Autoencoders with Fashion-MNIST

Akshay Muraleedharan Nair Santhy, Amit Ranjan, Andrew Boateng and Kaustubh Milindrao Sardar

*Abstract*— This project implements and evaluates autoencoder models using the fashion-MNIST data to train and test. This project has two parts: (i) De-noising autoencoder: In this part, we train an autoencoder with one hidden layer to reconstruct de-noised images from noisy versions. We then vary the noise level in the input and evaluate the de-noising capabilities of the network. (ii) Stacked autoencoder: For this part, we train an autoencoder layer-by-layer in an unsupervised manner. Three of such layers are trained and stacked to create a stacked autoencoder that is an unsupervised feature extractor. A classifier is added on top of the final layer and the stacked autoencoder is fine-tuned. The classification accuracies are then evaluated on the test dataset. We compare our stacked autoencoder with other models.

*Index Terms*— Autoencoder, Denoising Autoencoder, Fashion- MNIST, Neural nets, Sparse autoencoder, Stacked autoencoder, Unsupervised learning, Gaussian noise, Classification.

# INTRODUCTION

A

N autoencoder is an unsupervised machine learning algorithm. Autoencoder preserves important aspects of data while losing extra or noisy aspects. Autoencoders consists of two parts: the encoder and the decoder. The encoder converts the input into a fewer number of bits. This conversion is known as encoding. Decoder reconstructs the input based on the encoded input.

## De-noising autoencoder

The intuitive application of autoencoder is image compression and then de-noising the image which is done in the project. The intuitive application of autoencoder is image compression and then de-noising the image which is done in the project. The motivation for de-noising autoencoders is to be able to reconstruct data from an input of corrupted data [3]. After giving the autoencoder the corrupted/ noisy data, we force the hidden layer to learn only the more prominent features, rather than just the identity. The resulting output will then be a more fine-tuned copy of the input as given in figure 1.

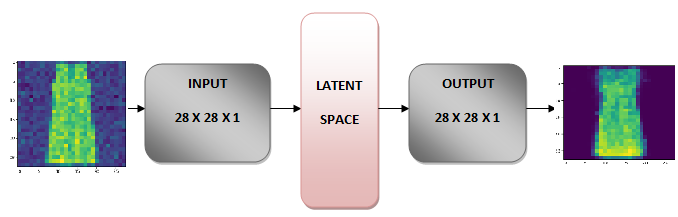
“This work was submitted on December 8th, 2018 as a mid-term report towards the course CSE-569, Fundamentals of Statistical Learning, at Arizona State University under the guidance of Dr. Hemanth Venkateswara.”

Akshay Muraleedharan Nair Santhy is a graduate student at Arizona State University, Tempe, AZ 85281 USA (e-mail: amurale1@asu.edu).

Amit Ranjan is a graduate student at Arizona State University, Tempe, AZ 85281 USA (e-mail: aranjan8@asu.edu).

Andrew Boateng is a graduate student at Arizona State University, Tempe, AZ 85281 USA (e-mail: aoboaten@asu.edu).

Kaustubh Milindrao Sardar is a graduate student at Arizona State University, Tempe, AZ 85281 USA (e-mail: kmilindr@asu.edu).



*Figure 1: Architecture of a denoising autoencoder*

## Stacked autoencoder

A stacked autoencoder, which is another part of the project, exploits all the characteristics of a deep network. While an autoencoder tries to learn aspects that form a good mimic of its input, in stacked autoencoder, the first layer of stacked autoencoder analyses basic features in the raw input. For example, edges are identified by the first layer when the image is provided as an input. The second layer of a stacked autoencoder uses the extracted features of the first layer and tries to find patterns in them. For images, it might be the corners and curves at the end of the edges. Similarly, higher layers perform higher order extraction. This is known as hierarchical grouping [4].

# Related Work

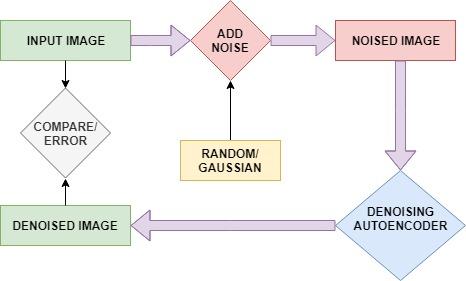
As defined in [5], the MNIST database is a large database of handwritten digits that is commonly used for training various image processing systems. As per [6], Fashion-MNIST is a dataset of Zalando's article images - consisting of a training set of 60,000 examples and a test set of 10,000 examples (same as MNIST).

In [2], the authors state the working of a basic autoencoder. The autoencoder takes an input vector and maps it to a hidden representation through a deterministic mapping. Loss function is defined as the reconstruction cross-entropy derived by the interpretation of and as either bit vectors or vectors of bit probabilities (Bernoulli’s).

An extension to the basic autoencoder is proposed to include denoising [4] and enforce robustness to partially destroyed inputs. The autoencoder is trained to reconstruct a clean “repaired” input from a corrupted, partially destroyed one. This is done by first corrupting the initial input to get a partially destroyed version by means of a stochastic mapping . The number of layers that that the autoencoder can have varies depending on the scenario and the need. Having multiple layers in the autoencoder is referred to as a Stacked Autoencoder. Both the methodologies will be discussed in the next section.

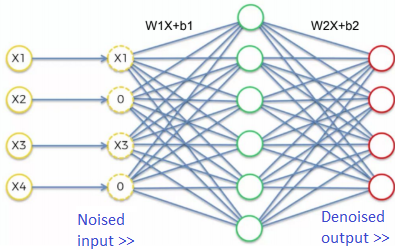
# Method

## De-noising autoencoder



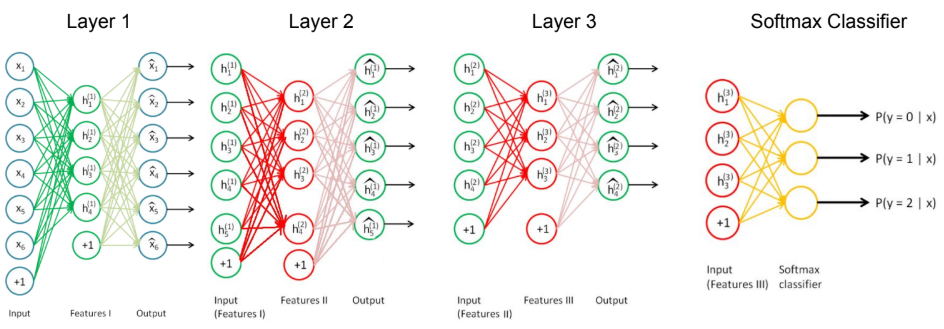
*Figure 2: Denoising autoencoder flowchart*

1. Training: We trained the network with noised images as input and their original images as expected output.
2. Testing: we provided a different set of noised images as input and got the Denoised images
3. Error Calculation: For each denoised image, we compared it with the original image using mean square error(MSE) and quantified the error.
4. Noise Functions: We have implemented two noise functions, a *Gaussian Noise* function that adds a Gaussian noise to every pixel in the given input image and a *Random noise* function, that changes a given fraction of input image’s pixel values to zero.
5. The architecture and best parameters: For the architecture, we kept the middle layer a constant 1000 nodes, inputs, and outputs are same as the image dimensions (784). To get the best-denoised image we tried with changing various parameters for the network, such as the learning rates, epochs, noises, and activation functions.



*Figure 3.1: Denoising Encoder*

## Stacked Autoencoder



*Figure 3.2: Stacked Encoder Architecture*

In the stacked autoencoder, we are training 3 sparse autoencoders layer by layer. The trained weights of each layer of the sparse autoencoder is taken as input to the next autoencoder. In the end, a SoftMax classifier is applied to the stack of these autoencoders as shown in figure 3.2. We perform fine-tuning with various no. of labels samples per class - 1, 5, 20, 50, 100. In traditional autoencoder, the hidden neuron units fire way too frequently which becomes undesirable as it does not add much significance to the network activation. Thus, to lower their activation, we add a sparsity constraint which is the moderator for the activation of units by adding KL-divergence to the objective function [7].

# Experiments

For each of the autoencoder model, we worked on various types of experiments which are elaborated below. The results of these experiments are discussed in the next section.

## De-noising autoencoder

For this part, we designed and implemented a denoising autoencoder by implementing the following experiments:

* *Introduced noise to an input*:

We used two different functions namely Gaussian noise and Random noise to introduce noise to the input. Our implementation of random noise function takes a specified fraction of the elements of an input and set them to zero whereas, for the Gaussian noise function, we add values to every input based on Gaussian distribution.

* *Trained and Tested using varying noise levels*:

We trained using different noise levels and for each noise level, we tested using a set of noises with the different noise functions we implemented (Gaussian and Random).

* *Tested the autoencoder with a different combination of activation functions*:

For testing our denoising autoencoder, we used different combinations of activation functions for different layers. We used tanh activation function from first to the second layer and sigmoid from second to the third layer (tanh - sigmoid), we also used sigmoid-sigmoid activation functions and compared the outcome (which is discussed in the Results section).

## Stacked Autoencoder

We designed a stacked autoencoder with 3 layers and a SoftMax classifier on top of the stacked layers. For this model, we performed the following experiments:

1. *Model testing with different parameters*

* Trained the network with 4 different architectures:
  + [500, 200, 100]
  + [500, 400, 50]
  + [500, 50, 20]
  + [300, 200, 100]
* Trained the network with various no. of epochs:

1000, 500, 200, 100, and 50

* Fine-tuned the network with various sizes of labeled data per class:
  + 1-labels per class
  + 5-labels per class
  + 20-labels per class
  + 50-labels per class
  + 100-labels per class
  + Fully labeled (entire dataset)

We experimented with all the combinations of the above-mentioned parameters to find the corresponding accuracies and evaluate them.

1. *Baseline accuracy compared with other models*

To get a proper estimate of our model’s performance, we implemented a few other models of supervised and unsupervised classification algorithms on the Fashion-MNIST datasets. For supervised algorithms, we created a Neural Net of 3 hidden layers implemented using TensorFlow- Keras. From sklearn library- we implemented SVM with a polynomial kernel and Logistic Regression with ‘lbfgs’ solver. Since stacked autoencoders are unsupervised, we have implemented K-Means algorithm also from sklearn library to compare the unsupervised classification accuracies too.

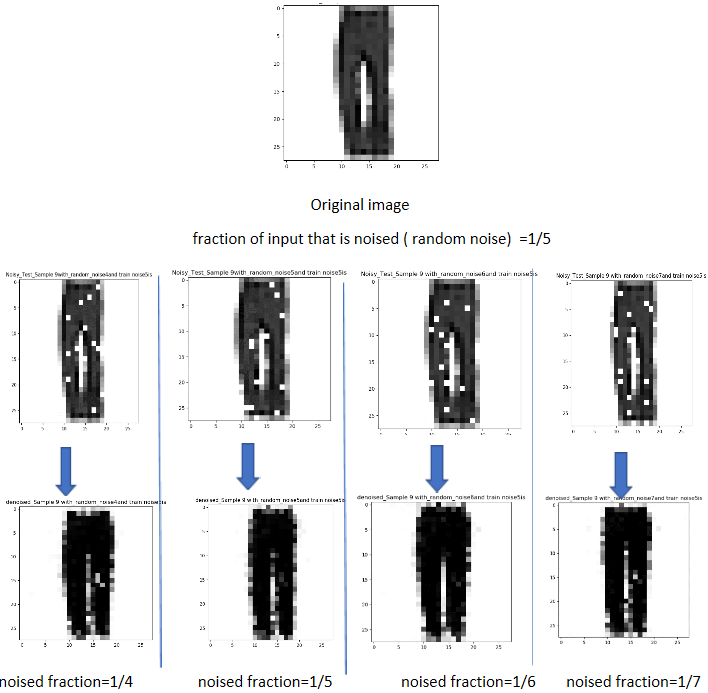
All the machines are trained on the complete 60,000 MNIST training images and tested on the 10,000 MNIST testing images. For Means, we classified the given training data unsupervised and used the unsupervised labels to train an SVM to map with the ground truth and calculate the accuracies of K-Means. The test accuracies are mentioned in the Results section.

# Results

## De-noising autoencoder

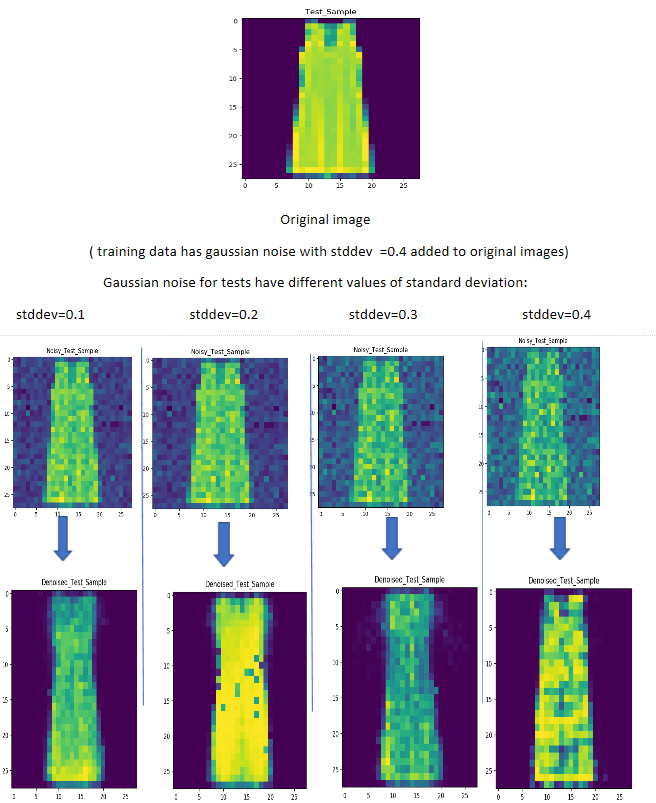
We tried different learning rates and epochs in our quest to find the combination that yields a good output image. Figure 4 and figure 5 below show, the input, noisy and denoised image when we used Epochs of 1000, the learning rate of 1, sigmoid → sigmoid activation function, random noise, and Gaussian function respectively.

1. Random Noise:



*Figure 4: Input, noisy and denoised images using Random Noise, Epochs=1000, learning rate=1, sigmoid → sigmoid activation*

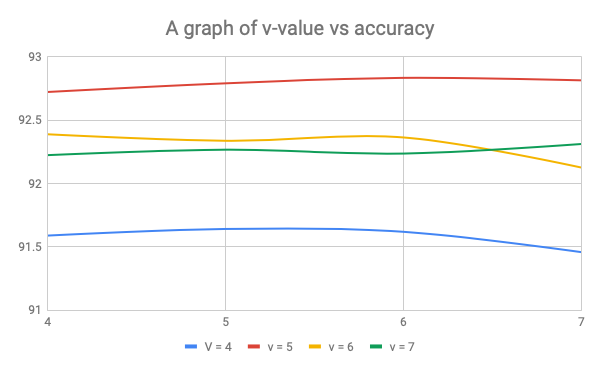
1. Gaussian Noise:



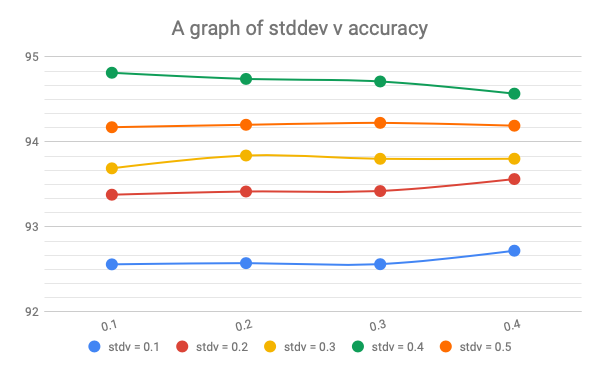
*Figure 5: Input, noisy and denoised images using Gaussian Noise, Epochs=1000, learning rate=1, sigmoid → sigmoid activation.*

For a better outcome, we trained our dataset with a certain level of noise and then tested with different noise levels. The results are shown in figures 6.1 and 6.2. For example in figure 6.1, It can be seen with the blue line that we trained with a v-value (the fraction of elements that are noised) of 4 and tested with different v-values (4, 5, 6 and 7). It was observed that at first, increasing the v-value improved accuracy but began decreasing with increasing v-value after v = 5. We also observed that at v-5, we got a better accuracy than the rest.

To vary the noise of the Gaussian noise level, we varied the standard deviation. The result is shown in figure 6.2. The accuracy increased when the value of the standard deviation is increased. This pattern reversed after we reached 0.5 standard deviation. We observe a decrease in accuracy which may be because we are adding too much noise. In general, accuracy for Gaussian noise is more than random noise.



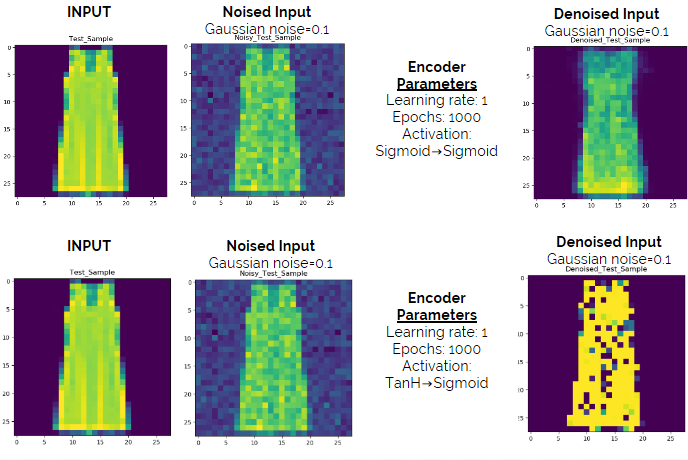
*Figure 6.1: A graph of v versus Accuracy for random noise*



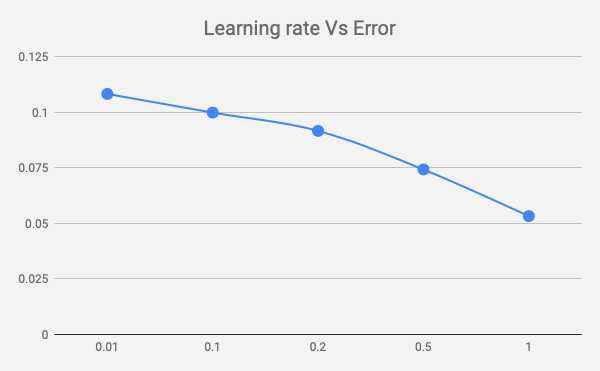
*Figure 6.2: A graph of standard deviation versus accuracy\for gaussian noise*

1. Activation Function:

As shown in figure 7, the sigmoid->sigmoid activation gives better results than sigmoid->tanh



*Figure 7: Input, noisy and denoised images using Gaussian Noise, Epochs=1000, learning rate=1, sigmoid -> sigmoid activation, .and tanh -> sigmoid activation*

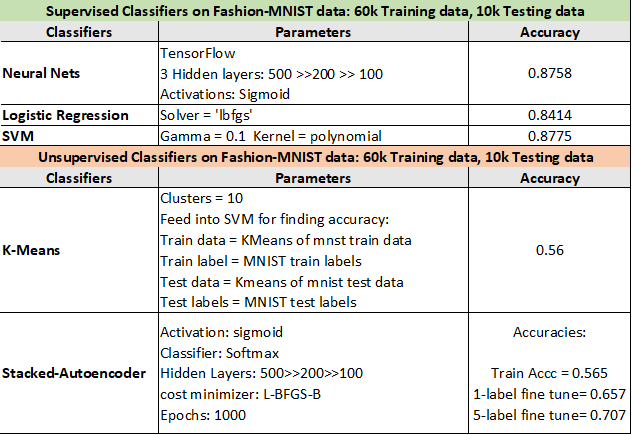


*Figure 8: A graph of learning rate versus Error*

## Stacked Autoencoder

Complete tabular experimental results on the combinations of the parameters mentioned in the section *IV.B.(i)* above have been attached as ***Stacked\_Autoencoder\_Result\_Sheet.pdf***

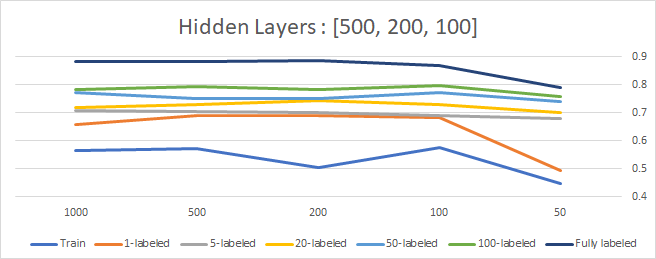
To evaluate our model and compare it with a benchmark. Since we evaluated Neural Net, LogReg, SVM, and K-Means, the accuracies obtained on the Fashion-MNIST dataset using these models with the hyper-parameters are mentioned below:



The mean accuracies across multiple architectures and iterations are mentioned below:

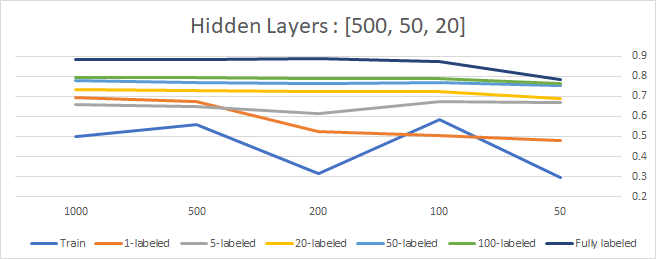
|  |  |  |  |
| --- | --- | --- | --- |
| **Network** | **Tr. Acc.** | **1-label FT** | **5-label FT** |
| [500, 200, 100] | 0.534 | 0.643 | 0.696 |
| [500, 400, 50] | 0.541 | 0.648 | 0.694 |
| [500, 50, 20] | 0.451 | 0.575 | 0.653 |
| [300, 200, 100] | 0.544 | 0.641 | 0.686 |

In most of the cases, the accuracy of a fine-tuning with 1-label per class was found to be lower as compared to a fine-tuning with 5-labels or 50-labels per class as shown below in Fig. 9.



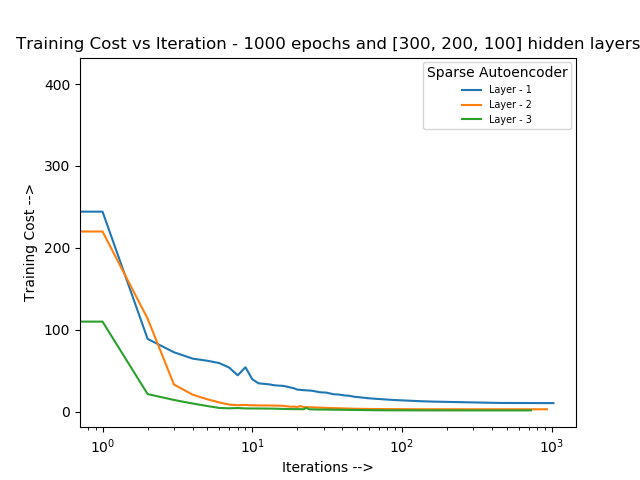
*Figure 9: Fine-tuning Accuracy vs Epochs on [500, 200, 100] hidden layers*

For the hidden layers [500, 50, 20], we observed instances where a fine-tuning with 1-label performed better than a fine-tuning with 5-labels per class.



*Figure 10: Fine-tuning Accuracy vs Epochs on [500, 50, 20] hidden layers*

The graph below shows the behavior of cost vs epochs:



# Conclusion

## De-noising autoencoder

1. From the results of the Denoising Autoencoder, we concluded that a learning rate of 1.0 and no. of epochs=1000 gives the least error. Also,   
   (sigmoid->sigmoid) activation does better job than (tanh->sigmoid).
2. Gaussian noise is found to be more accurate as compared to random noise. This is because Gaussian noise adds noise to the whole input forcing the autoencoder to learn better.
3. Increase in noise level for training increases accuracy for testing till a point, but when the noise is too much, accuracy decreases (figure 6.1 and figure 6.2).

## Stacked Autoencoder

1. We have observed the stacked autoencoder performs well with an implementation of Sparse autoencoder
2. The Stacked autoencoders should have comparable hidden layer sizes to get better accuracies
3. Fine tuning has improved the accuracies, adds the supervised training modularity to the mix

# Division of Work

|  |  |  |
| --- | --- | --- |
| **#** | **TASKS** | **Team Member** |
| 1 | Research papers and get information about Denoising and Stacked autoencoders | Akshay, Amit  Andrew, Kaustubh |
| 2 | Implement denoising autoencoder | Akshay, Andrew, Kaustubh |
| 3 | Implement Different Noise Functions (Gaussian/ Random) | Andrew, Kaustubh |
| 3 | Implement cost and error functions for Denoising Autoencoder | Andrew, Kaustubh |
| 4 | Implement and execute all the test cases for Denoising Autoencoder | Akshay, Amit  Andrew, Kaustubh |
| 5 | Find the best parameters and plot all the results | Andrew, Kaustubh |
| 6 | Implement 3 Sparse autoencoders and connect them | Amit, Akshay |
| 7 | Implement SoftMax classification | Amit, Akshay |
| 8 | Implement fine tuning with 1 and 5 samples sets | Amit, Akshay, Kaustubh |
| 9 | Implement and execute all the test cases for Stacked Autoencoder | Akshay, Amit  Andrew, Kaustubh |
| 10 | Find the best parameters and plot all the results | Akshay, Andrew, Amit |
| 11 | Baseline classifier machines to compare the Stacked Autoencoder | Akshay, Amit |
| 12 | Mid Term and Final Reports | Akshay, Amit  Andrew, Kaustubh |

# Self-Peer Evaluation Table

|  |  |
| --- | --- |
| Akshay M. N. Santhy | 20 |
| Andrew Boateng | 20 |
| Kaustubh Milindrao Sardar | 20 |
| Amit Ranjan | 20 |

References

1. H. Xiao, K. Rasul, and R. Vollgraf, “Fashion-MNIST: a Novel Image Dataset for Benchmarking Machine Learning Algorithms” in *arXiv:1708.07747*, 2017.
2. P. Vincent, H. Larochelle, Y. Bengio, and P.-A. Manzagol, “Extracting and composing robust features with denoising autoencoders,” in *Proceedings of the 25th international conference on Machine learning*, ACM, 2008, pp. 1096–1103.
3. H. Larochelle, *Neural networks [6.6]: Autoencoder - denoising autoencoder*, 2018 (accessed December 1, 2018). [Online].  
   Available: https://www.youtube.com/watch?v=t2NQ c5BFOc
4. A. Ng, *Stacked Autoencoders*, 2018 (accessed December 1, 2018). [Online]. Available:

http://deeplearning.stanford.edu/wiki/index.php/Stacked Autoencoders

1. Wikipedia, *MNIST Database*, 2018 (accessed December 1, 2018). [Online]. Available: https://en.wikipedia.org/wiki/MNIST\_database
2. Zalandoresearch, *Fashion-MNIST*, 2017 (accessed December 2018). [Online]. Available: https://github.com/zalandoresearch/fashion-mnist
3. Andrew Ng, CS294A Lecture notes – Sparse Autoencoder (accessed December 2018). [Online].   
   Available: https://web.stanford.edu/class/cs294a/sparseAutoencoder.pdf