Image Reconstruction using Denoising Autoencoder and Classification of Fashion MNIST Data using Stacked Autoencoders

Manoj Yaramsetty, Vishal Gade, Sesha Chandra Adusumilli

I. INTRODUCTION

IN recent years, improved usage of deep learning techniques in the field of feature learning has enhanced the performance of image processing tasks, such as image classification and object detection. Deep neural networks (DNN) with the ability to imitate latent features from data using their deep layer-wise architecture gains the potential to classify patterns. However, in most cases, the performance of these DNNs for image classification is constrained as natural data is adulterated by noises. For a good classification, DNNs requires good preprocessed data when they are being trained. Noisy data affects DNNs learning and thus produce a bad classification outcome.

To solve this issues regarding noise, deep learning researchers in recent years have successfully employed deep architecture based mechanisms(auto encoders) to achieve image denoising and proved that these procedures achieve better performances than traditional statistical techniques. Further, these autoencoders are stacked to construct a deep unsupervised learning network called Stacked Autoencoders. These networks can be used for efficient classification tasks. This paper talks about the implementation and use cases of both Denoising Autoencoders (DAN's) and Stacked Autoencoders.

Section 2 introduces readers to various topics necessary to understand the paper. Starting from Autoencoders, we present a brief understanding of Denoising Autoencoder and then extend it to Stacked Autoencoders. Section 3 introduces pre-existing works and approaches. Section 4 starts with the description of Dataset and then walk the readers through the working mechanism of Denoising Autoencoder. We then demonstrate the use of Stacked Autoencoders. Section 5 presents experiments that qualitatively study the feature detectors learned by a single-layer denoising autoencoder under various conditions and describe experiments with multi-layer architectures obtained by stacking autoencoders and compare their classification performances. Section 6 summarizes our findings and concludes our work.

II. BACKGROUND

This section gives a short introduction to topics we implement and functionality behind them.

A. Autoencoders

An autoencoder is a neural network that is trained to translate given input to its output, by performing dimensionality reduction, which is a process of reducing the number of random variables under consideration. It features an encoder function that describes the input, which is used to create a single or multiple hidden layers.

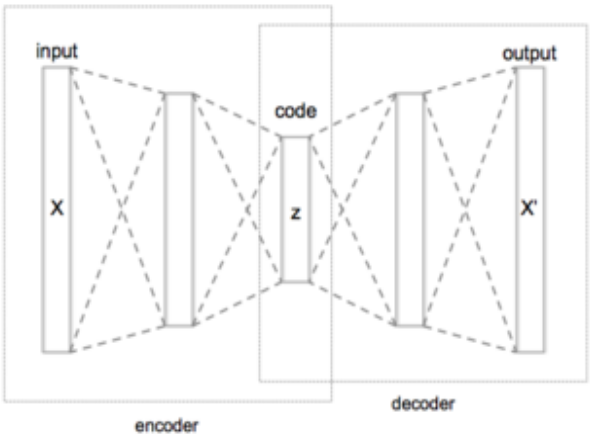

Fig. 1. Structure of an Autoencoder with 3 fully connected hidden layers.

Through an unsupervised learning algorithm for linear reconstruction, the autoencoder will try to learn a function $h_{W,b}(x) = x$, so as to minimize the mean square difference:

$$(L(x, y) = \Sigma((x - h_{W,b}(x))^2) \quad (1)$$

where x is the input data and y is the reconstruction. However, when the decoders activation function is the Sigmoid function, the cross-entropy loss function is typically used:

$$L(x, y) = \Sigma_{i=1}^{d_x} x_i log(y_i) + (1 - x_i) log(1 - y_i) \quad (2)$$

We can obtain optimum weights for this by starting with random weights and calculating a gradient. This is done by using the chain rule to back-propagate error derivatives through the decoder network and then the encoder network.

B. Denoising Autoencoders

The principle behind denoising autoencoders is to be able to reconstruct data from an input of corrupted data. After giving the autoencoder the corrupted data, we force the hidden layer to learn only the more robust features, rather than just the identity. The output will then be a more refined version of the input data.

We can train a denoising autoencoder by stochastically corrupting data sets and inputting them into a neural network.

The autoencoder can then be trained against the original data. One way to corrupt the data would be simply to randomly remove some parts of the data so that the autoencoder is trying to predict the missing input.

An example of this is shown below, with the image an autoencoder is trained on being on the left, and a reconstruction of the middle image on the right.

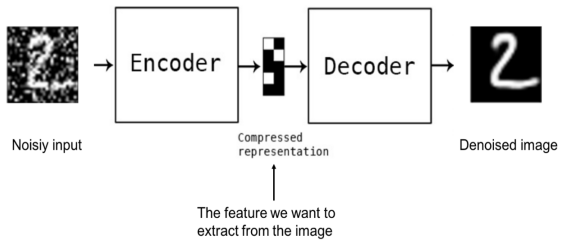

Fig. 2. Structure of a Denoising autoencoder with an example.

C. Stacked Autoencoders

A stacked autoencoder is a neural network consisting of multiple layers of sparse autoencoders in which the outputs of each layer is wired to the inputs of the successive layer. Formally, consider a stacked autoencoder with n layers. Using notation from general neural networks, let $W^{(k,1)}$,$W^{(k,2)}$,$b^{(k,1)}$,$b^{(k,2)}$ denote the parameters $W^{(1)}$,$W^{(2)}$,$b^{(1)}$,$b^{(2)}$ for kth autoencoder. Then the encoding step for the stacked autoencoder is given by running the encoding step of each layer in forwarding order:

$$a^l = f(z^l) \tag{3}$$

$$z^{l+1} = W^{l,1}a^l + b^{l,1} \tag{4}$$

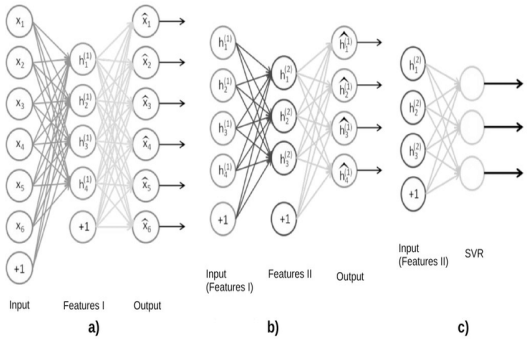

Fig. 3. Layers of Stacked Autoencoder.

The decoding step is given by running the decoding stack of each autoencoder in reverse order:

$$a^{n+l} = f(z^{n+l}) \tag{5}$$

$$z^{n+l+1} = W^{n-1,2}a^{n+l} + b^{n-l,2} \tag{6}$$

The information of interest is contained within a(n), which is the activation of the deepest layer of hidden units. This vector gives us a representation of the input in terms of higher-order features.

The features from the stacked autoencoder can be used for classification problems by feeding a(n) to a softmax classifier. Fig(4) shows the final structure of stacked autoencoders with 2 hidden layers.

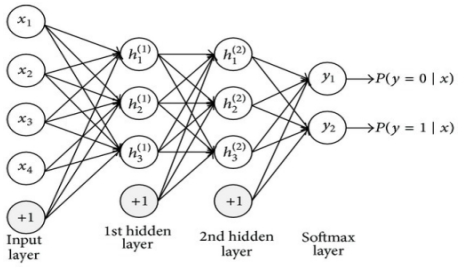

Fig. 4. Final Structure of Stacked Autoencoder.

III. Related Work

The current work has already been discussed and implemented using different techniques. Many advanced techniques are implemented using Deep Learning libraries such as tensorflow, caffee etc. Among them, [7] proposes a denoising sparse autoencoder (dsAE) which combine sparsity constraint and corrupting operation. Their results show that their approach yielded higher test accuracy compared to other deep neural networks. The current work is mostly a learning experience, where we as a team try to implement neural networks from scratch. We wanted to develop a system that almost produces start-of-the-art accuracy.

IV. Method

A. Dataset Description

We collected images from Fashion-MNIST dataset which consists of 60000 examples of train examples and a test set of 10,000 examples. Each example is a 28x28 grayscale image, associated with a label from 10 classes. Value of each pixel in images is converted to lie between 0 to 1.

B. Denoising Autoencoders

To demonstrate the working of DAN, we manually added noise to our train data set using the following algorithm:

```
thres = 1 - prob
for i = 0 → image.shape[0] do
  for j = 0 → image.shape[1] do
    if rdn < prob then
      image[i][j] = 1
    else if rdn > thres then
      image[i][j] = 0
    end if
  end for
end for=0
```

Next, the DAN network takes an input x' and with the help of an encoder maps it to a hidden representation y,

through a deterministic mapping y = s(Wx +b) where s is a non-linearity such as the sigmoid. The latent representation y, is then mapped back with the help of a decoder into a reconstruction z of the same shape as x. The mapping happens through a similar transformation z = s(W'y +b'). Parameters W, W, b, b' are trained to minimize the average reconstruction error over a training set, that is, to have z as close as possible to the uncorrupted input x. The considered reconstruction error is the cross-entropy loss (equation 1). Parameters are initialized at random and then optimized by using the chain rule to back-propagate error derivatives through the decoder network and then the encoder network.

C. Stacked Autoencoders

For Stacked Autoencoders, to obtain parameters we train the layers independently using hidden units of previous layers. First, we train the first layer on input to obtain parameters $W^{1,1}$, $W^{1,2}$, $b^{1,1}$, $b^{1,2}$. Then, transform the input into a vector consisting of activation of the hidden units using the first layer. Next, We train the second layer on this vector to obtain parameters $W^{2,1}$, $W^{2,2}$, $b^{2,1}$, $b^{2,2}$. Repeating the same procedure for subsequent layers, using the output of each layer as input for the subsequent layer, this method trains the parameters of each layer individually while freezing parameters for all other layers. In this paper, we train 3 such layers. Following this training, the softmax classification layer is added on the final layer. Finally, fine-tuning using backpropagation is done to improve the results by tuning the parameters of all layers at the same time.

V. Experiments

Experiments on both Denoising and Stacked Autoencoder use below mentioned dataset description for training and testing phases.

- Train data size - 6000 samples (600 samples from each of 10 classes)
- Test data size - 600 samples (60 samples from each of 10 classes)

A. Denoising Autoencoder(DAN)

We carried out multiple experiments on denoising autoencoders with varied values of train and test noises. The dimensions of the network used are 784, 256, 784 and learning rate = 0.001 for 1000 iterations.

1) Experiment: We trained 2 layer network (excluding input layer) with train noise = 0.1. Fig(5) demonstrates the training cost vs iterations plot.

After training the network, We tested this autoencoder with the same noise level and the reconstruction of images was almost similar to the original images. The results can be seen in Fig(6).

2) Experiment: We increased the train noise level to 0.3 and trained the autoencoder. As noise level is high, there are few discrepancies during the initial phase of training. Below Fig(7) shows the training cost vs iterators plot.

To test the behavior of DAN, we tested with test noise = 0.5. Below Fig(8) shows the reconstructed images using DAN.

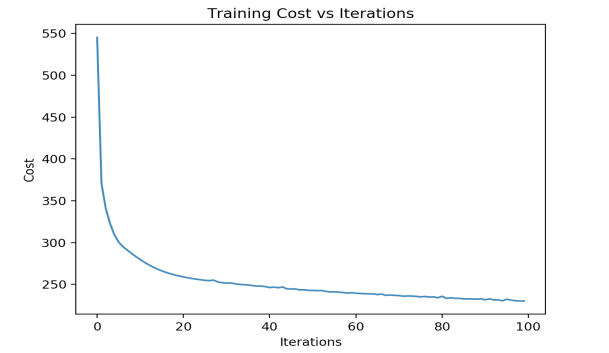

Fig. 5. Training Cost vs Iterations for Train Noise = 0.1

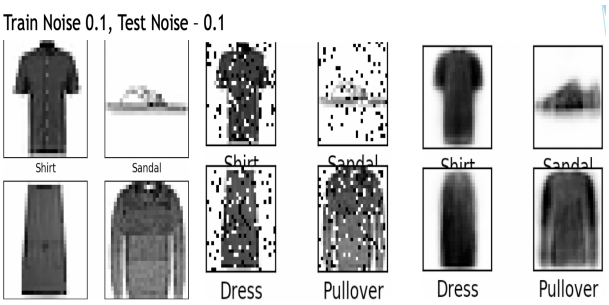

Fig. 6. Original image, Noisy image, Denoised image

Fig. 7. Training Cost vs Iterations for Train Noise = 0.3

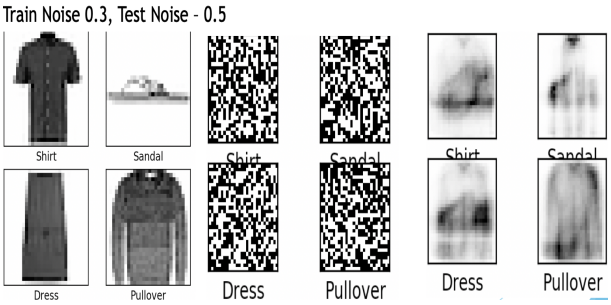

Fig. 8. Original image, Noisy image, Denoised image

3) Experiment: Till now, we trained and tested with the low noise level. Next we want to increase the noise further to 0.5 and train the network to see how it learns noise and

decodes images. Fig(9) shows plot for the training cost vs iterations.

Fig. 9. Training Cost vs Iterations for Train Noise = 0.5

Since we trained with the high noise level, we tested with low test noise level = 0.3. Fig(10) shows the reconstructed images from noisy images using this DAN.

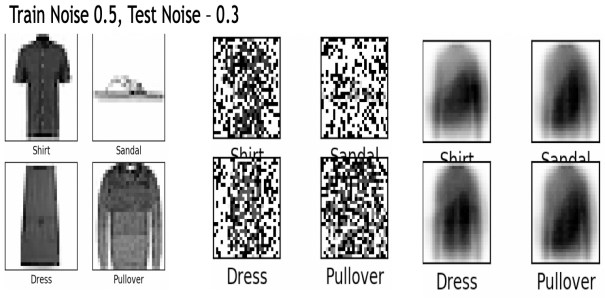

Fig. 10. Original image, Noisy image, Denoised image

4) Experiment: After testing the DAN with the low noise level, we now test the DAN with high noise level equal to the train noise level. Fig(11) shows the reconstructed images from noisy images using this DAN.

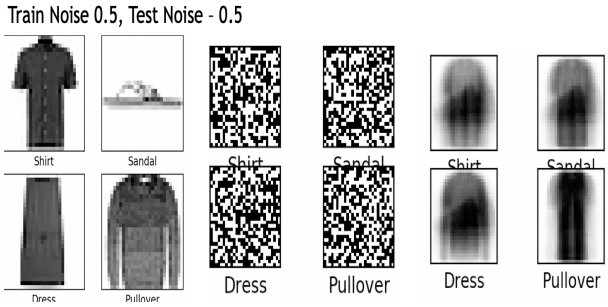

Fig. 11. Original image, Noisy image, Denoised image

B. Stacked Autoencoders

For this part, we conducted multiple experiments with three different architectures. First architecture has dimensions (784-500-200-100) where 784 is the size of the input layer and 500,200,100 are the sizes of hidden layer 1,2 and 3. Second, third architectures have (784-300-100) and (784-100) respectively. We trained each hidden layer independently with learning rates of 0.01, 0.005 and 0.02. Then we added a classification layer on top of final layer. For each architecture, we tuned the neural network with 1 sample per class and 5 samples per class. Table I and Fig. 12 shows the training cost during training of hidden layers 500, 200, 100.

Stacked Autoencoder Architecture 1			
Dimensions	Learning Rate	No of iterations	Final Cost
784 500 - 784	0.01	1000	255.973
500 - 200 - 500	0.005	700	0.25
200 - 100- 200	0.02	500	9.13

TABLE I

After training the hidden layers, we tuned the network with 1 sample per class, training only the classification layer and almost freezing hidden layers. We also, tuned the network with various combinations of learning rates for classification layers and hidden layers. Table II shows the training cost, training accuracy and test accuracy for each combination of learning rates.

Fine Tuning 1 samples per class, Architecture 1					
No of Iterations	Learning rate for last year	Learning rate for other layers	Final Cost	Training Accuracy	Test Accuracy
2000	1	0	0.00352	51.180	51.167
2000	0.1	0.001	0.08165	51.340	51.167
2000	0.5	0.00001	0.01451	51.300	51.000

TABLE II

Next, using same architecture and not changing the parameters we tuned the network with 5 samples per class. Table III shows the training cost, training accuracy and test accuracy for each combination of learning rates.

Fine Tuning 5 samples per class, Architecture 1					
No of Iterations	Learning rate for last year	Learning rate for other layers	Final Cost	Training Accuracy	Test Accuracy
2000	1	0	0.08322	67.280	63.033
2000	0.1	0.001	0.37515	66.180	64.7
2000	0.5	0.00001	0.14674	65.140	63.250

TABLE III

For the **second architecture**, we removed a layer (Hidden layer with size 500) and trained two hidden layers in a unsupervised manner. Table IV shows the training cost for each combination of learning rate.

Same as architecture 1, we tuned the network with 1 and 5 labeled samples per class. Tables V and VI shows costs and accuracy for training and testing for various combinations of learning rate.

In **third architecture**, we removed two layers and trained the network. We repeated same evaluations for tuning the

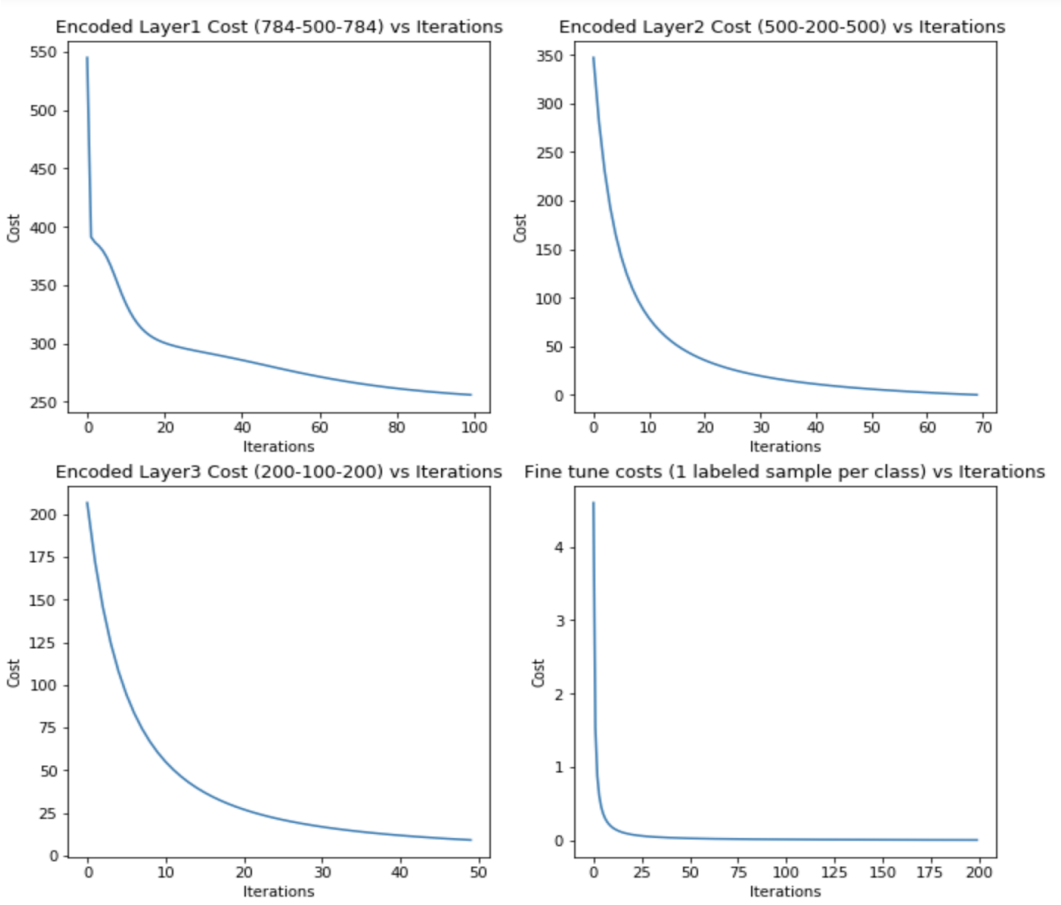

Fig. 12. Layer wise training cost curves for Stacked Autoencoder.

Stacked Autoencoder – Architecture 2			
Dimensions	Learning Rate	No of iterations	Final Cost
784 300 - 784	0.01	1000	260.353616
500 - 100 - 500	0.005	1000	6.152187

TABLE IV

Fine Tuning 1 samples per class Architecture 2					
No of Iterations	Learning rate for last year	Learning rate for other layers	Final Cost	Training Accuracy	Test Accuracy
2000	1	0	0.013322	52.280	51.033
2000	0.1	0.001	0.027515	51.455	51.30
2000	0.5	0.00001	0.087674	52.260	51.897

TABLE V

Fine Tuning 5 samples per class Architecture 2					
No of Iterations	Learning rate for last year	Learning rate for other layers	Final Cost	Training Accuracy	Test Accuracy
2000	1	0	0.00722	64.570	64.140
2000	0.1	0.001	0.01790	64.940	63.956
2000	0.5	0.00001	0.024674	65.540	65.250

TABLE VI

.

Stacked Autoencoder - Architecture 3			
Dimensions	Learning Rate	No of iterations	Final Cost
784 - 100 - 784	0.01	1000	313.080

TABLE VII

.

network. Table VII shows the training cost for different combinations of learning rate.

Next, we tuned the network with 1 labeled samples per class. Tables VIII gives the detailed evaluation of the experiment.Further, we tuned the network with 5 labeled samples per class. Table IX demonstrates the training and test costs and accuracy.

VI. Conclusions

A. *Denoising Autoencoders*

We have conducted 4 experiments in this part, where we varied both train and test noises. We trained and tested the network with various combinations of high and low train and test noise levels and analyzed those results.

Fine Tuning 1 samples per class - Architecture 3					
No of Iterations	Learning rate for last year	Learning rate for other layers	Final Cost	Training Accuracy	Test Accuracy
2000	0.1	0.0	0.0000	55.595	54.295
2000	0.5	0.001	0.00365	55.34	55.027
2000	1	0.00001	0.00006	54.540	54.833

TABLE VIII

.

Fine Tuning 5 samples per class - Architecture 3					
No of Iterations	Learning rate for last year	Learning rate for other layers	Final Cost	Training Accuracy	Test Accuracy
2000	0.1	0.0	0.0000	70.100	69.333
2000	0.5	0.001	0.00153	70.300	70.167
2000	1	0.00001	0.000059	69.580	69.833

TABLE IX

.

1) Low Train Noise = 0.1, Low Test Noise = 0.1
2) Low Train Noise = 0.3 , High Test Noise = 0.5
3) High Train Noise = 0.5 , Low Test Noise = 0.3
4) High Train Noise = 0.5 , High Test Noise = 0.5

Case 1: In this case, the reconstruction was almost equal to original image. This is because, since the train noise level is low, the network is able to learn important features and reconstruct the original image.

Case 2: In this case, the reconstruction is satisfactory, but is fairly less than case 1. In this case, the network was able to learn important features, however reconstruction is comparatively blurred since test noise is high.

Case 3: In this case, the network is not able to learn important features since the train noise level is high. Moreover, since the test noise is also little high, the reconstructed images were almost similar to each other and blurred.

Case 4: In this case, the network is not able to learn important features, since the train noise level is very high. Thus reconstructed images were almost similar to each other and blurred.

B. Stacked Autoencoders

We have conducted 3 experiments in this part, where we varied hidden layers. Below are the types of architectures we evaluated:

- Architecture-1 [784-500-300-100]
- Architecture-2 [784-300-100]
- Architecture-3 [784-100]

Surprisingly, as the network size increases, there is a drop in accuracy. We think its because of less complexity in Fashion MNIST dataset, as it has only 784 dimensions. For this dataset there is no need of deep networks to learn features. Apart from this, we changed the learning rates of last layer, but that did not increase the accuracy of the network. Fine tuning is the only part that impacted the accuracy. 5 labeled sample per class has an edge over 1 labeled sample per class, as last layer can see more samples for training. As increasing in sample size increases the accuracy, we conducted one more experiment with 10 labeled samples. We found that 10 labeled samples are giving 76.7% accuracy. So, giving high labeled samples improves the network accuracy.

Moreover, we compared our network with other models. Below table shows accuracy of different networks:

Model	Accuracy
Network1(784-500-300-100)	64.7
Network2(784-300-100)	64.7
Network3(784-100)	**70.167**
KNN Classifier	55.4
CNN Classifier	93.43

TABLE X

.

To conclude, our network has performs more than the baseline model. But it couldn't produce state of the art accuracy.

VII. Division of Work

Manoj Yaramsetty - Worked on both Denoising Autoencoders and Stacked Autoencoders

- DAN - Experiment 1,2
- Stacked Autoencoder- Architecture 1 and 3

Vishal Gade - Worked on both Denoising Autoencoders and Stacked Autoencoders

- DAN - Experiment 3,4
- Stacked Autoencoder- Architecture 2 and 3

SeshaChandra Adusumilli - Worked on both Denoising Autoencoders and Stacked Autoencoders

- DAN - Experiment 3
- Stacked Autoencoder - Architecture 3 and experiment on 10 labelled sample.

VIII. Self Peer Evaluation

Manoj Yaramsetty 20	Vishal Gade 20	SeshaChandra Adusumilli 20

References

[1] https://www.doc.ic.ac.uk/ js4416/163/website/autoencoders/
[2] http://ufldl.stanford.edu/tutorial/unsupervised/Autoencoders/
[3] https://towardsdatascience.com/autoencoders-are-essential-in-deep-neural-nets-f0365b2d1d7c.
[4] https://towardsdatascience.com/denoising-autoencoders-explained-dbb82467fc2.
[5] Vincent Pascal et al., "Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion", Journal of Machine Learning Research, vol. 11, pp. 3371-3408, 2010.
[6] Vincent Pascal et al., "Extracting and composing robust features with denoising autoencoders", Proc. of the 25th international conference on Machine learning. ACM, pp. 1096-1103, July 2008.
[7] https://link.springer.com/content/pdf/10.1007%2Fs00521-016-2790-x.pdf