Cognition and Computation: noise robustness with self-supervised learning

Sinigaglia Alberto
alberto.sinigaglia@studenti.unipd.it

1. Introduction

1.1. Goals

The project aims to compare the efficency of Deep Belief Networks and Self-Supervised Autoencoder, as support to readout models when dealing with noisy data.
Specifically, this data will be the MNIST Digits dataset, and the noise will be an autogenerated perturbation over those images.

1.2. Reasons

Supervised, unsupervised and self supervised has their pros and cons, which might be summarized as:

- Supervised has the pro to be "direct" and almost effortless usually, but labeling data it's expensive
- Unsupervised has the pro to be cheap, but the best we can do is clustering and learning latent factors

Nonetheless, Self-Supervised takes the pros of both the techniques, where it can be applied, since:

- it has a target value, therefore it can learn specific task
- it does not need the data to be labeled

As cons, it does not have too many applications, since to train such models, the target is either the data itself, or some transformation of it.

However, for the goal of this project, autoencoders can be applied, since it funges as denoiser of the images.

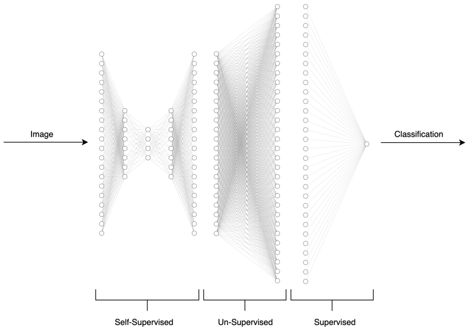
Especially this tool has been applied in 2 specific ways:

1. Self-supervised → Supervised
2. Self-supervised → Unsupervised → Supervised

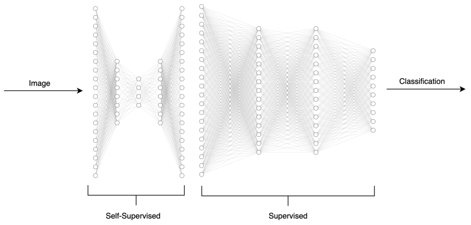
It's been used in this way because, if the self supervised part is able to reduce the noise, and the unsupervised is able to extract latent features form the reconstructed images, the final supervised readout model should be almost immune to noises in the images (when dealing with reasonable amount), and so, as already demonstrated, be almost as powerful as a FeedForward Neural Network.

Following are the structure of the models that has been tested:

1. Autoencoder + DBN + Perceptron



2. Autoencoder + Neural Network



1.3. Properties

The aim of this project is to find a model that is both resilient to noisy images, but at the same time, not too complex.

In particular, the focus was on the ability of the model to be resilient to different levels of noise, even in exchange of accuracy on not noisy images

1.4. Results Overview

It will be shown that the self-supervised autoencoder helps the neural network (and even a simple perceptron) to be immune to noise, way more than the DBN can do.

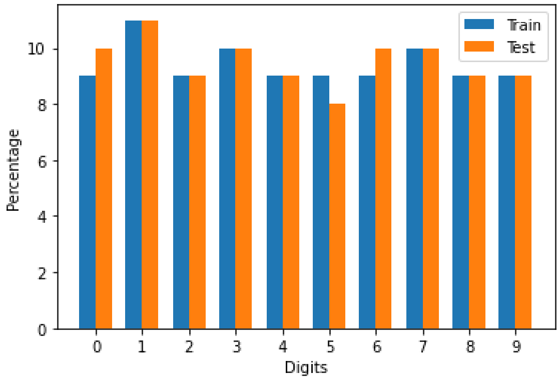
2. DataSet

2.1. Overview

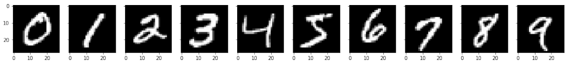
To have more realistic and accurate results, the dataset was splitted in 2 parts:

1. Test set (first 10000 images)
2. Training set (remaining images)

Following are the distributions of the digits in the 2 sets:



Some examples of digits in the dataset:



2.2. Preprocessing

The images are represented by 28×28 values between $[0, 255]$.
Each value was then scaled down to $[0, 1]$ since DBN works with probabilities.

3. Models

3.1. Self-Supervised Autoencoder

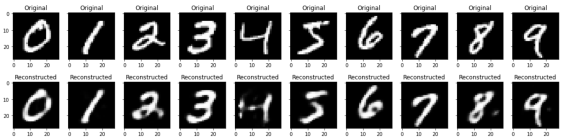
The autoencoder is a self-supervised model that aims to reduce the complexity/dimensionality of the problem.
This might be complicated with supervised learning, because it would require the desired outputs in the lower dimensions.
To avoid this problem, the autoencoder is composed by 2 parts, encoder and decoder. The first part is responsible of mapping the input to a lower dimensional space, and the second part is responsible of, starting with the lower dimensional representation, rebuilding the desired image, and so the targets of our training set is the training set itself.

3.1.1 Simple Dense Encoder

The first attemp we tried was using a simple/basic autoencoder, composed by:

1. Input layer (784 neurons)
2. Encoder layer (32 neurons)
3. Decoder layer (784 neurons)

This model has been trained with as input and target the original images, with the intention of seeing if the idea of an autoencoder trained on images was actually appliable.
After the training, those are some images reconstructed for each class:



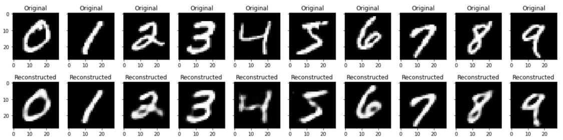
The autoencoder was reconstructing the images in a "clear way", however the loss was still evident, so probably 1 layer was not powerful enough

3.1.2 Complex Dense Encoder

This time instead of one layer per sub-component, we used 2, which helped to "gradually" come to the lower dimension, and then back to the initial image:

1. Input layer (784 neurons)
2. Dense layer (128 neurons)
3. Dense encoded layer (32 neurons)
4. Dense layer (128 neurons)
5. Dense decoded layer (784 neurons)

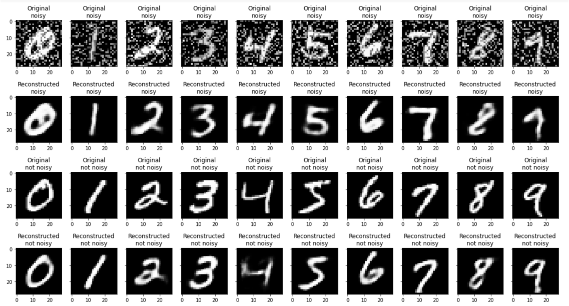
These are some of the reconstructions of this autoencoder after the training:



The difference is noticeable, mostly on the number 4, where before it was almost an H, now it resembles a proper 4.

Yet, the aim of this "component" is not reconstructing images, but removing noise, therefore we have then proceeded to train the autoencoder on the noised data, to see if it was able to learn how to remove noise. To do so, we fed the network with the perturbated images, and as taget, we had the non perturbated ones.

Following are the results:



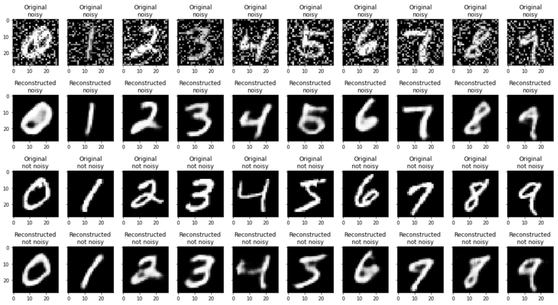
In the first 2 rows, there are the reconstructions from noised images, and in the second 2 rows there are the reconstructions from images without noise.
This clearly evinces that the model is learning what it's noise and what is not, and that is not just "removing random white elements".

3.1.3 Convolutional Encoder

Since the dataset is composed by images, we also tried using convolutional layers to see if it helps:

1. Input layer (784 neurons)
2. (2D Convolutional + MaxPooling) x 3 layers
3. (2D Convolutional + UpSampling) x 3 layers

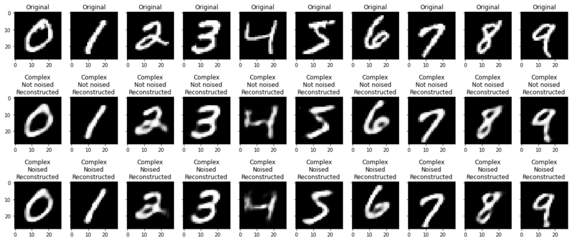
Following are the recostructions:



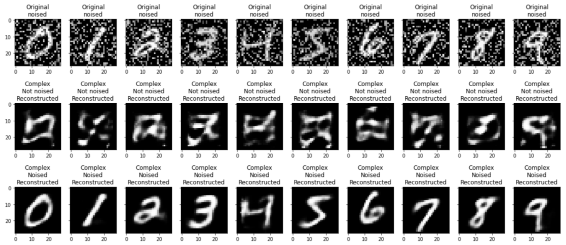
There is no much difference in the reconstructions, except for the digit 4, but the training took almost 3 times longer.

3.1.4 Comparison of Autoencoders

We then proceeded to compare the different autoencoders, primarly focusing on the "complex model", since we have 2 copies of that autoencoder, one trained on original data, and the other one on the noisy one.
The below image represents the reconstuctions of those 2 models, starting with not noisy images:



There is no much difference between the two reconstructions, even though they are being trained with different data. However this small difference doesn't hold in the other direction, meaning that if we test those two models on noisy data, it makes a big difference:



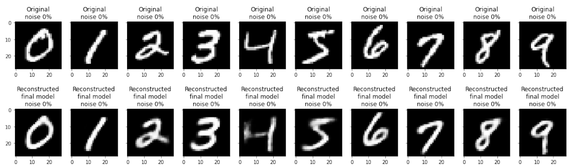
The model trained on the original data, has no notion of noise, and so as soon as it sees noisy data, it does not know how to manage it.

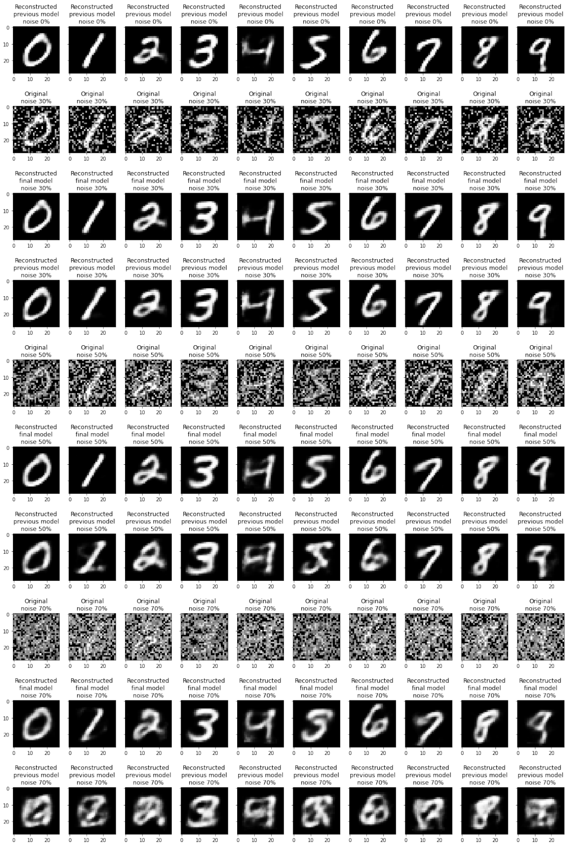
3.1.5 Final model

Ultimately, considering all the pros and cons of the previous models, we procceded to train a final version, with the following structure

1. Input layer (784 neurons)
2. Dense layer (128 neurons)
3. Dense layer (64 neurons)
4. Dense layer (32 neurons)
5. Dense layer (64 neurons)
6. Dense layer (128 neurons)
7. Dense layer (784 neurons)

In addition to the increasement of the complexity of the model, unlike the previous ones that were being trained on images noised by 30%, this model has been trained on data noised at different levels, from 0% up to 70%.
Following are the reconstructions at each noise level from this model, compared to the "complex" one previously presented:




Reconstructed previous model noise 0%
Original noise 30%
Reconstructed final model noise 30%
Reconstructed previous model noise 30%
Original noise 50%
Reconstructed final model noise 50%
Reconstructed previous model noise 50%
Original noise 70%
Reconstructed final model noise 70%
Reconstructed previous model noise 70%
0
10
20

It's evident the ability of this final autoencoder to handle noise also at different levels, where the previous one, already at level 50% was losing a lot of informations

3.1.6 Final evaluation of the autoencoders

To compare the models more formally, we used the following formula to calculate the accuracy of each model:

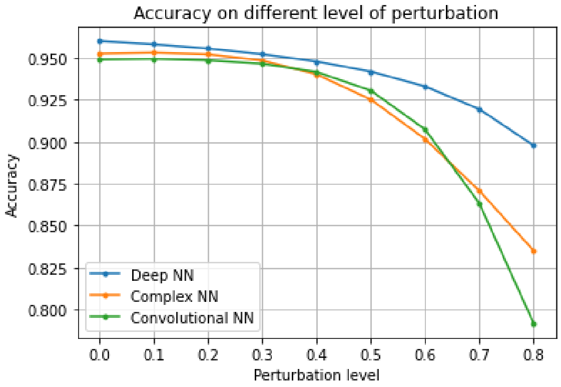
$$SingleAcc(prediction, target) = \\ 1 - \frac{\sum_{i=0}^{|prediction|} |prediction_i - target_i|}{\sum_{i=0}^{|target|} max(1 - target_i\ ,\ target_i)}$$

$$FinalAcc(prediction, target) = \\ \frac{\sum_{i=0}^{|predictions|} SingleAcc(predictions^{(i)}, targets^{(i)})}{|predictions|}$$

Where:

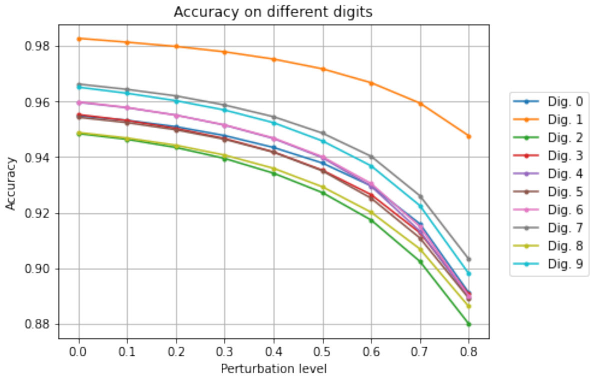
- $SingleAcc$ takes a single prediction vector and a single target vector, and calculates the distance between the prediction and the target, and divides it by the highest error possible, which is the max between $1 - target_i$ and $target_i$
- $FinalAcc$ takes a vector of predictions and targets, and calculates the average error

By using these formulas, following are the evaluations of the models (only the one trained with noisy images) at different perturbation levels:



Clearly the final Deep Neural Network trained on more data with different levels of noises, outperforms the other 2 models.

Considering only this final model, it's interesting also to take a look at the accuracy on each digit:



It's evident how some digits are more easily reconstructible, but what matters is that the model can asymptotically reconstruct each digit with the same ability.

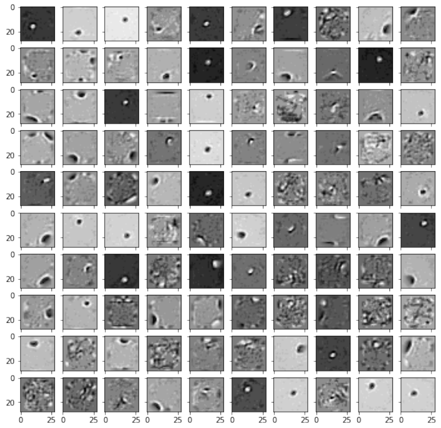
3.2. Unsupervised DBN

To handle the unsupervised part, we used a DBN which takes in input an image (either reconstructed by the autoencoder or not) and should try to find latent features.

This model is composed by:

- One input layer (784 units)
- One hidden/output layer (500 units)

It had been trained for 500 epochs to ensure the convergence of the units, infact, by plotting some of its hidden units, those are the receptive fields:



3.3. Supervised Perceptron and Neuran Network

To make the competition between models fair, we used a simple perceptron (trained in one-vs-all) for the DBN, and Deep Dense NN for the AutoEncoder, composed by:

1. Input layer (784 neurons)
2. Dense layer (500 neurons)
3. Dense layer (500 neurons)
4. Dense layer (10 neurons)

3.4. Final models

In the previous sections, the following componets are being presented:

1. Deep Autoencoder trained on noisy data
2. 2-layers DBN trained on original data
3. Perceptron trained one-vs-all
4. Deep NN trained on original images

With those, we proceeded to build the following 3 models:

1. DBN+Perceptron: used as benchmark, since already widely explored
2. AE+DBN+Perceptron: created to see if the DBN holds its ability to find latent features also from denoised images
3. AE+NN: created to see if the previous DBN+Perceptron still holds its ability to be comparable to a complex deep neural network
4. AE+Perceptron: created to compare with the second model, to see how much difference the DBN makes in transforming the images linearly separable (not part of the actual project)

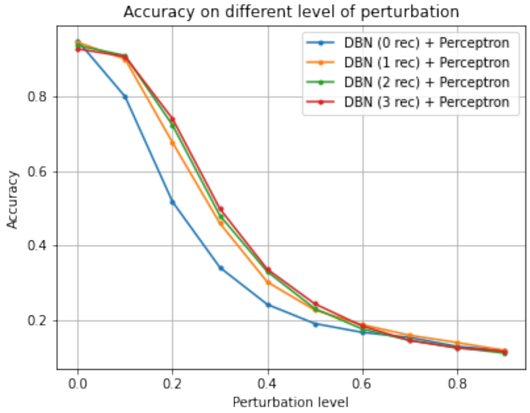
4. Models Evaluation

To evaluate more extensively the models, we created data at different levels of noise, and fed it to the models previously described, and stored the relative accuracies of the predictions.
In addition to the models themselves, it's also been tested each single model with different number of reconstructions, for example for the DBN, doing forward and back propagation of the image, and for the AE by feeding to itself its own output.

4.1. Repetitions on the DBN

4.1.1 DBN+Perceptron

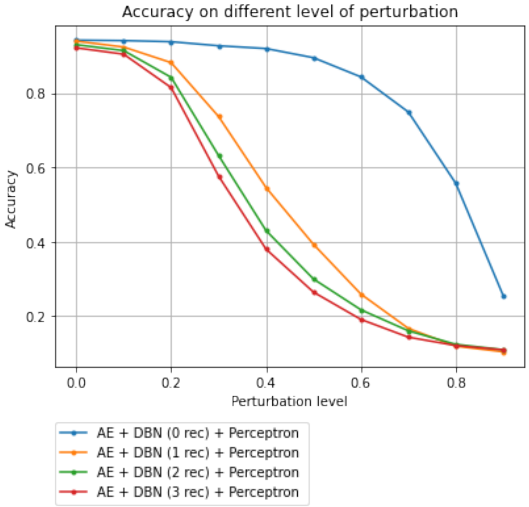
Following is a graph representing the accuracy in classification for the model DBN+Perceptron



Undeniably, after 1 reconstruction we got the highest improvement, but we can also see that the more reconstructions we do, the better it gets, even though the improvement gets closer to zero as the number of reconstructions goes to infinity.

4.1.2 AE+DBN+Perceptron

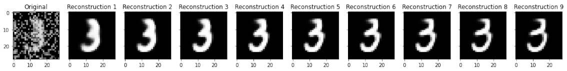
We then proceeded to check the combination AE+DBN+Perc, since also in this model there is the possibility to try different number of reconstructions for the DBN:



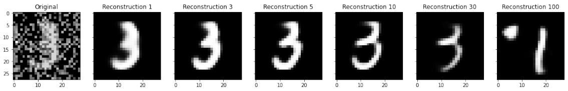
In this case the reconstructions do not help at all the perceptron to improve, and makes the classification therefore worse.

4.2. Repetitions on the AE

Since this has never been tested, it's better to see how the autoencoder behaves with different number of repetitions. To do so, the following image shows a noisy image reconstructed different number of times:

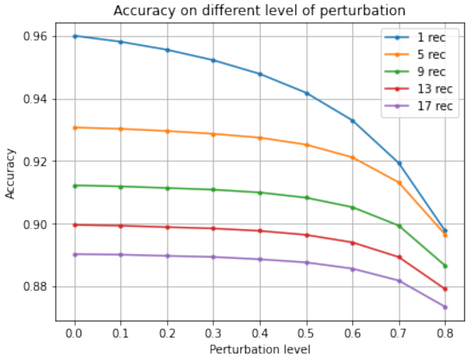


There is a clear convergence to a more (humanly) distinguishable number the more reconstruction we do. Taking in consideration this, since the reconstruction it's still significantly different from one step to another, we tried with an exponential number of steps:



There are clear evidences that the process does not converge to a nice sharp image, and that at some point, the autoencoder removes informations that should not be removed.

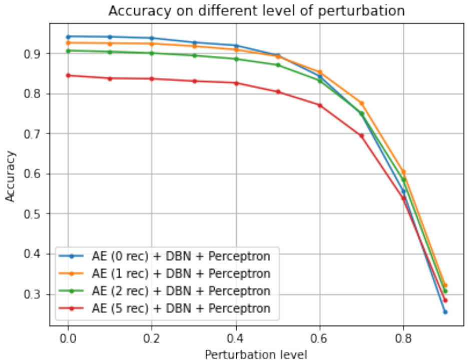
To avoid having to choose a wrong number of iterations, we decided to find it empirically.
Following is the error in the reconstruction:



Clearly, having more reconstructions, does not help in reconstructing the original image. On the other hand, this is not the aim of this component; infact, it's supposed to transform a noisy image, to the best "sharp" digit it sees.

4.2.1 AE+DBN+Perc

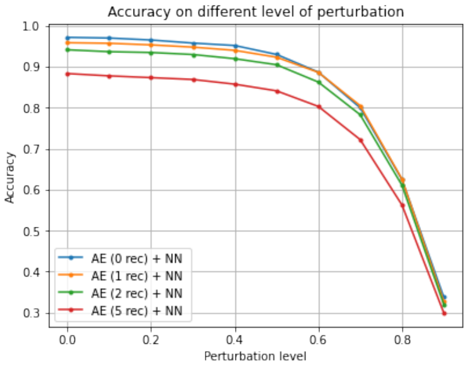
For this reason, we proceeded also to check the accuracy in the classification at different number of reconstructions, as done before with the DBN, for the AE+DBN+Perc:



Even though the images get sharper and sharper at each reconstruction of the autoencoder, the accuracy decreases.

4.2.2 AE+NN

Lastly, we have done the same with the AE+NN, and following are the results:



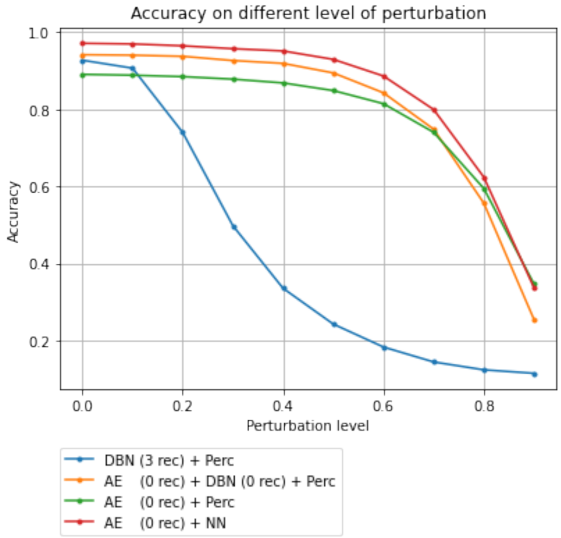
Generally speaking, it looks like that when we are dealing with autoencoders, multiple reconstructions, even if for the human eye the reconstruction gets better (up to a specific point), there is no improvement from a NN point of view (probably using CNN, which will also allow rotations, shifts and other transformations of the data to be ignored, this might not happen).

4.3. Final Evaluation

Lastly, we have compared the best version fo each model:

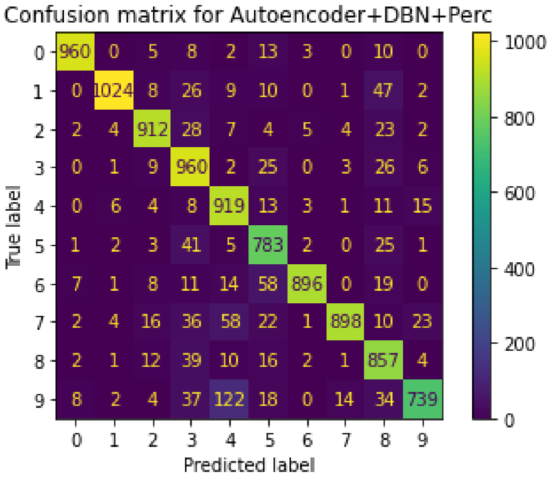
- AE (0 reconstructions)+DBN (0 reconstructions)+Perc
- AE (0 reconstructions)+NN
- AE (0 reconstructions)+Perc
- DBN (3 reconstructions)+Perc

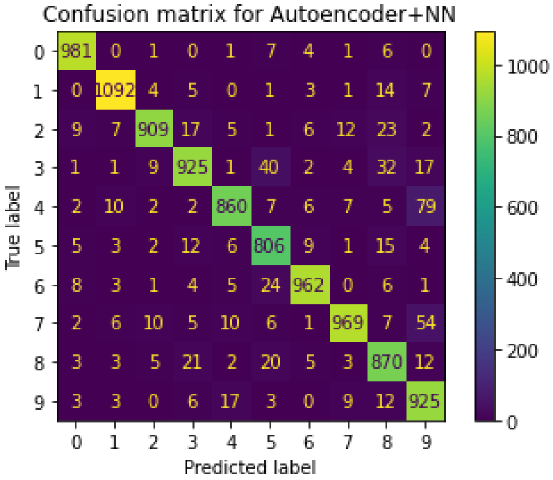
Following there is the graph with the resulting accuracies in classifications in function of different level of perturbation of those 4 models:



5. Models Error

From the previous image, we clearly see that the autoencoder is very important for each model to deal with the noise in the incoming images.
We then checked which errors were being made by the models, in particular from the AE+DBN+Perc and the AE+NN, specifically at level 0.5 since it's the point where the gradient of the curve of those 2 modes starts changing.
To evaluate the models, we have calculated the respective confusion matrices, which are the following images:

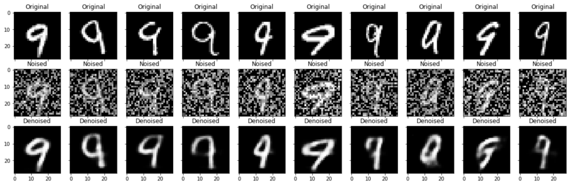




Both the matrices show evident difficulties in classifying 4 and 9, in particular:

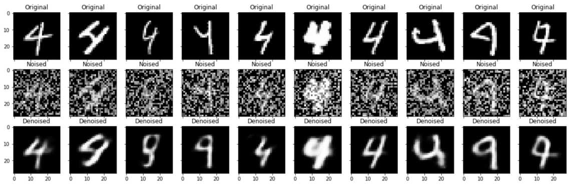
1. for AE+DBN+Perc a lot of 9 were predicted as 4
2. for AE+NN a lot of 4 where predicted as 9

Finally we checked some examples of missclassifications for those 2 models.
For AE+DBN+Perc, the following are missclassification of 9 predicted as 4:



What is most likely happening is that there are no more “clear traits” in some areas on the images after the denoising, and so probably, since the number 4 is a “more elastic” 9, not all features of those 9 are being captured by the DBN, and so they were exchanged by the perceptron as a 4 (and thus it’s like “taking out” some pieces from the 9, which probably becomes something like a 4)

For AE+NN, following are missclassifications of 4 predicted as 9:



The reason why the NN is missclassifying those images is probably because the digits where pretty similar to a 9 since the beginning, and the autoencoder, trying to denoise the images, has transformed them to a 9.
This is probably due to the layer between the encoder and the decoder, which will map those samples way more close to the average latent space of the number 9 than to the number 4.
This can also be seen by the fact that if we have a very thin digit since the beginning, and we add some noise over that, the noise might fill some gaps, so the noise becomes part of the “thin digit” itself, and can no more be disciminized by the autoencoder.
In addition to this, there is the fact that the big difference between the 4 and the 9, are the acute angles, and the encoder produce more “opaque” images, removing that acuteness.

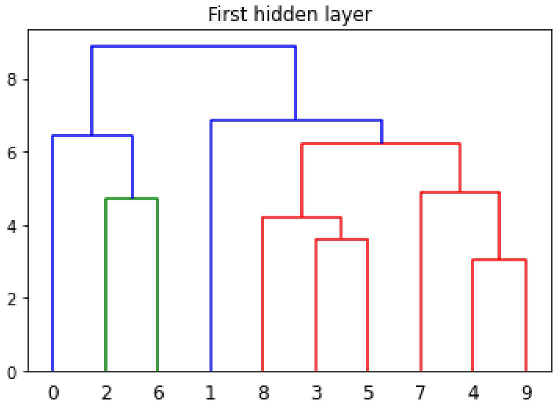
All of this contributes to the missclassification (but arguably some of those noised images might be even recognized by humans with more precision than those models)

6. Conclusions

What we can observe from the accuracy image in section §4.3, the encoder makes a very significant difference when dealing with noise, and this was the point of the project.
In addition to this, noteworthy:

1. already the autoencoder (AE+Perc) and the perceptron were enough to obtain a good accuracy when dealing with noised images
2. the DBN visibly helps a perceptron to deal with noised images, up to a certain point, but the intersection is almost at 70% noise level, the accuracy it’s around 70%, and the difference is not that significant
3. the NN, even though is significantly more complex than a perceptron, it doesn not hold such a big difference compared to the AE+DBN+Perc and even to AE+Perc
4. as previously already shown, the DBN with N reconstructions helps the perceptron dealing with noise, but nowhere near the autoencoder

Regarding the errors, in addition to what said before, we also generated the following dendogram, which shows how much similar are the digits for the DBN:



Clearly the pair 4-9 is very close in representation for the DBN, but also in real life, many times it happens to read one for the other.

7. Notes on the attached notebook

Some notes on the notebook:

1. Since Colab has limited resources, many parts of the code are being wrapped inside a `def free():` function, so that the objects created inside are being freed after the call
2. To avoid having to re-train the models, after the first training, the models will be saved in a file
3. The notebook, essay and trained models can be found at `https://github.com/AlbertoSinigaglia/cog-comp-project`