnitude results:



We point this fact about the magnitude of the latents as it might be relevant in designing smarter architectures for the decoder taking the normalization of the latent vector into account.

1.3.2 Experimenting with VAD and the latent space

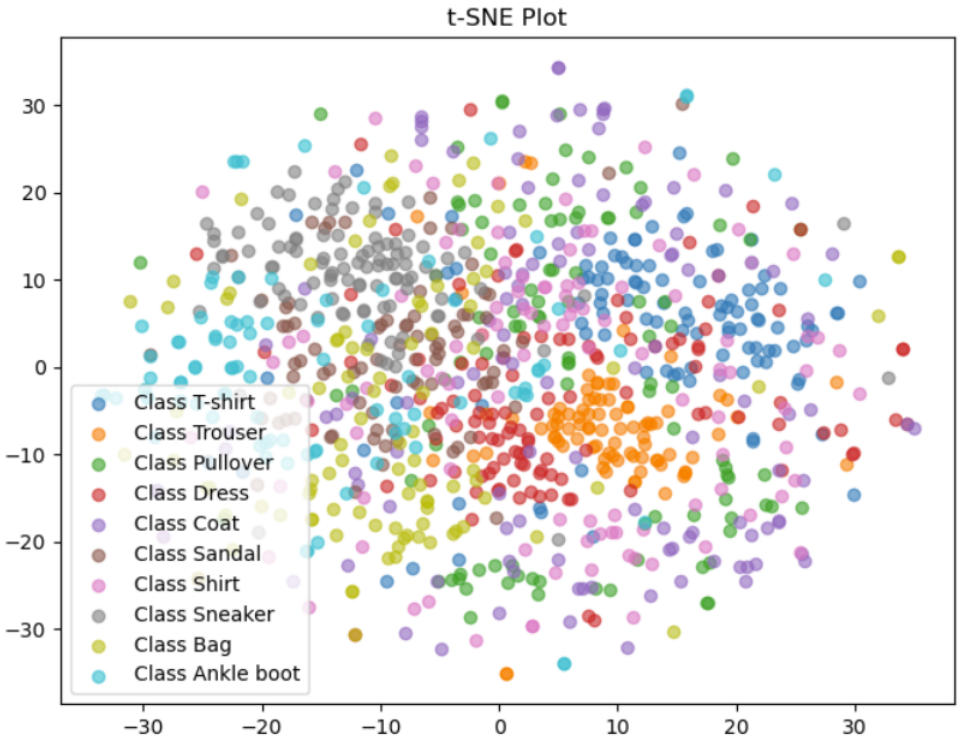
1. An Overview of Variational Auto-Decoder implementation

The architecture of the VAD model is composed of the AD model, when the training and evaluation are a little different. The reason we chose to base our VAD on the AD model from the previous section is that the AD model did show some reconstruction abilities, basing the VAD on it will "isolate the variables" and allow us to learn the strength and weaknesses of the VAD approach compared to it.

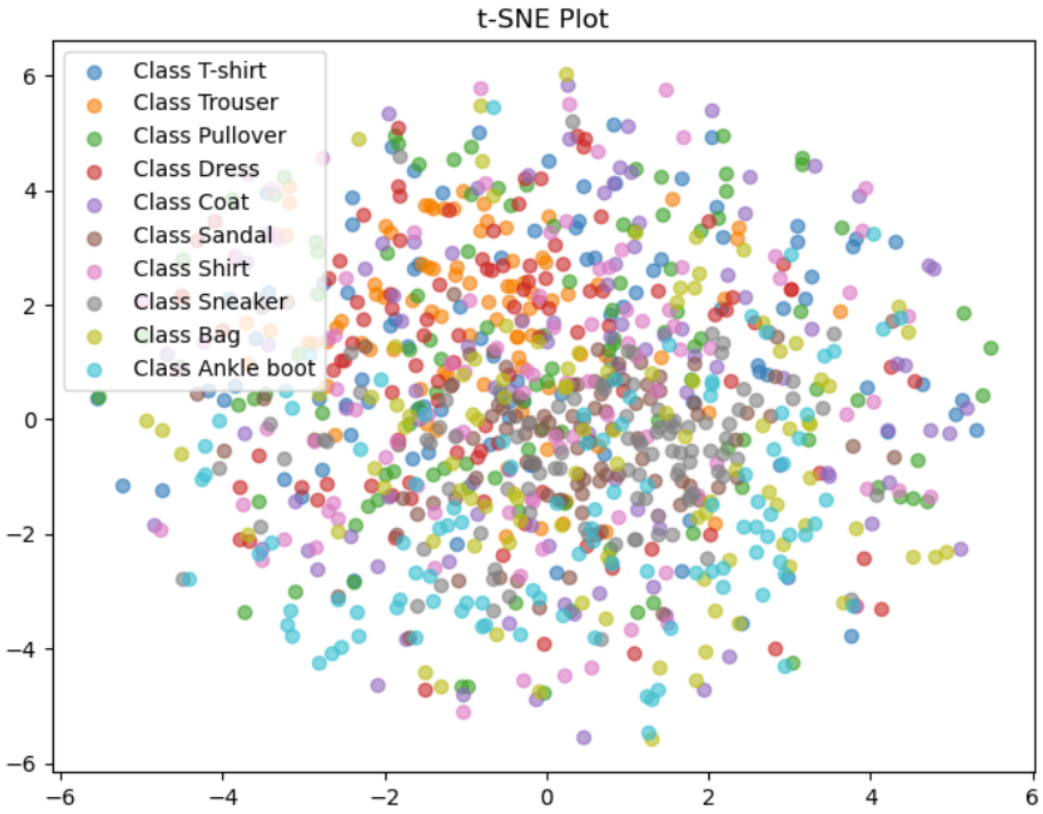
To train the VAD we pass it a distribution name, and distribution parameters for each sample in the training set. We sample from a standardized distribution (according to the distribution case we are checking) and compose a latent vector out of it using the distribution parameters by the reparametrization trick. We pass the latent vector through the AD model and get our output reconstruction. Upon training we optimize over the distribution parameter and the model parameters, upon evaluation we optimize only over the distribution parameters. The loss function is the ELBO loss which is composed of addition between the KL divergence term and the reconstruction loss, $loss_{total} = loss_{KL} + loss_{reconstruction}$. The model training was done over 100 or 200 epochs with ADAM optimizer with lr=10^-2 (which we found to work best for most after hyper-parameters search). For some cases we found that optimizing for more iterations, neglecting the KL-loss and only optimizing the reconstruction loss produced much better results.

2. Visualization of the latent space from the AD model using TSNE

For the train-set we get the following TSNE:



For the test-set we get the following TSNE:

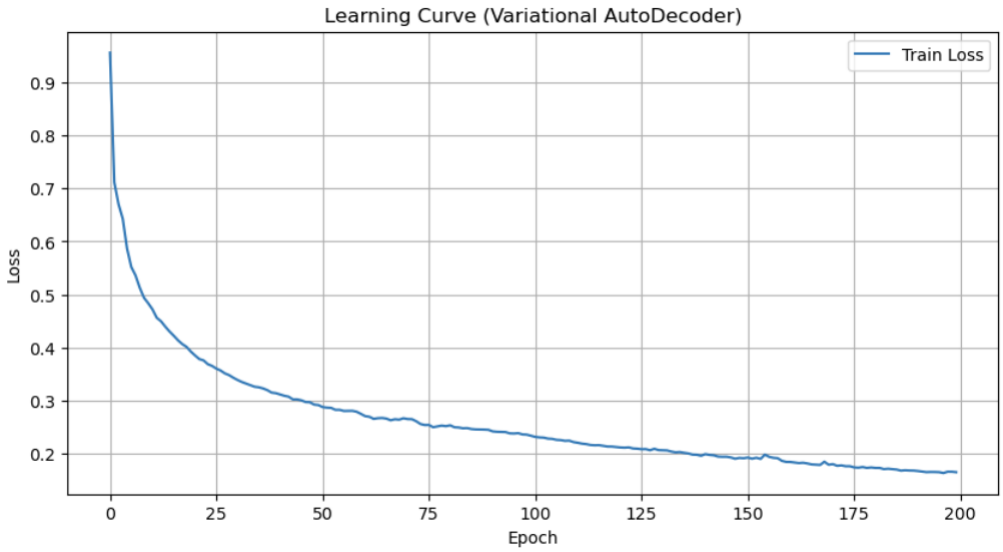


Both TSNEs are quite disorganized and no reasonable clustering of the classes occurs.

3. Evaluation of the VAD model (Gaussian)

Train

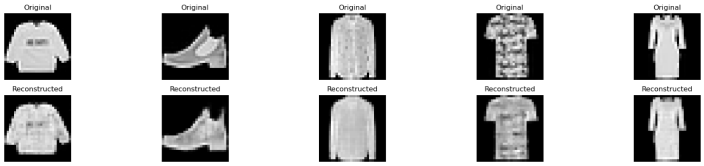
The model training loss graph over 200 epochs:



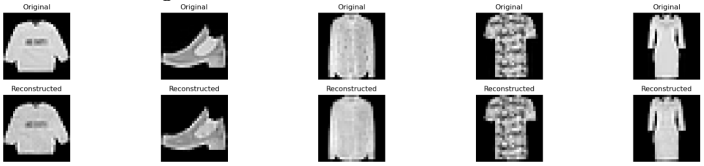
The training loss drops down to around 0.15 from an initial loss value around 1.

Right after the training, the reconstruction loss for the train-set achieved with the evaluation function is 0.116.

Sampling from the distribution learned from the learned distribution parameters for the train-set we get the following reconstructions



Evaluating 100 more epochs optimizing the distribution parameters results with the following reconstructions

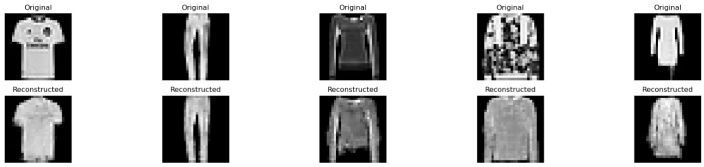


Both results seem quite good and capture the essential features, but the later results seem to capture finer details as expected.

Test

Evaluating the test-set with 100 epochs of optimization only on the distribution parameters we get a reconstruction loss of 0.22.

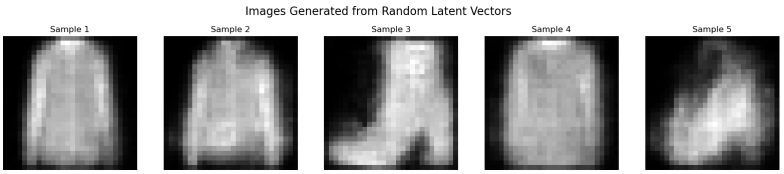
The test reconstruction results:



We can see that the overall shape is reconstructed, but fine details are lost in the reconstruction. Pretty similar to the train-set.

Randomly sampled from standard normal distribution

We sampled 5 latent vectors from $\mathcal{N}(0, I)$, decoded them and reconstructed them. These are the results obtained:

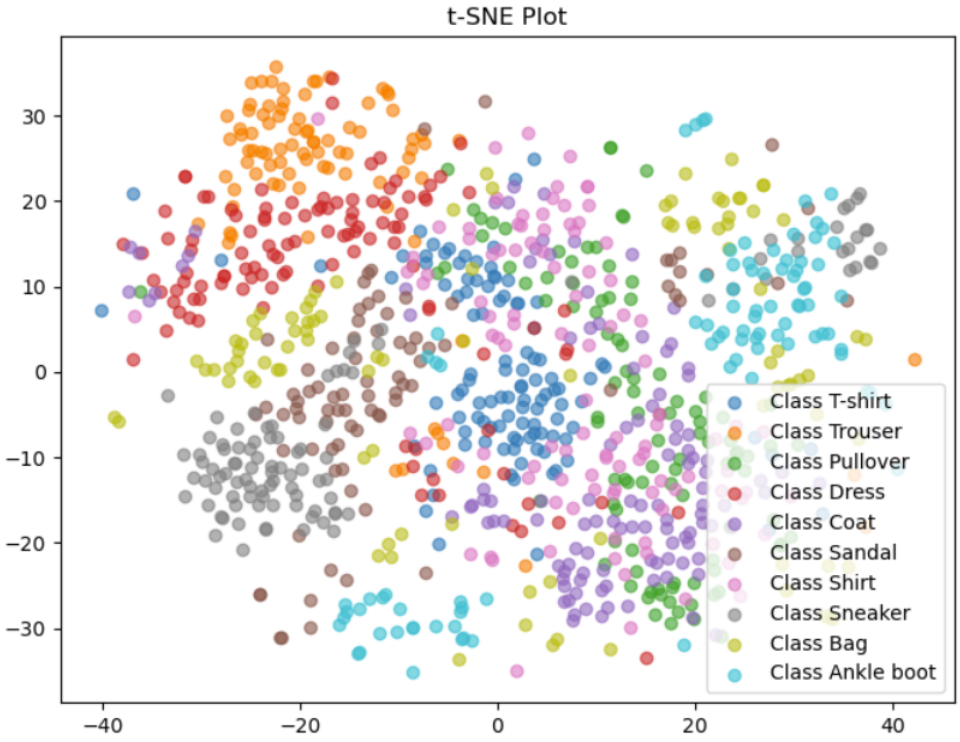


These samples look much closer to Fashion items than the reconstructions from the AD model. But the results are still poorer than the results obtained from trained test-latents, for the same reason mentioned in the previous section - they were optimized to obtain a known shape, while in the randomly sampled latents case did not. We got not too bad results, we can recognize most of the items and give a rough label to them (top clothing item or a shoe). Sample 5 is the most confusing one and is an example for the imperfection of this method to always generate something we can make sense of.

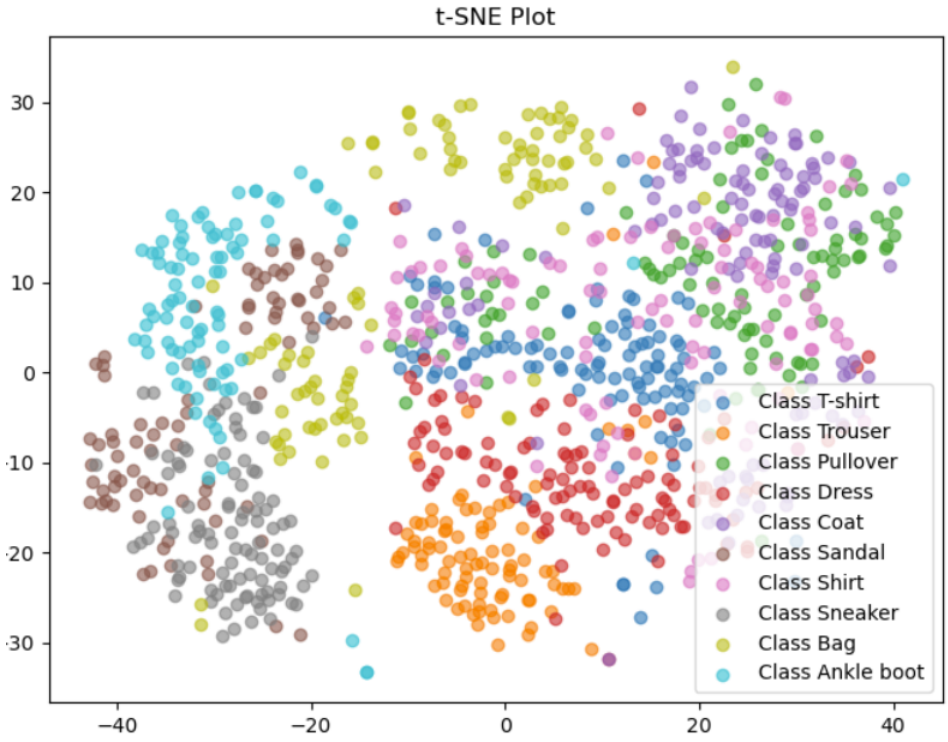
TSNE for Gaussian case (belongs to 1.3.2.5 but for better organization of the report we'll put it here)

TSNE plot for the train-set right after training the VAD (with the latents described by the parameters optimized):

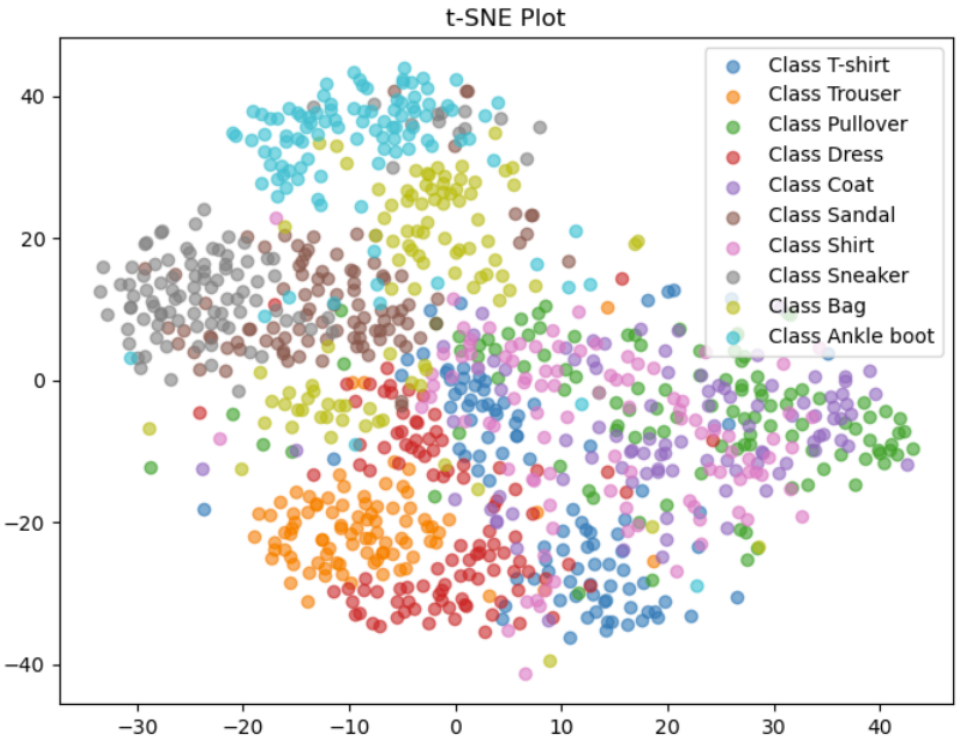
Train TSNE right after model training:



Train TSNE after 100 epochs of optimizing the distribution parameters of the train_set:



Test TSNE after 100 epochs of optimization of the distribution parameters of the test-set:



The TSNE's result makes sense to us, similar looking classes are clustered together, and most of the classes are pretty localized. we can see that footwear are close together and are quite far from the upper body items. the upper body items get mixed together, and the trousers are on their own.

4 Picking different distributions we can use the reparametrization trick on

Initially we wanted to choose distributions which are continuous and simple which a reparametrization trick can work on and there is an easy way to calculate the KL-divergence given two sets of parameters (preferably exists a known closed expression).

After many trials and errors with other distributions, we chose the point-wise (meaning each latent dimension is independent on the others) Exponential ($z \sim Exp(\lambda)$) and Laplace ($z \sim U(a, b)$) to complement the Gaussian.

4a. VAD model (Exponential)

Explanation, reparametrization trick and KL-divergence

We chose the Exponential distribution parameterized by λ: $z \sim Exp(\lambda)$

For the Exponential distribution, our parameter is $k = log(\lambda)$ and get the decay rate by $\lambda = exp(k)$ (and thus is positive for all k and will be easier to use as a parameter).

Sampling from a uniform distribution $\epsilon = Uni(0, I)$ the reparamatrization trick for $z \sim Exp(\lambda)$ is $z = -\frac{1}{\lambda} log(\epsilon)$. this can be seen by inverting the CDF of the distribution $F(x; \lambda) = 1 - e^{-\lambda x}$ to get $F^{-1}(u; \lambda) = -\frac{1}{\lambda} ln(u - 1)$ if we sample $u \sim Uni(0, 1)$ we also have $u - 1 \sim U(0, 1)$ and thus $X = -\frac{1}{\lambda} lnU$ when $U \sim Uni(0, 1)$ and $X \sim Exp(\lambda)$.

The KL divergence between two exponential distribution $P = Exp(\lambda)$ and $Q = Exp(\lambda')$ is a known expression:

$$D_{KL}(P||Q) = log(\frac{\lambda}{\lambda'}) - 1 + \frac{\lambda'}{\lambda}$$

(Derivation:

$$D_{KL}(P||Q) = \int_0^\infty f(x; \lambda) \log(\frac{f(x; \lambda)}{f(x; \lambda')}) dx =$$

for $f(x; \lambda) = \lambda \exp(-\lambda x)$

$$= \int_0^\infty \lambda \exp(-\lambda x)(\log(\frac{\lambda}{\lambda'}) + (\lambda' - \lambda)x) dx =$$

using $\int_0^\infty \lambda \exp(-\lambda x) dx = 1$ and $\int_0^\infty x\lambda \exp(-\lambda x) dx = \frac{1}{\lambda}$

$$= log(\frac{\lambda}{\lambda'}) - 1 + \frac{\lambda'}{\lambda}$$

)

We choose the standardized distribution to optimize toward to be $Exp(\lambda = 1)$ and thus the KL-divergence between $P = Exp(\lambda)$ and the target $Q = Exp(1)$ is given by

$$D_{KL}(P||Q) = log(\lambda) - 1 + \frac{1}{\lambda}$$

4b. Evaluation of the VAD model (Laplace)

Explanation, reparametrization trick and KL-divergence

The Laplace distribution $X \sim Laplace(\mu, b)$ is a distribution with a mean μ, supported by $\mathbb{R}$ and decays from both sides around the mean by a factor b. It is kind of like a combination between the Gaussian distribution and exponential, when the support is all of $\mathbb{R}$ and its symmetric around a mean like a Gaussian, but decays exponentially on both sides of the mean.

The CDF of Laplace distribution is

$$F(x;\mu,b) \;=\; \begin{cases} \frac{1}{2}\exp\left(\frac{x-\mu}{b}\right), & \text{if } x < \mu, \\ 1 - \frac{1}{2}\exp\left(-\frac{x-\mu}{b}\right), & \text{if } x \geq \mu. \end{cases} = \frac{1}{2} + \frac{1}{2} sign(x-\mu)(1 - \exp(-\frac{|x-\mu|}{b}))$$

And the inversion is (taken from Wikipedia)

$F^{-1}(p) = \mu - b \cdot sign(p - 0.5) \cdot ln(1 - 2|p - 0.5|)$

Thus for a $p \sim Uni(0,1)$ we get $X \sim Laplace(\mu, b)$ and this defines our reparametrization trick.

For the KL-divergence between two Laplace distribution , it can be calculated using the definition and solving the integrals. We found this Mathematica code online which shows how one can calculate the KL-divergence between two Laplace distributions using Mathemtica and shows its result.

```
P = PDF[LaplaceDistribution[μ1, λ1]];
Q = PDF[LaplaceDistribution[μ2, λ2]];
sec1 = FullSimplify[Expand[P[x] Log[P[x]/Q[x]]], Assumptions → {x < μ1, μ1 ∈ Reals, μ2 ∈ Reals, λ1 ∈ Reals, λ2 ∈ Reals, λ1 > 0, λ2 > 0, μ1 < μ2}];
sec2 = FullSimplify[Expand[P[x] Log[P[x]/Q[x]]], Assumptions → {x > μ1, x < μ2, μ1 ∈ Reals, μ2 ∈ Reals, λ1 ∈ Reals, λ2 ∈ Reals, λ1 > 0, λ2 > 0, μ1 < μ2}];
sec3 = FullSimplify[Expand[P[x] Log[P[x]/Q[x]]], Assumptions → {x > μ2, μ1 ∈ Reals, μ2 ∈ Reals, λ1 ∈ Reals, λ2 ∈ Reals, λ1 > 0, λ2 > 0, μ1 < μ2}];
FullSimplify[Expand[Integrate[sec1, {x, -∞, μ1}] + Integrate[sec2, {x, μ1, μ2}] + Integrate[sec3, {x, μ2, ∞}]], Assumptions → {μ1 ∈ Reals, μ2 ∈ Reals, λ1 ∈ Reals, λ2 ∈ Reals, λ1 > 0, λ2 > 0, μ1 < μ2}]
```

$$\frac{e^{\frac{-|\mu_1-\mu_2|}{\lambda_1}}\lambda_1 + (-1 - \text{Log}[\lambda_1] + \text{Log}[\lambda_2])\lambda_2 - \mu_1 + \mu_2}{\lambda_2}$$

We take the result as a given and try implementing it to our VAD model, for $\lambda_2 = 1$ and $\mu_2 = 0$ to be our standard Laplace distribution to aim with in the training:

$$KL(P||Q) = \exp(-|\mu|/b)b - 1 - \log(b) + |\mu|$$

(We don't know if it is the correct expression, and don't have time to check as trying to calculate it on our own produced some bugs, but at-least the sanity check of P=Q resulting with KL(P||Q)=0 and also very large $|\mu|$ resulting with diverging quantity)

4Extra - Additional options we considers to evaluate the VAD on, but we couldn't finish in time

4c. Evaluation of the VAD model (Uniform)

We tried playing with a uniform distribution as its reparametrization trick is very simple. But we got trouble with the kl-divergence. A kl-divergence between non-overlapping uniform distributions is infinite, so we had to restrict the parameters to always overlap, this caused numerical issues we couldn't overcome in time.

4d. Evaluation of the VAD model (True Multivariate-Gaussian)

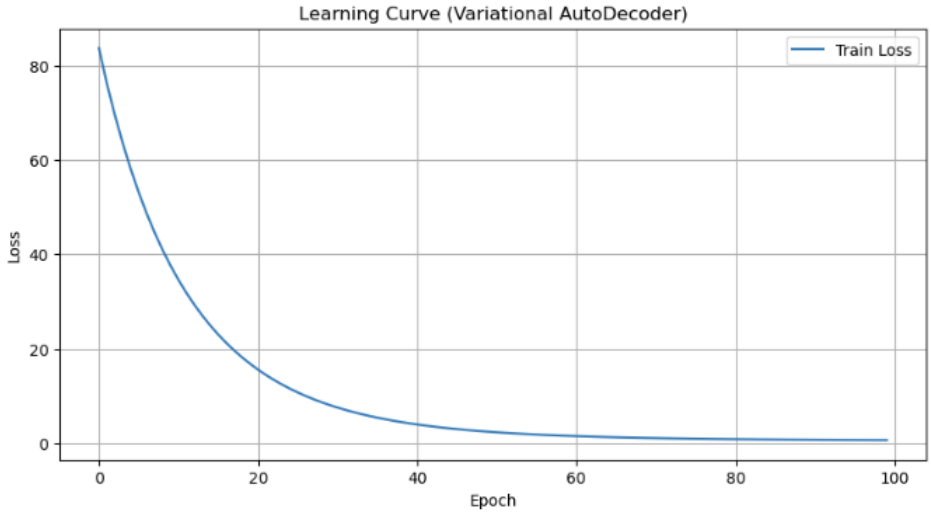
In the Multivariate-Gaussian case we don't assume diagonal Σ matrix like in the "Gaussian" case.

We couldn't find a way to optimize the variance matrix in a way that it would make sense

5. Repeat 1.3.1 and Visualize TSNE for the other models

5a. Exponential distribution results

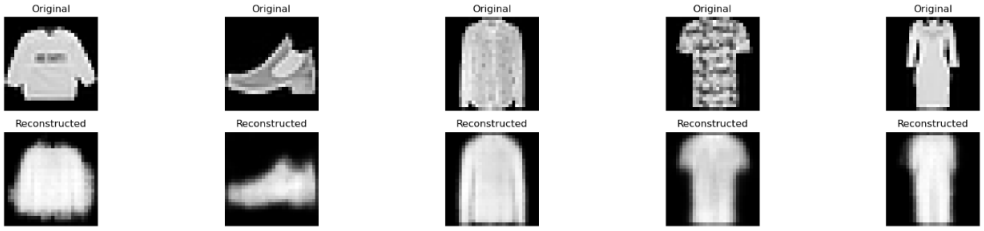
The model's training loss graph over 100 epochs:



The training loss drops down to around 0.6 from an initial loss value around 80.

Right after the training, the reconstruction loss for the train-set achieved with the evaluation function is 0.3.

Sampling from the distribution learned from the learned distribution parameters for the train-set after training we get the following reconstructions, with a reconstruction loss of 0.2619

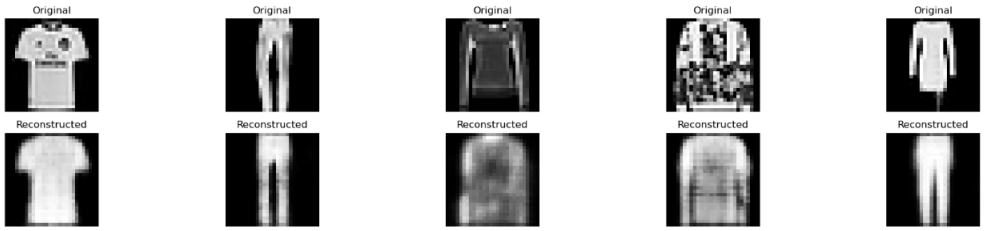


Results seem blurry and not very optimistic considering it is an evaluation for the train-set. But overall shapes of the items is obtained from the reconstruction.

Test

Evaluating the test-set with 100 epochs of optimization only on the distribution parameters we get a reconstruction loss of 0.295.

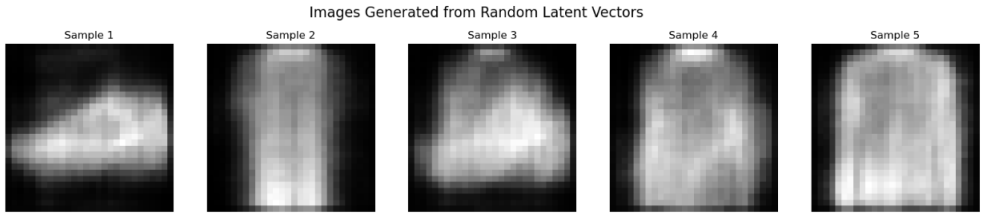
The test reconstruction results:



We can see that the overall shape is reconstructed, but is still quite poor, seems comparable to the train-set which is a sign that there is no over-fitting (at-least not in the model parameters).

Randomly sampled from standard normal distribution

We sampled 5 latent vectors from $Exp(\vec{1})$, decoded them and reconstructed them. These are the results obtained:

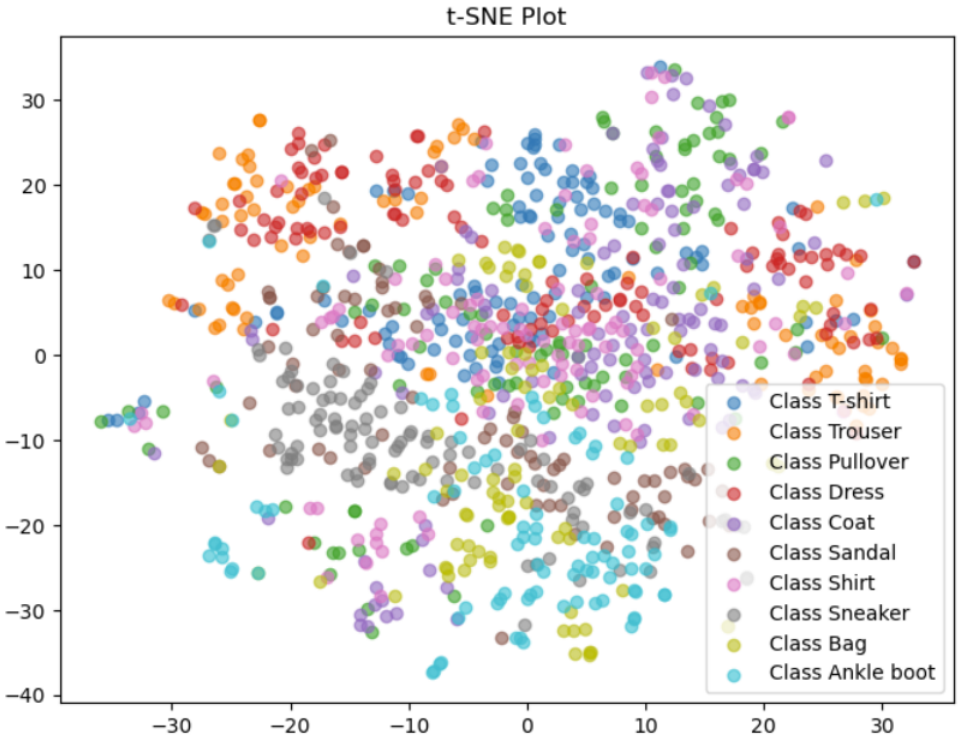


we can see some resemblance to fashion items in the generated images, a shoe to the left and a shirt to the right. but the middle images are not very recognizable.

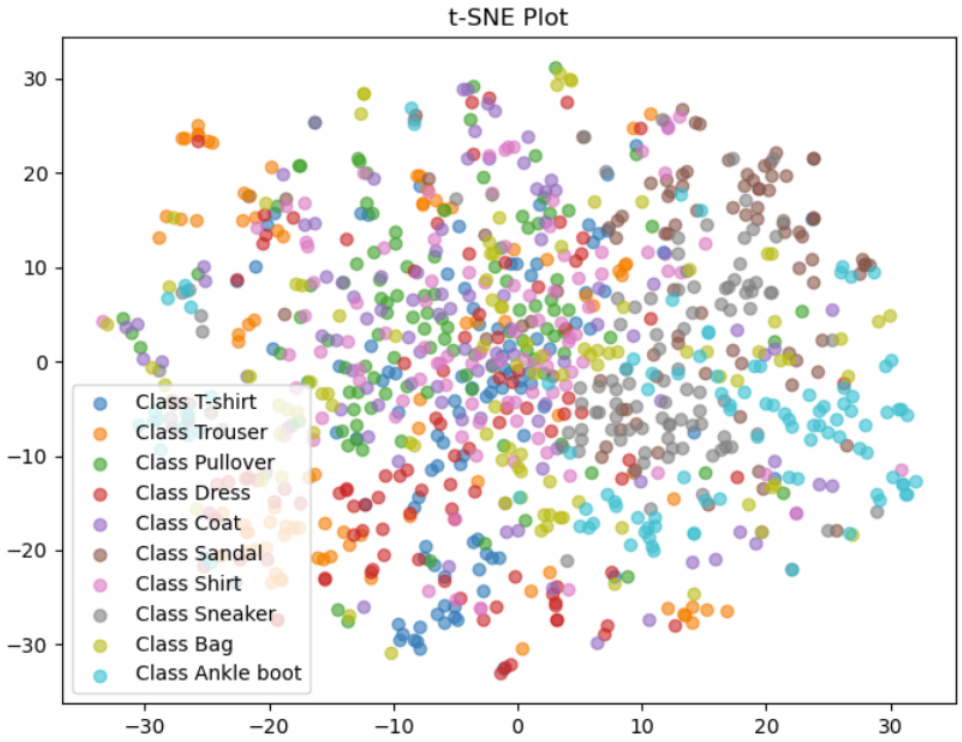
TSNE for exponential case

TSNE plot for the train-set right after training the VAD (with the latents described by the parameters optimized):

Train TSNE after 100 epochs of optimizing the distribution parameters of the train_set:



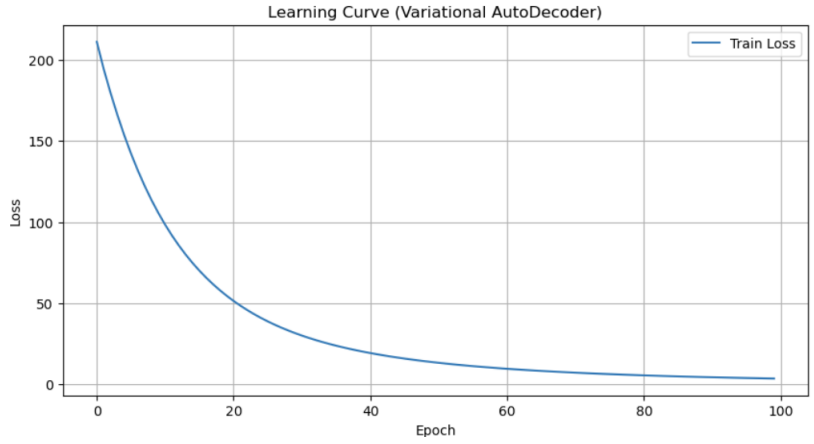
Test TSNE after 100 epochs of optimization of the distribution parameters of the test-set:



Both TSNE results seem poor and do not separate the different classes well. Maybe the exponential distribution is not a good distribution for this separation of classes task given the FashionMnist data-set (or we should have played with the training parameters even more than we already did).

5b. Laplace distribution results

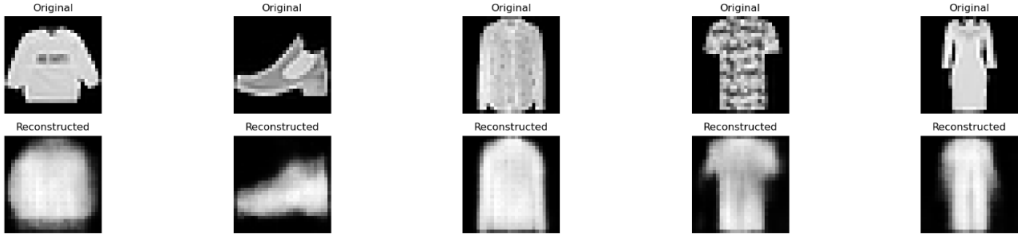
The model's training loss graph over 100 epochs:



The model training loss graph over 100 epochs:

The training loss drops down to around 3.5 from an initial loss value around 210.

Sampling from the distribution learned from the learned distribution parameters for the train-set after training we get the following reconstructions, with a reconstruction loss of 0.2543

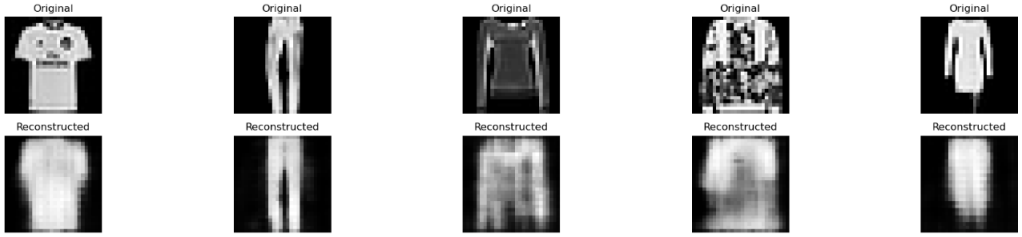


Results seem blurry and not very optimistic considering it is an evaluation for the train-set. But overall shapes of the items is obtained from the reconstruction. Just like the Exponential case.

Test

Evaluating the test-set with 200 epochs of optimization only on the distribution parameters we get a reconstruction loss of 0.2528.

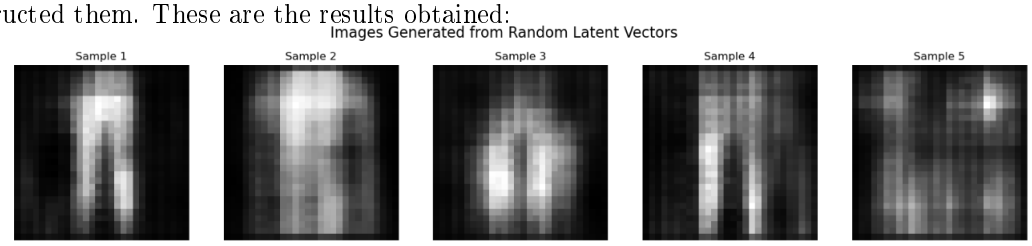
The test reconstruction results:



We can see that the overall shape is reconstructed, but is still very poor, seems comparable to the train-set which is a sign that there is no over-fitting (at-least not in the model parameters).

Randomly sampled from standard normal distribution

We sampled 5 latent vectors from $Laplace(0, 1)$, decoded them and reconstructed them. These are the results obtained:

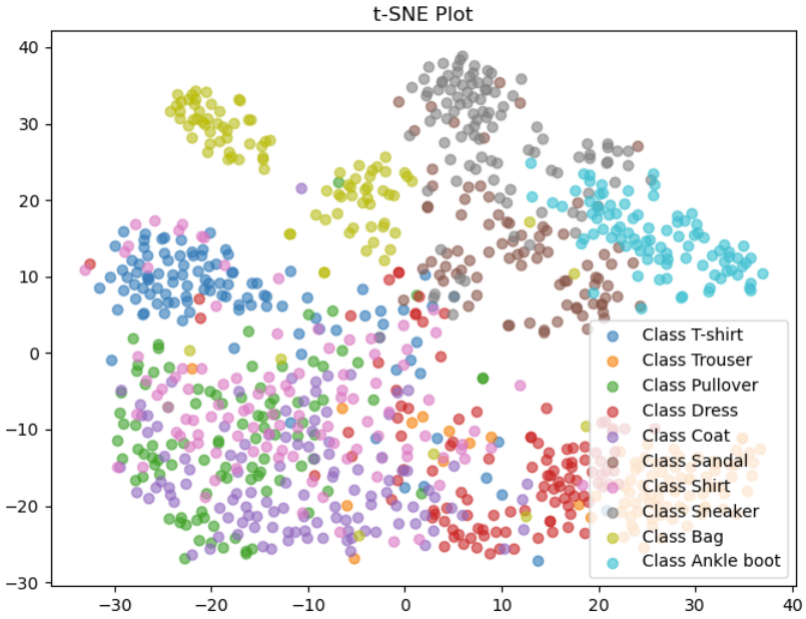


These results don't resemble items from the FashionMNIST data-set. Over many examples looked at we saw some favoring of the reconstructed images to have gap similar to that of a trouser, we can't explain this behavior.

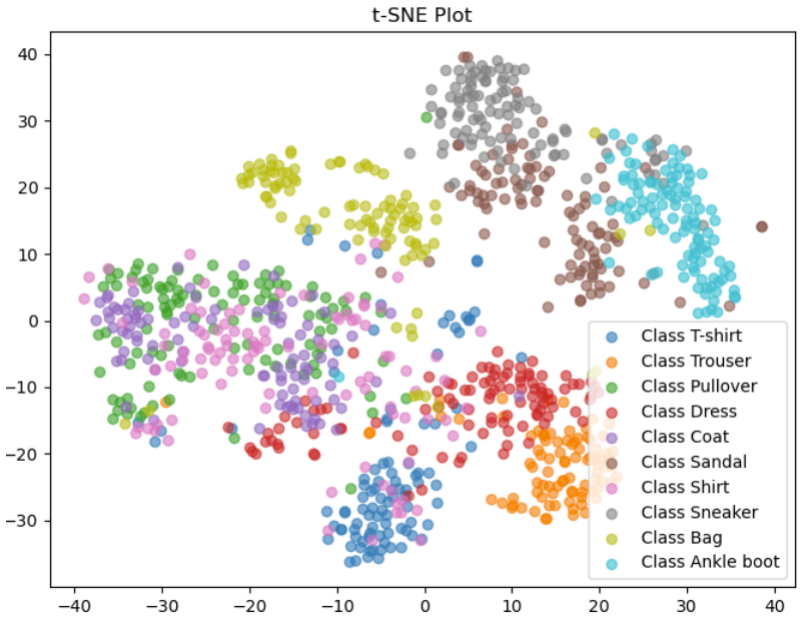
TSNE for Laplace case

TSNE plot for the train-set right after training the VAD (with the latents described by the parameters optimized):

Train TSNE after 200 epochs of optimizing the distribution parameters of the train_set:



Test TSNE after 200 epochs of optimization of the distribution parameters of the test-set:



The results seem pretty good, in a similar manner to the ones from the Gaussian distribution, and even somewhat better. The same classes are clustered pretty much together and similar looking classes are close to each other also.

5c. Summary between the distribution models

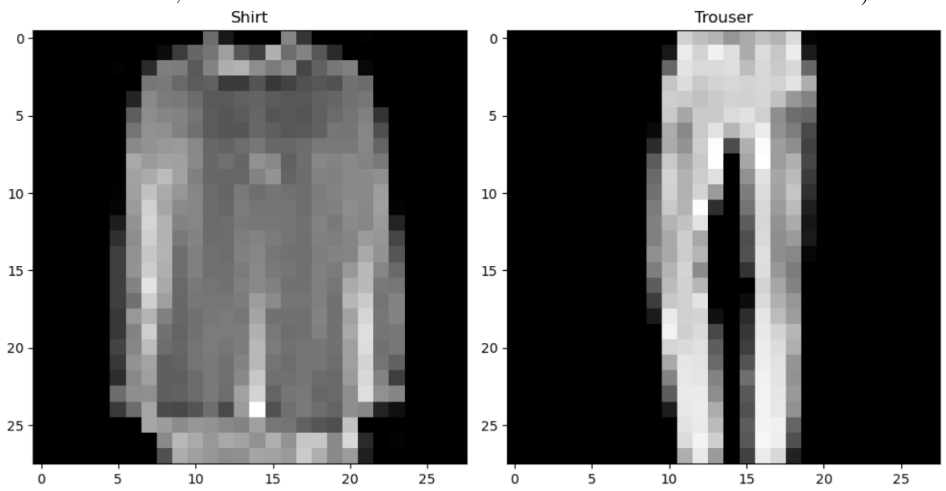
The Gaussian model seem to be pretty good at reconstruction and classification capabilities.

The Exponential seems to be poor but not too bad at reconstruction but very bad at classification.

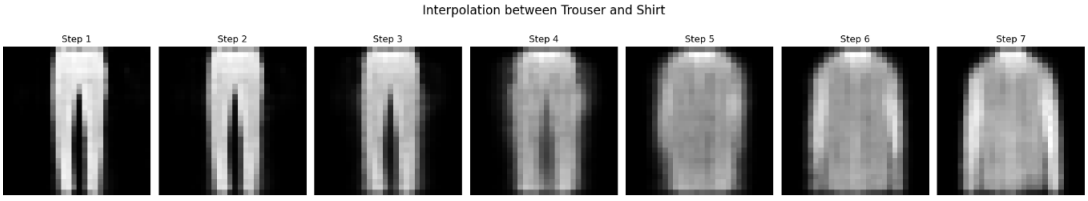
The Laplace distribution seem to classify pretty good and best out of the three, but not good at reconstructing or fitting to the desired distribution of the samples.

6. Interpolations

The interpolations were done between a Trouser (1) and a Shirt (7). We expect to see something similar to a dress between them (as can be somewhat expected from the tsne view, as the bulk of the trousers are somewhere in between them).

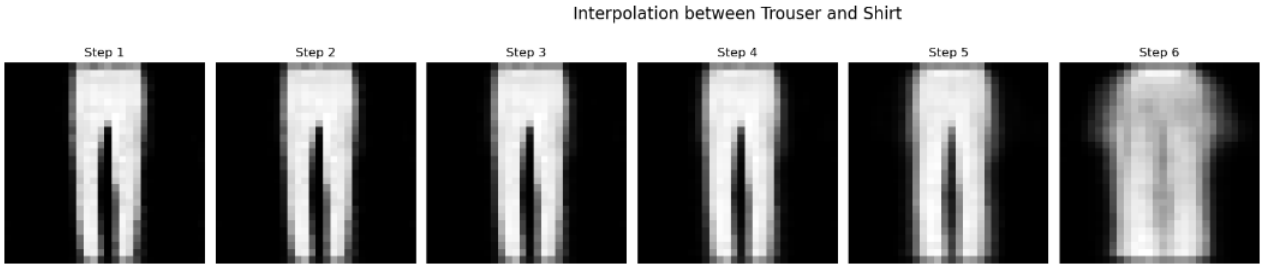


Gaussian case -



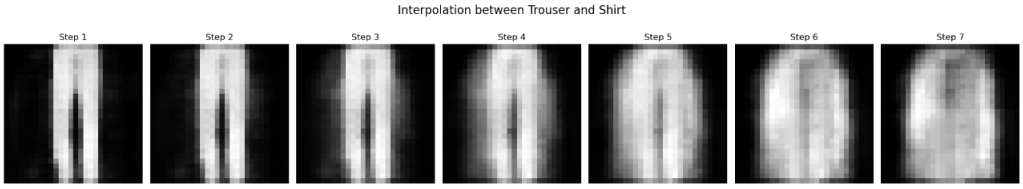
We can "imagine" reconstructions of steps 5 and 6 somewhat resemble a dress as expected.

Exponential case -



Most of the intermediate steps very look similar to the trouser and we can't observe any meaningful new interpolated result, maybe that is because of the nature of the exponential function but we couldn't find any deep reasoning behind it. It is expected though that if the TSNE plots are disorganized the interpolation won't be very meaningful. And again, maybe it is just our fault that the model wasn't trained well enough.

Laplace case -



The interpolation between the shirt and the trouser latents don't seem to result with some new looking item. We expected a dress like we could observe in the Gaussian case and we can't really see it. Maybe its because the reconstruction of the shirt is not very good by itself so its hard to interpolate between it and another point (the trouser looks find). We didn't choose another two choices so the interpolation results would be comparable, but to investigate further we would have used more examples to interpolate between (but we're out of time and need to submit the project).