

Ex3 - Deep Image Priors

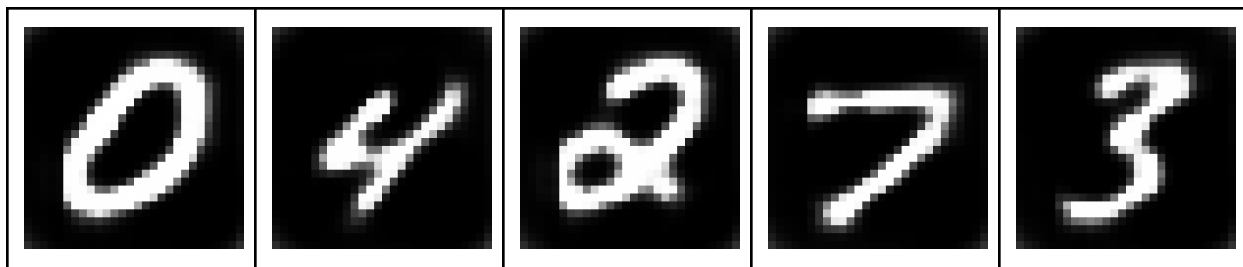
Submission files:

- **Ex3.py** - Has the main code for training the net of **Q1** and **Q4**. To run training for Q4 add the flag '--fit_z' when running from the shell. For the description of the rest of the command line arguments, run with the help flag .
- **Ex3_2.py** - Here we plot the scattering as described in **Q3**. We in addition enabled the possibility of adding out-of-class images to the scattering (see Q4 answer for details). To add the out-of-class images add the flag '--outlier' when running from the shell. Again, for additional cli description run with the help flag.
- **Ex3_5.py** - Has the code for the restoration as described in **Q5**. To run the deblurring add '-t denoising' and to run the inpainting, add '-t inpainting'. For the rest of the options run with the help flag.
- **basemodel.py** - our custom base class for the torch Module
- **models.py** - here we have the AE model
- **common.py** - contains utility code related to pytorch which is used across the different files
- **utils.py** - contains utility code for plotting, printing, saving images, printing feedback etc.
- **NN Images - Ex3.pdf** - Is this file containing the report
- **README**
- **em-M1_dm-M1_e-50_b-48_z-10_lr-0.001_name-urim_auto_encoder_param.pkl** - trained model for Q1. Note that the name is necessary when running ex3_2.py or ex3_5.py (as the parameters are parsed from the name)
- **em-M1_dm-M1_e-100_b-48_z-10_lr-0.0005_name-urim_auto_encoder_param_method-5.pkl** - trained model for Q4. Note that the name is necessary when running ex3_2.py or ex3_5.py (as the parameters are parsed from the name)
- **man.png** - the outlier image of the man

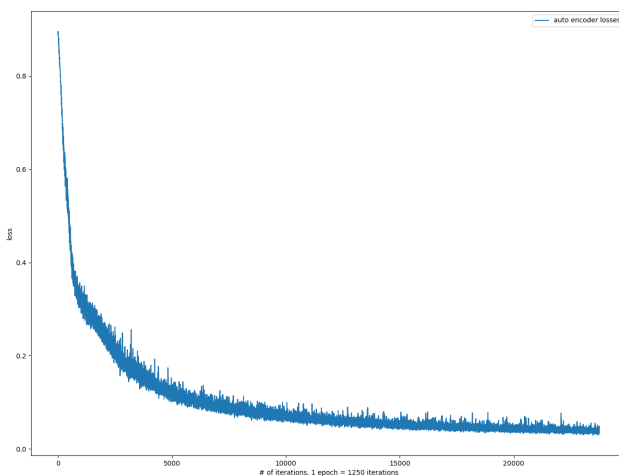
Q1:

Using the architecture from ex2 for the encoder and decoder, we successfully trained the autoencoder to generate digits.

In the following table are results of generated images by the autoencoder when the latent space is big - 100.



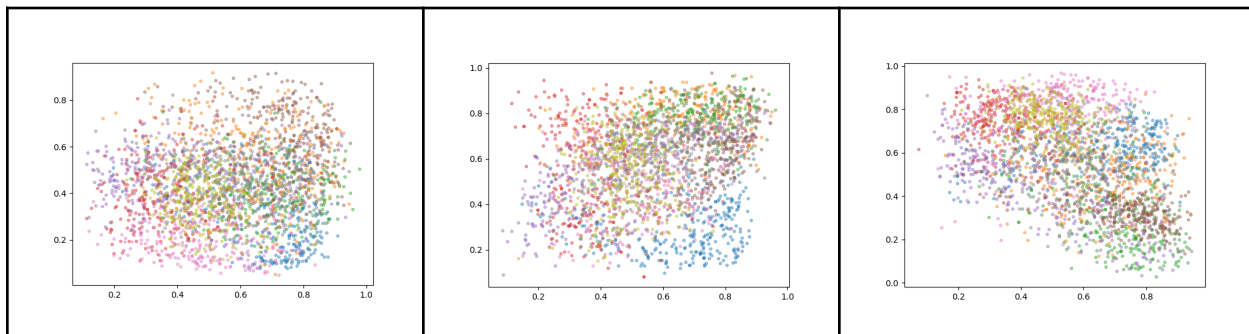
With the corresponding loss function:



Q2:

As required, we retrained the AE with a latent space of 10 (as opposed to 100 in Q1). To understand better the scattering of the latent space, we colored them according to the class of the digit.

The following plots corresponds to the label mapping : {0:blue, 1:orange, 2:green, 3:red, 4:purple, 5:brown, 6:pink, 7:gray, 8:yellow, 9:cian}



As can be seen in the plots above, latent space vectors that belong to the same class are relatively adjacent, but there is an overlap between different classes.

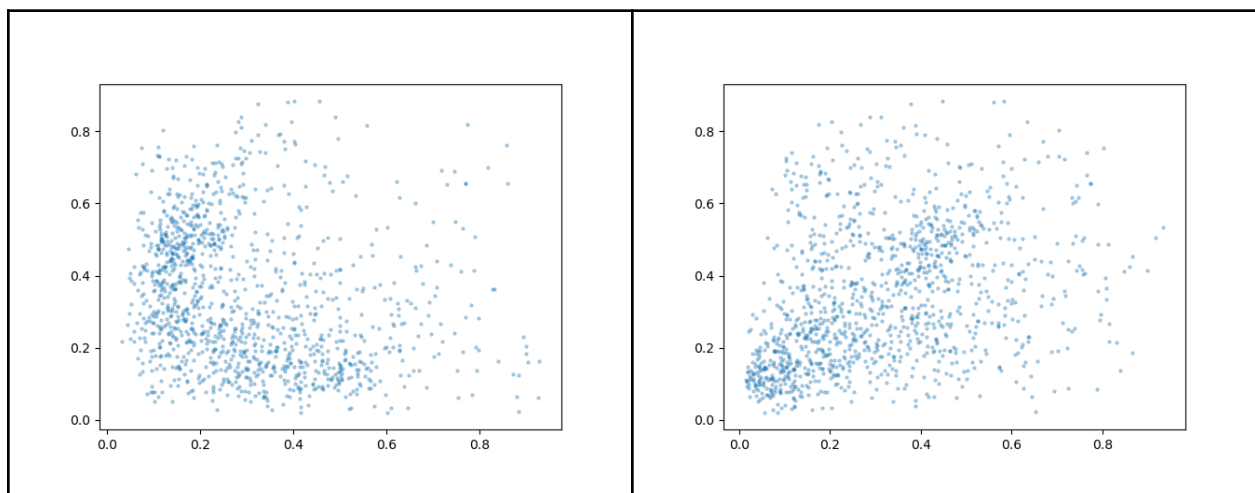
Q3:

The problem with the latent dist is that even though we added a sigmoid at the end of the encoders net restraining it from producing a latent vecot out of $[0, 1]^d$, the scattering plots (see the scattering plots in Q2) show that not all of the $[0, 1]^d$ domain is covered. This will potentially allow out-of-class images not to be projected onto the span of the dataset resulting in the images to be successfully restored.

Q4:

Given the above, our first try was to add a loss to the z latent vector such that it would approximately generate a uniform distribution and so it will cover most of the $[0, 1]^d$ space not leaving much place for outliers. We did this in numerous ways. We tried to use the same approach as in Ex2 where we were asked to fit the z latent vector to be normally distributed by matching its statistics to the statistics of the normal distribution, i.e. match the mean, variance, and kurtosis. Therefore we tried matching the current statistics with that of the uniform distribution which are [mean: 0.5, variance: 1/12, kurtosis: 9/5]. This did not yield the desired results since as it turned out, the scattering was even more centered than before. Later on we argued that a possible reason this did not work is due to the restraining in the mean. We argue that the net at some point faivorse to have a higher loss from the other statistics and group the points more together probably because it will compensate from the original AE loss (reconstruction loss).

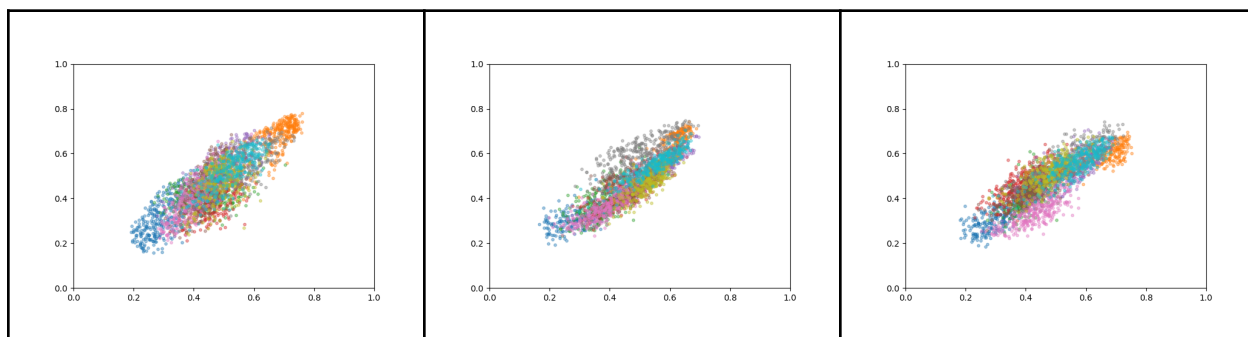
In the following table there are two examples of how this affected the loss. (Note that this scattering was done before we added the feature of coloring different digits with different colors)



Since we did not achieve the desired result, we chose to change the loss function and use the KL Divergence loss which measures the distance between two distributions, where the

measurement is between the latent vector z and a representation of the uniform distribution which is equal to 1 over the latent space size - In our case it is 10, therefore $[1/10, \dots, 1/10]$. This again did not work and we argued that it was because we either might be comparing it to a wrong vector, or more probable, we are trying to make the distribution uniform within the z vector and not across different z vectors a.k.a. the batch. We will go back to this option but first we tried something else.

Since both matching the statistics and using the KL Divergence to fit to the uniform distribution did not work, we tried doing the opposite - maybe our approach is incorrect and in fact if we force the latent space to be more centered in the middle, even an outlier image will be mapped to within the cloud of points in the scattering and therefore we are forcing out-of-class images to be projected into the in-class cloud. For that we used the same mechanism of using the KL Divergence loss but between the *sum* over the batch for each coordinate and the vector $[1/100, \dots, 1/100]$. This gave us the following results:



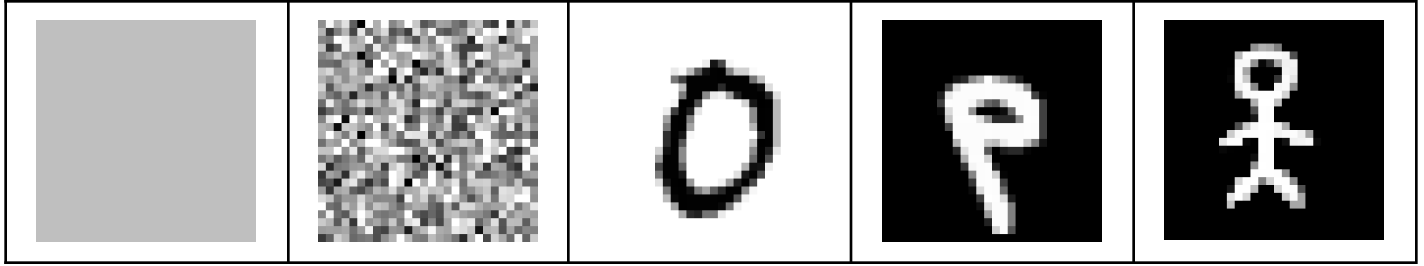
Given the aesthetic beauty of this scattering, this result is quite impressive, nevertheless, we needed a way to verify that we indeed achieved the objective of having out-of-class mapped to where in-class are mapped.

To do so we did the same scattering as before, but we added 5 new images that are out-of-class. The images are:

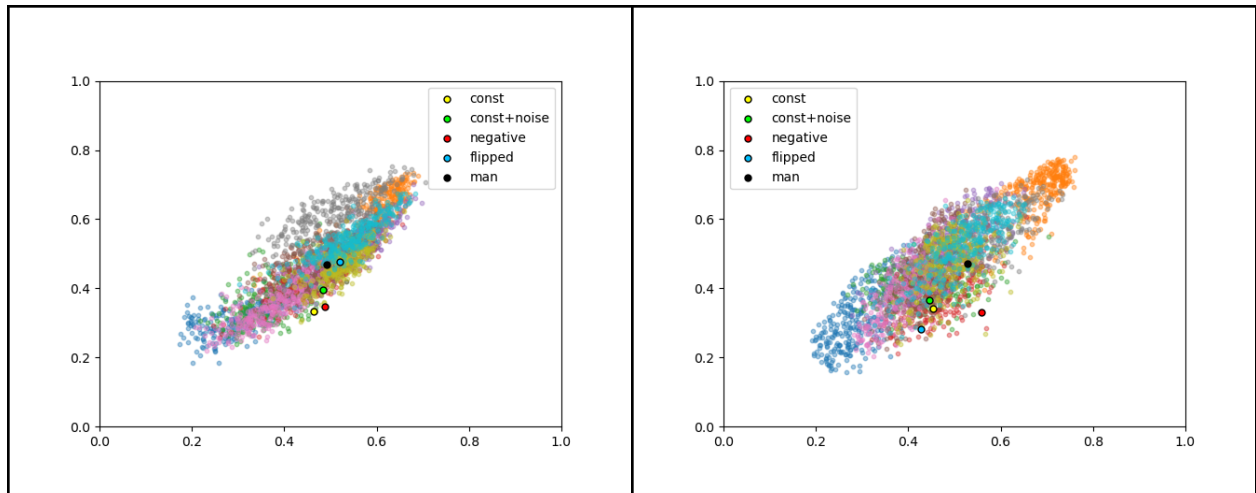
1. A constant image with 0.5 in all pixels. (recall that images are mapped between $[-1, 1]$ making it have the value of 192 in 8 bit representation).
2. A constant image with 0.5 in all pixels plus normal noise with std of 0.5.
3. The negative of a random image from the dataset.
4. A random image from the dataset flipped horizontally.
5. An image of a man

Illustration of the 4 outlier images is in the following table:

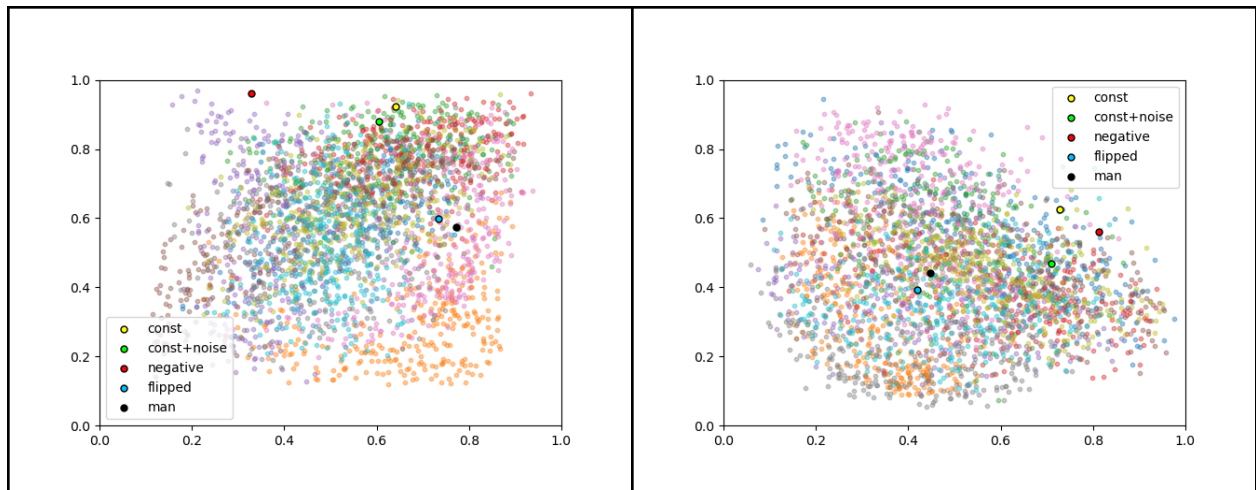
const	const+noise	negative	flipped	man
-------	-------------	----------	---------	-----



We then made new scattering plots testing the trained AE with the mentioned outliers. The results were the following: (The new added points are colored with an additional outer black edge for better contrast)



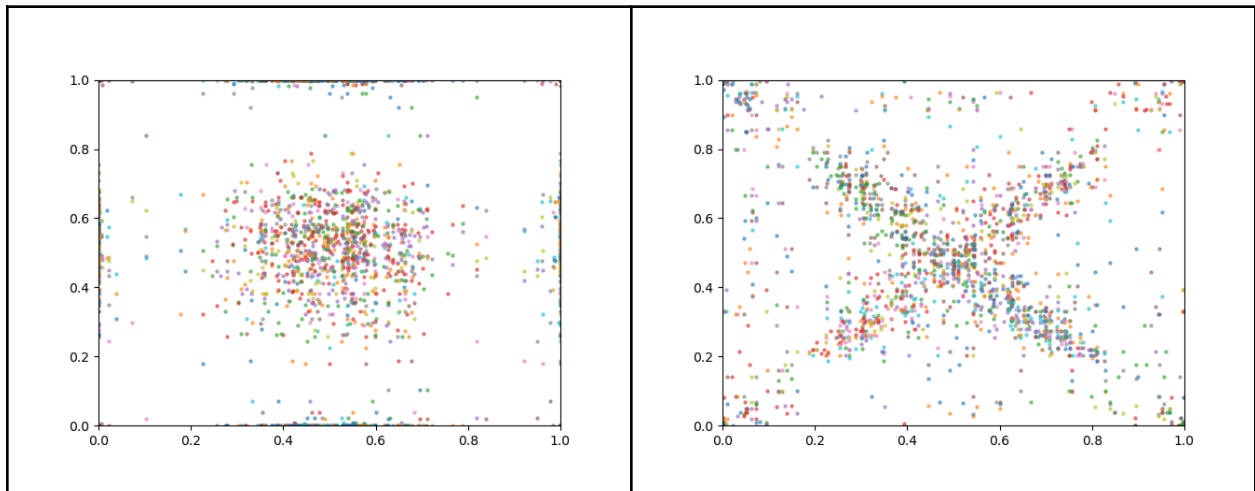
The corresponding scattering for the original net (of Q1 with no additional loss) gives:



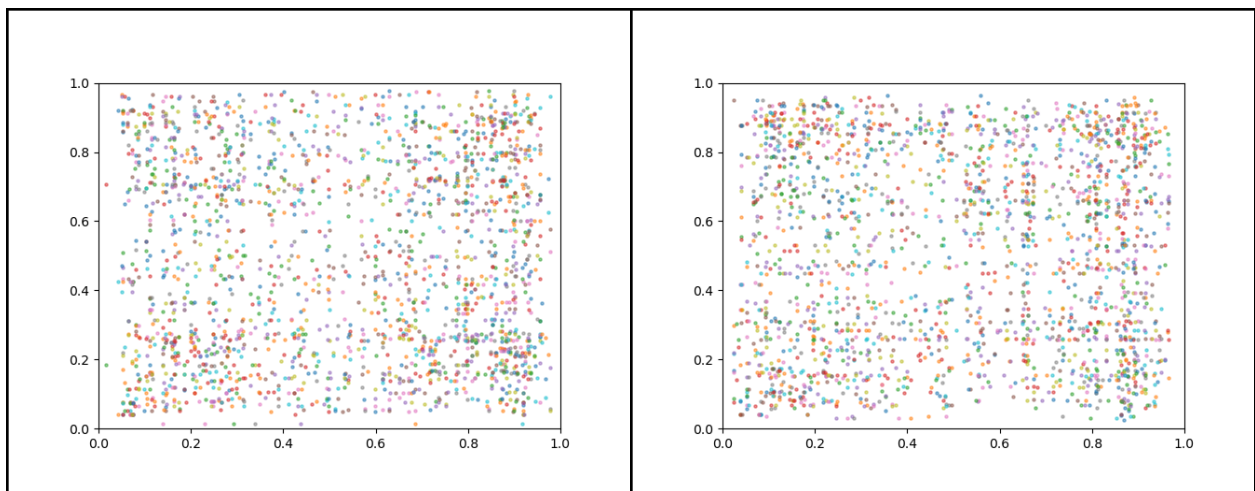
In the last couple of tables it can be seen that the negative and const images are clearly out of class, and the const+noise also tends to be out. In other plots, almost all images are out of the scattering cluster at some point.

Therefore, we decided to go back to the initial approach of trying to cover all the $[0, 1]^d$ space such that we can ensure outliers will be mapped to it.

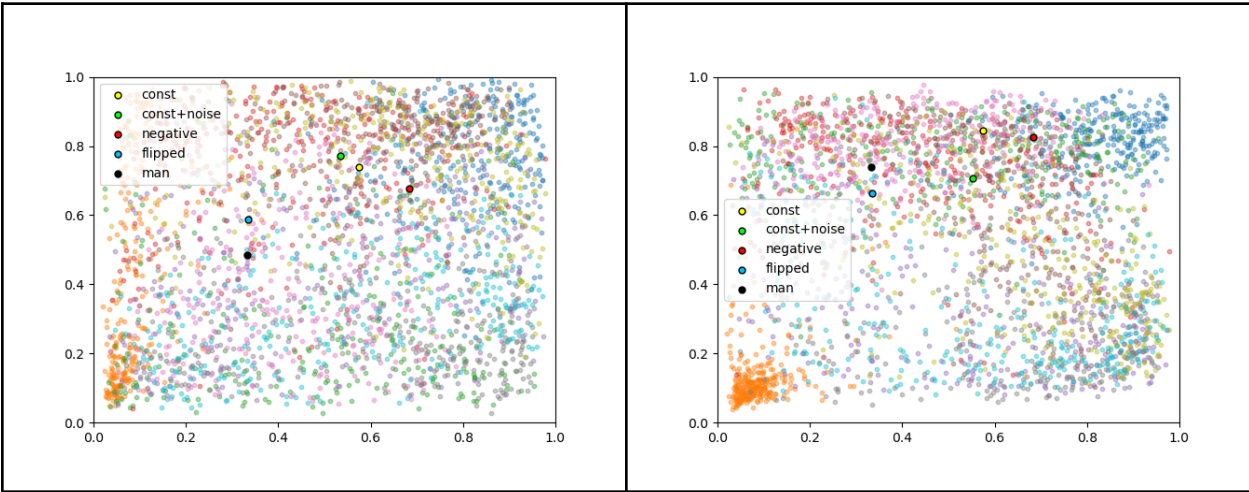
Given the experience we acquired during testing we realized that indeed the effect of fitting the mean makes the scattering eventually collapse to the center. In addition, we noticed that fitting the z vector is not the correct approach because, if for instance, we achieve a uniform distribution across the z vector, it might mean that for some dimension i , across the batch we have only values close to 1, while for another dimension j , we only have values close to -1. In other words, this can converge into very undesirable results. For example:



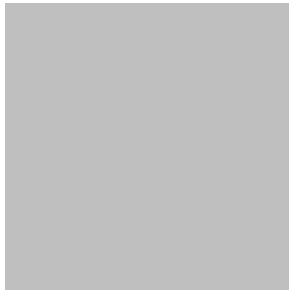
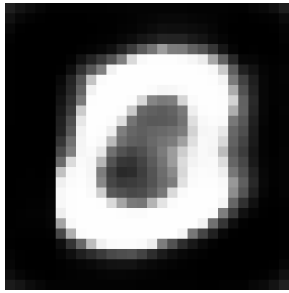
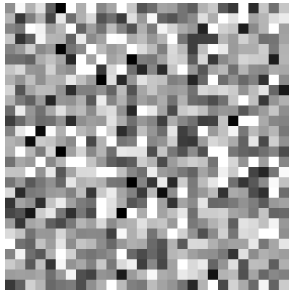
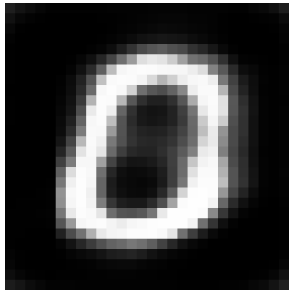
Eventually we decided to fit only the variance and kurtosis moments, but over the batch dimension, while giving the loss a lower weight in comparison to the AE loss (reconstruction loss). After some time we found the best hyper parameters which gave us a relatively uniform covering of the latent space which can be seen in the following table (even after 100 epochs!):

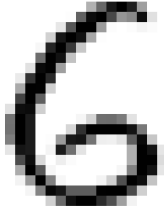
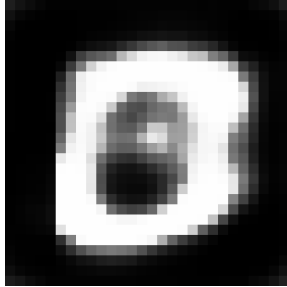
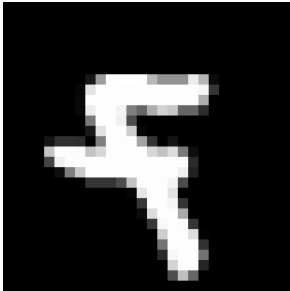
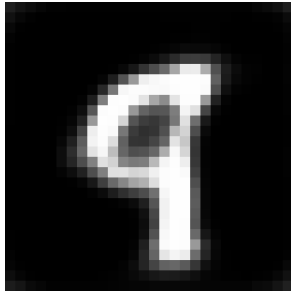
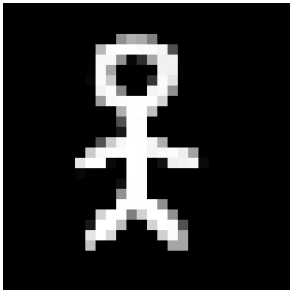
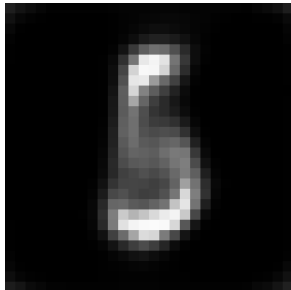


After testing it with the outlier images we got:

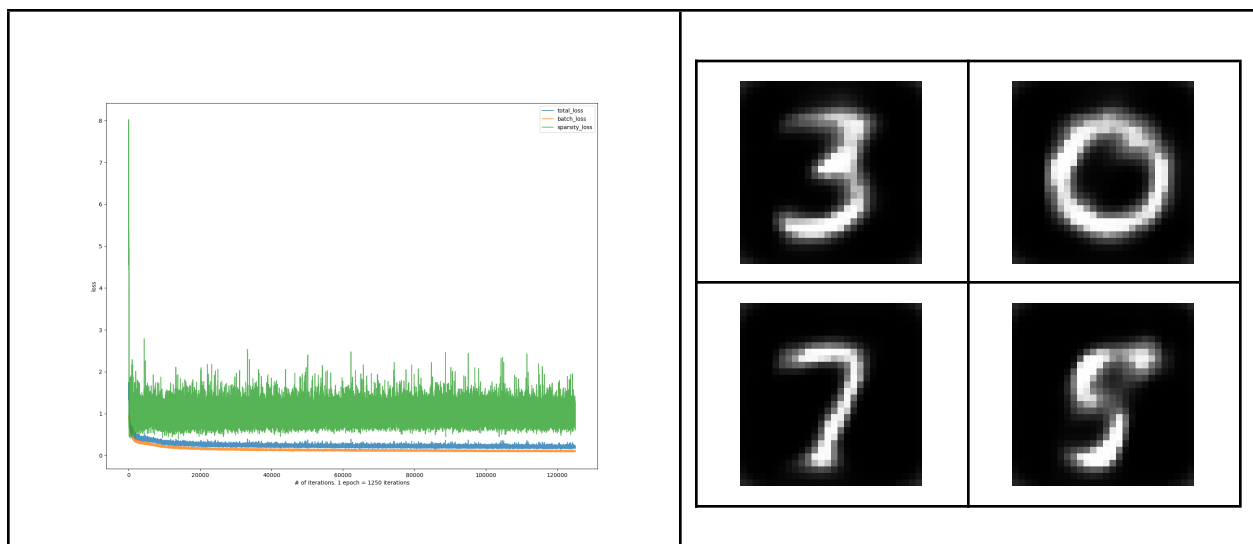


Now, to illustrate that indeed the AE maps an out-of-class image onto in-class we check the output of the outlier images and see what is returned. This can be seen next:

	Input: Image	Output: AE(Image)	Comments
Const (outlier-1)			The const was converted into a zero
Const + Noise (outlier-2)			The const+noise was converted into a zero

Negative (outlier-3)			Even the negative was successfully converted into a 0!!
Flipped-1 (outlier-4)			The flipped 7 was converted into a 9
Man (outlier-5)			The man was converted to something between a 5 or 6

Last but not least, the resulting trained AE with the described additional loss function gave the following loss plot and generated images: (note that the loss for the AE had 20 times more weight and therefore the overall loss is much higher than the sum)



Now, it should be mentioned that the reason the digits created are not so crisp is due to the low latent space degree of 10 but mainly as a result of the additional loss for the fitting.

To sum up, due to sigmoid activation in the last layer of the encoder, the latent space z is restrained to $[0, 1]^d$, and therefore, by having in practice some if not all of that space covered, we ensure that any image will be mapped to the space where the decoder was trained for.

Q5:

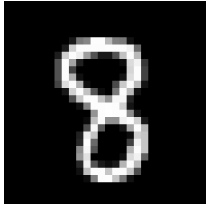
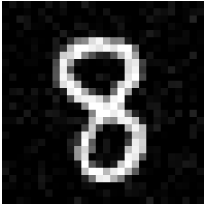
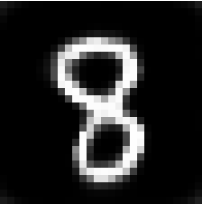
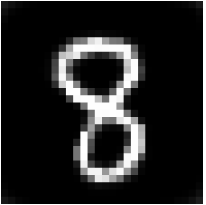
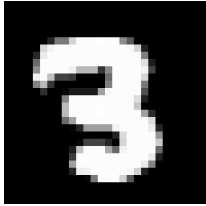
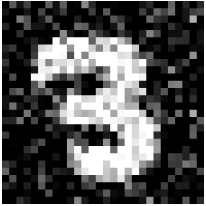


Denoising:

The optimization problem for this section has the following form:

$$\arg \min_{restored} \frac{||restored - corrupt||}{2\sigma^2} + ||AE(restored) - restored||$$

Where the first term is the likelihood term and the second is the prior/regularization term.

We tried the denoising with two different σ 's. As we learned in class, early finish also has a big part in the DIP approach and therefore we also added the comparison with the restored image after 10000 iterations.

	Original	Noisy	1000 iterations	10000 iterations
$\sigma = 0.1$				
$\sigma = 0.5$				

It is clear that in the two cases, over iterations give a less accurate result.


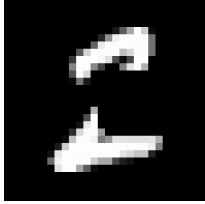


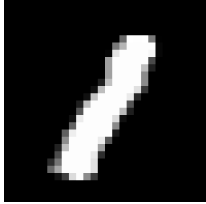
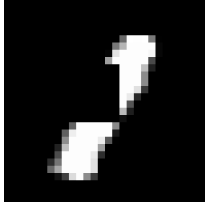
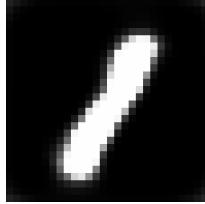
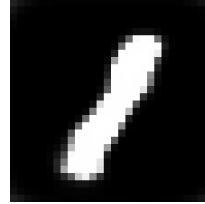
Inpainting:

The optimization problem for this section has the following form:

$$\arg \min_{restored} \frac{||mask(restored - corrupt)||}{2\sigma^2} + ||AE(restored) - restored||$$

Where the first term is the likelihood term and the second is the prior/regularization term. The difference between the denoising and the inpainting is in the MSE loss of the likelihood where we mask both the restored and the corrupt images before the calculation.

Here we also compare the results after 1000 and 10000 iterations.

Original	Corrupted	1000 iterations	10000 iterations
			
			

Again it is relatively clear that in the two cases, over iterations give a less accurate result.

Theoretical Questions

1. The Gram matrix is a matrix of element-wise correlations between 2 feature activations between 2 convolution layers. Meaning, the gram matrix models the correlation between two features. If we think of every feature as a simple pattern (i.e. edge detector) then the gram matrix of two features i and j will contain high values if a high activation of feature i is followed by a high activation of feature j (and vice versa). Since textures are characterized by the same patterns repeatedly all over the image, the features' activation correlation should be similar everywhere in the image and therefore the gram matrix in that case describes behavior of the entire image. In case of images that aren't texture, a correlation between the activations of 2 features might be high at a specific location in the image but low in other locations, and therefore in that case the gram matrix is less informative.

As discussed in class, in the finer layers of the network, the gram matrix describes coarse correlation while as we go deeper, the gram matrix describes smaller features in the image. Therefore, if we use a gram matrix from a fine resolution layer to reconstruct the image, the coarse details will be reconstructed

well but the small details won't. If we choose the gram matrix from a deeper position, the reconstruction will present better the small and gentle details, but the rough details wouldn't be reconstructed well.

2. In AdaIN, each feature activation in the content input is aligned according to the corresponding feature activation in the style input. Since the alignment is applied among all features according to the style image features, the existing correlation between features in the style image are "forced" on the features of the content image during the alignment. Hence the dependencies between different channels are preserved when doing style transfer using AdaIN.
3. (a) GLOW, which fits P to the data by maximizing the log likelihood.
 (b) In principle, VAE and GLOW will improve their accuracy as the number of samples and neurons increases. In the case of GLO on the other hand, increasing the size of the network might cause the z latent space to lose its normal distribution form (as the network is bigger we allow more modifications of z), and therefore decrease the accuracy. In the manner of the number of samples, since the optimization process is independent for every image, it might improve the test error in case there's an overlap between the test and the train set, but otherwise it shouldn't matter.
 (b) All methods that rely on Maximum Likelihood Estimation (MLE) optimization should, in principal, improve as we have a larger dataset, since MSE is consistent (as n goes to infinity) and is asymptotically efficient (converges as 1 over \sqrt{n}). This include: VAE, GLO, GLOW, IMLE. Nevertheless, in GLO it is expected to store the z vectors of every seen image in memory (GPU) and therefore there is a practical limit on the amount of data that can be learned, and in IMLE, due to the complexity of computing the ANN this is not practical for high dimensional spaces (see (d)). On the other hand, Markov-Chain Monte Carlo (MCMC) which is also a generating model (though not NN based), can have very long mixing times making it non suitable for large data.
 Regarding the net size, in general, as long as the data is $O(\text{big } O)$ of some form to the net size, in principle we would expect all methods to improve as they grow. But a possible exception is with GLO where having a larger decoder may affect smoothness of the latent distribution since the number of steps to converge increases and therefore the z vector which is also optimized, even with its lower learning rate, will lose it's initial distribution making it lose its normalizing flow property.
 (c) LeakyReLU since it is invertible as opposed to Relu which is not. (disclaimer: according to Raanan, there is a way to make an invertible translation using Relu, but due to the high risk it is not used). In addition, even though LeakyReLU is not differentiable, we learned that we may consider it as being differentiable at the evaluation point, and therefore we can also compute its determinante.
 (d) reducing the dimension of I using GLO/AE and then using IMLE with ANN.

4. As discussed in class, the Deep Image Prior method takes advantage of the inductive bias of CNN networks to create natural clean images. Meaning, the deep image prior is based on the tendency of CNN networks to generate clean natural images rather than noisy/non-natural images, and therefore if instead we use FC layers we won't benefit from the inductive bias. Therefore, we don't expect this approach to provide regularization of any form.