Multi-class classification using Unsupervised Learning [Auto Encoder, KMeans, Gaussian Mixture Models]

Shivaleela Ishwarappa Hubli
Department of Computer Science
University at Buffalo
Buffalo, NY 14260
shivalee@buffalo.edu

Abstract

The aim of the project is to perform cluster analysis on Fashion MNIST dataset using Unsupervised learning. The objective is to cluster using various models and come up with the optimal prediction. The approach is to determine the accuracy of naïve KMean clustering and also compare it with the accuracy obtained by using the encoded images using autoencoder. Also determine how the accuracy varies when try to cluster the encoded data using the Gaussian Mixture Model.

1 Introduction

Unsupervised learning is a type of machine learning algorithm that draw inferences from unlabeled data. Most common unsupervised learning is the cluster analysis which is used to find pattern in the data and group them into clusters [1]. One such clustering algorithm is KMeans. It is the simplest and most popular unsupervised learning algorithm. In KMeans you basically group similar data points together into clusters.

The only catch here is that the number of clusters, a parameter which KMeans cannot determine and have to be manually fed to the algorithm.

2 Dataset

The dataset is the Fashion MNIST dataset. It is a 10-class classification dataset with 70,000 images. It contains 60,000 training images and 10,000 testing images. Each image is grayscale or single channel image with 28x28 pixels [3].

The ten class labels include: (0) T-shirt/top, (1) Trouser/pants, (2) Pullover shirt, (3) Dress, (4) Coat, (5) Sandal, (6) Shirt, (7) Sneaker, (8) Bag and (9) Ankle boot. Each pixel value is the intensity ranging between 1 and 255.

3 Preprocessing

The data that is given is just a table of tons of content. It is difficult to work with crude data. Hence preprocessing is a vital step for the implementation of the model. In this step the data is preprocessed using various techniques based on different requirements of the given three tasks. In the given problem the data has useful information in terms of pixel values required to train the model and the labels as results required to find the ground truth of clustering. The following set of preprocessing is done on the given dataset.

3.1 Splitting of dataset

The purpose of splitting the dataset is basically to avoid overfitting problem. Overfitting of a model, in simple terms is when the model memorizes the data. The objective of the project is to train a model that is generalized and provide undistorted prediction for unseen data inputs.

3.1.2 Train and test sets

In order to achieve the objective, the data is split into training and testing datasets. Training set, as the name specifies, is used to train the model for updating the parameters of the model. The testing set is used to evaluate the model and the parameters. The training and testing sets are split in the ratio of 6:1 respectively. There are a lot of libraries available to achieve this.

3.2 Normalization of data

Normalization is a technique used to bring the values of various features into a common scale and avoid any biasing based on the range of the feature values. The pixel values are divided by 255 to make the inputs range between 0 and 1. This has an advantage of reduction in computation cost as we use these values in a sigmoid

function which computes the exponential values of these input. These operations are done on training and testing sets.

4 Tasks

The Task 1 is to perform clustering using KMeans algorithm. In Task 2, the higher dimensionality of dataset is condensed using auto encoder and clustered using KMeans. In Task 3, the higher dimensionality of dataset is condensed using auto encoder and clustered using Gaussian Mixture Model.

4.1 Task 1

The task 1 is to design and build a model of clustering using unsupervised learning algorithm naive KMeans for various k values ranging from k = 9 to k = 11.

4.1.1 KMean

Given a set of observations (x_1, x_2, ..., x_n), where each observation is a d-dimensional real vector, k-means clustering aims to partition the n observations into k (≤ n) sets S = {S_1, S_2, ..., S_k} so as to minimize the within-cluster sum of squares. Formally, the objective is to find [2]:

$$\arg\min \sum_{i=1}^{k} \sum_{x \epsilon S_i} \|x - \mu_i\|^2 = \arg\min \sum_{i=1}^{k} |S_i| \, Var \, S_i$$

where μ_i is the mean of points in S_i. This is equivalent to minimizing the pairwise squared deviations of points in the same cluster:

$$\arg\min \sum_{i=1}^{k} \frac{1}{2|S_i|} \sum_{x,y \in S_i} \|x - y\|^2$$

Given an initial set of k means $m_1^{(1)}$, ..., $m_k^{(1)}$, the algorithm proceeds by alternating between two steps.

Assignment step: Assign each observation to the cluster whose mean has the least squared Euclidean distance; this is intuitively the nearest mean:

$$S_i^{(t)} = \left\{ x_p : \left\| x_p - m_i^{(t)} \right\|^2 \leq \left\| x_p - m_j^{(t)} \right\|^2 \forall j, 1 \leq j \leq k \right\}$$

where each x_p is assigned to exactly one $S^{(t)}$, even if it could be assigned to two or more of them.

Update step: Calculate the new means of the observations in the new clusters.

$$m_i^{(t+1)} = \frac{1}{\left| S_i^{(t)} \right|} \sum_{x_j \in S_i^{(t)}} x_j$$

The algorithm converges when there is no change in the means.

4.1.2 Elbow method

In unsupervised learning, it is difficult to determine the number of classes since the data is unlabeled. One way to determine the number of classes is elbow method.We run the KMeans algorithm for various values of k and plot the inertia of each value of k. From the plot choose the optimum value of k [3].

4.1.3 Testing

After training the model, it is tested with an unseen dataset. The accuracy and loss are calculated using a library provided by sklearn called the metrics. In metrics, the accuracy score function is used to compute the accuracy of the test dataset.

4.1.4 Results

The accuracy is computed with various values of k. The table shows few accuracies of the model for respective k values.

Number of clusters (k)	Accuracy
9	52.175%
10	57.478%
11	58.647%

Table 4.1: Number of clusters and respective accuracy

```
[[3407  200    0    0  177  573 1606    2   26    9]
 [ 237 5416    0    0   65  151  131    0    0    0]
 [ 115    9    0    0 3536  504 1793    1   32   10]
 [1684 3208    0    0   51  507  542    0    7    1]
 [ 878  153    0    0 3604  248 1081    0   29    7]
 [   2    1    0    0    0 3897   30 1455   11  604]
 [1054   60    0    0 1991  759 2089    6   21   20]
 [   0    0    0    0    0  547    0 4856    0  597]
 [  22   28    0    0  418  467  297  193 2561 2014]
 [   4    3    0    0    1  226   39  252    0 5475]]
```

Figure 4.1.1: Confusion matrix for k = 9

```
[[3412  174  173    0   39  434 1719    2   47    0]
 [ 236 5407   28    0   44  127  156    0    2    0]
 [  84    8 1891    0 1856  424 1690    1   46    0]
 [1706 3139   17    0   54  408  664    0   11    1]
 [ 711  148  944    0 3026  194  938    0   39    0]
 [   3    1    0    0    0 4006   52 1431   22  485]
 [1036   49  952    0 1190  629 2074    6   62    2]
 [   0    0    0    0    0  650    1 5056    6  287]
 [  21   20  286    0   36  423  327  289 4590    8]
 [   8    3    1    0   10  199   62  684    8 5025]]
```

Figure 4.1.2: Confusion matrix for k = 10

```
[[2934   77    0  676  138  563 1557    2   53    0]
 [   7 5267    0  393   66  138  126    0    3    0]
 [  36    4    0  106 3522  491 1784    1   55    1]
 [  67 2310    0 2535   66  486  526    0   10    0]
 [   6   67    0  974 3649  243 1017    0   44    0]
 [   0    0    0    3    0 3801   32 1409   17  738]
 [ 639   29    0  483 1946  738 2077    6   81    1]
 [   0    0    0    0    0  524    0 4664    1  811]
 [   6   18    0   34  266  482  243  237 4643   71]
 [   0    1    0    6    1  170   35  165    4 5618]]
```

Figure 4.1.3: Confusion matrix for k = 11

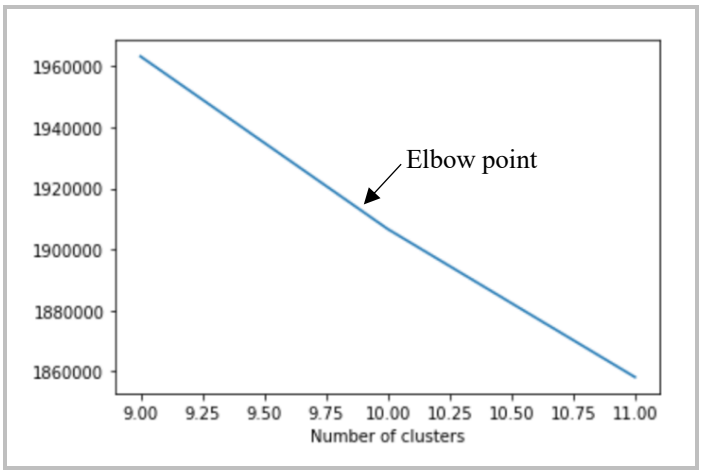

Figure 4.1.4: Elbow method

The testing accuracy for the KMeans on 10,000 images is **56.03%**.

4.2 Task 2

The task 2 is to design and build an auto encoder for condensing the images of Fashion MNIST dataset. The reduced dimensionality is used as input to KMeans for clustering.

4.2.1 Autoencoder

The autoencoder is an unsupervised learning algorithm that takes an image as the input and encodes it into fewer bits and reconstructs the original image using the fewer bits. This technique is very useful when the dataset has no labels. Also, it reduces the dimensionality of the data space. The compression in autoencoders is achieved by training the network for a period of time and as it learns it tries to best represent the input image at the bottleneck [4]. The autoencoder uses nonlinear transformations. The mechanism of autoencoding is based on 3 steps [5]:

Encoder: In this phase, the autoencoder compresses the data to delete unnecessary information.

Decoder: In this step the algorithm learns how to reconstruct the original input, keeping it as similar as possible to the original input.

Loss function: It helps the process to correct the error produced by the decoder.

This is a recursive process until the error between the original image and the reconstructed image is minimized. The error is called reconstruction error.

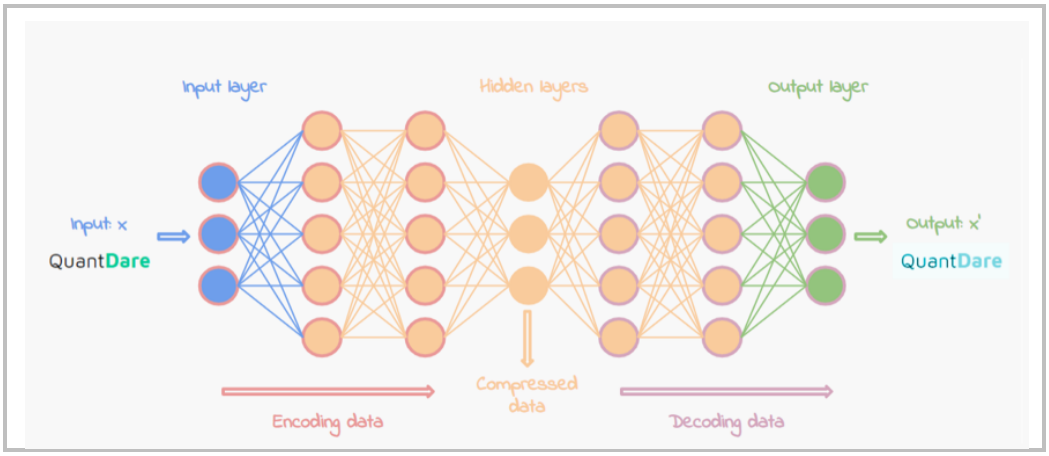

Figure 4.2.1: Autoencoder [5]

4.2.2 Convolutional autoencoder

Each encoder consists of various convolutional layers responsible for generating feature maps. To introduce non-linearity, a ReLU layer is used as an efficient activation function. Then, a max-pooling window with a non-overlapping stride produces sub-sampled feature maps. After using several steps of convolutional layers, a low-resolution feature map is achieved, which is up-sampled in the decoder part. The problem lies with the decoder learning to deconvolve or decode the low-resolution map. Thus, boundary information of the encoder feature maps is stored before sub-sampling and used in the decoder step. Badrinarayanan et al. (2015) showed that it is sufficient to store only the max-pooling indices. Since the maximum feature values exist in each encoder feature maps [6].

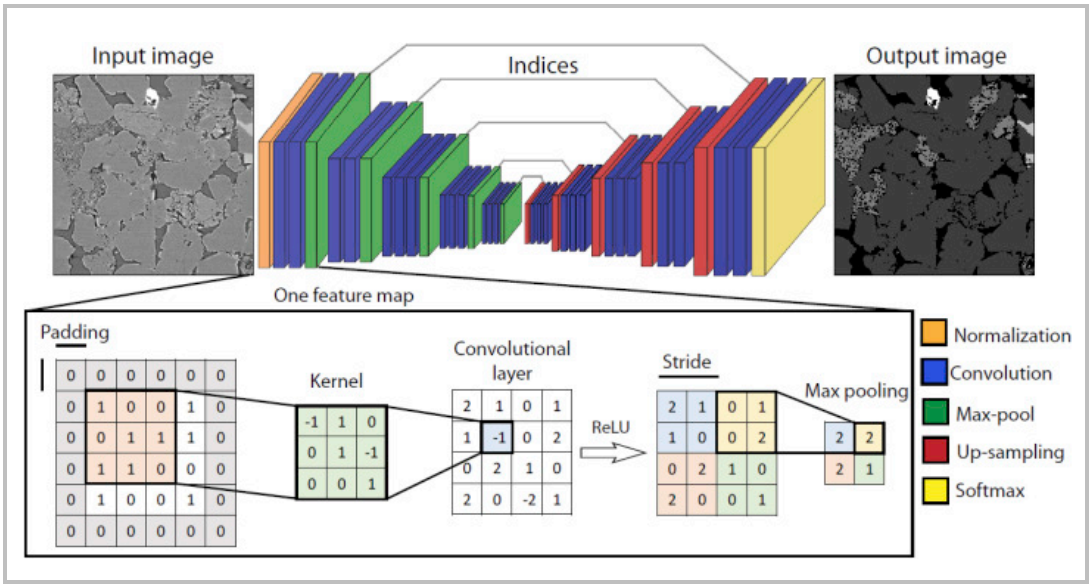

Figure 4.2.2: Convolutional Autoencoder [6]

4.2.3 Architecture

In this task, we have used a convolutional autoencoder with the following architecture.

The encoder comprises of 2 convolution layers with 15 and 8 filters respectively. Each convolution layer is followed by a max pool layer and we drop 20% data at each layer to maintain generality of the model. The activation function used is ReLU.

Similarly, the decoder comprises of 2 convolution layers with 8 and 15 filters respectively. Each convolution layer is followed by a up scaling layer and we drop 20% data at each layer to maintain generality of the model. The activation function used is ReLU.

Kernel size used is (3, 3) along with padding to retain the data.

Firstly, the autoencoder is compiled and trained for some 8 epochs. Once the autoencoder is trained and the loss is minimized, the KMeans is applied for clustering using the encoded data.

Layer (type)	Output Shape	Param #
conv2d_11 (Conv2D)	(None, 28, 28, 15)	150
max_pooling2d_5 (MaxPooling2	(None, 14, 14, 15)	0
dropout_9 (Dropout)	(None, 14, 14, 15)	0
conv2d_12 (Conv2D)	(None, 14, 14, 8)	1088
max_pooling2d_6 (MaxPooling2	(None, 7, 7, 8)	0
dropout_10 (Dropout)	(None, 7, 7, 8)	0
conv2d_13 (Conv2D)	(None, 7, 7, 8)	584
up_sampling2d_5 (UpSampling2	(None, 14, 14, 8)	0
dropout_11 (Dropout)	(None, 14, 14, 8)	0
conv2d_14 (Conv2D)	(None, 14, 14, 15)	1095
up_sampling2d_6 (UpSampling2	(None, 28, 28, 15)	0
dropout_12 (Dropout)	(None, 28, 28, 15)	0
conv2d_15 (Conv2D)	(None, 28, 28, 1)	136

Total params: 3,053
Trainable params: 3,053
Non-trainable params: 0

Figure 4.2.3: Model summary

4.2.3 Results

The testing accuracy for the KMeans on 10,000 encoded images is **53.12%**. From the graph below we can conclude that the optimum number of epochs is 10.

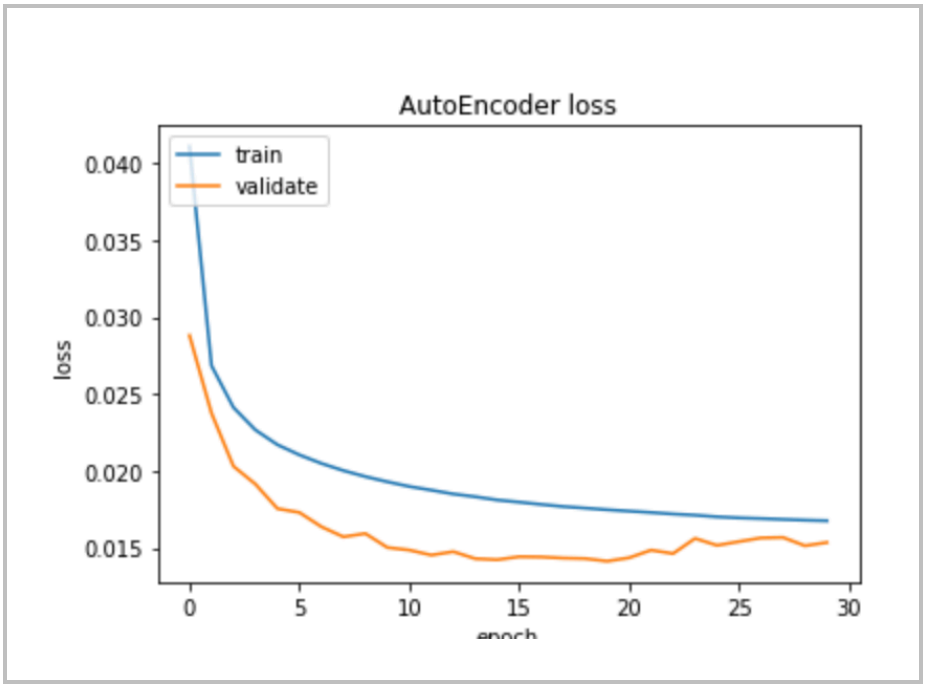

Figure 4.2.4: Autoencoder training and validation loss v/s epochs

```
Confusion matrix :
[[ 604    0    0  137   20    5  635    2   64    0]
 [   0 1124    0   56   11    0  292    0   12    0]
 [   5    0    0   24  649    4  450    0  358    0]
 [  24   17    0  844   18    0  531    0   37    0]
 [   0    2    0  180  742    0  254    0  322    0]
 [   0    0    0    3    0 1202   22  170   11   67]
 [ 131    0    0   90  316   11  657    0  265    0]
 [   0    0    0    0    0  545    0  947    0   50]
 [   0    0    0   18   32  176  152   79 1060    3]
 [   0    0    0    2    2  147   20  201   16 1182]]
```

Figure 4.2.5: Confusion Matrix

4.3 Task 3

The task 3 is to design and build an auto encoder for condensing the images of Fashion MNIST dataset. The reduced dimensionality is used as input to Gaussian Mixture Model for clustering.

4.3.1 Data condensation

The dimensionality of the data is reduced using the same technique as in task 2. We use convolutional autoencoder for feature extraction as mentioned section 4.2.2.

4.3.2 Gaussian Mixture Model (GMM)

GMM is a tool used for data clustering. It differs from the popular KMeans by the fact that KMeans is hard clustering i.e. each data point is assigned to only one cluster, whereas GMM is a soft clustering, a probabilistic assignment i.e. each data point is assigned to multiple clusters based on the probability. GMM is a mixture of Gaussian functions each identified by $k \in \{1, \ldots, K\}$ where K is number of clusters of datasets, hence the name Gaussian Mixture [7].

Each gaussian k is comprised of a mean μ that defines the center, a covariance $\sum$ that defines the width and a mixing probability π that defines the size of the Gaussian function. The mixing coefficients are probabilities hence:

$$\sum_{k=1}^{K} \pi_k = 1$$

Gaussian density function is given by:

$$\mathcal{N}(x|\mu,\Sigma) = \frac{1}{(2\pi)^{\frac{D}{2}}\,|\Sigma|^{\frac{1}{2}}}\, exp\left(-\frac{1}{2}\,(X-\mu)^T\Sigma^{-1}(X-\mu)\right)$$

The probability of data given the Gaussian is given by:

$$p(x_n|z) = \prod_{k=1}^{K} \mathcal{N}(x_n|\mu_k,\Sigma_k)^{z_k}$$

Finding these parameters is difficult, hence we use an iterative method called Expectation-Maximization or EM algorithm where:

$$\theta = \{\pi,\mu,\Sigma\}$$

In the maximization step the parameters are revised:

$$\theta^* = \arg max_\theta Q(\theta^*,\theta)$$

$$\pi_k^* = \frac{\sum_{n=1}^{N} \gamma(z_{nk})}{N}$$

$$\mu_k^* = \frac{\sum_{n=1}^{N} \gamma(z_{nk}) \mathbf{x}_n}{\sum_{n=1}^{N} \gamma(z_{nk})}, \qquad \Sigma_k^* = \frac{\sum_{n=1}^{N} \gamma(z_{nk})(\mathbf{x}_n - \mu_k)(\mathbf{x}_n - \mu_k)^T}{\sum_{n=1}^{N} \gamma(z_{nk})}$$

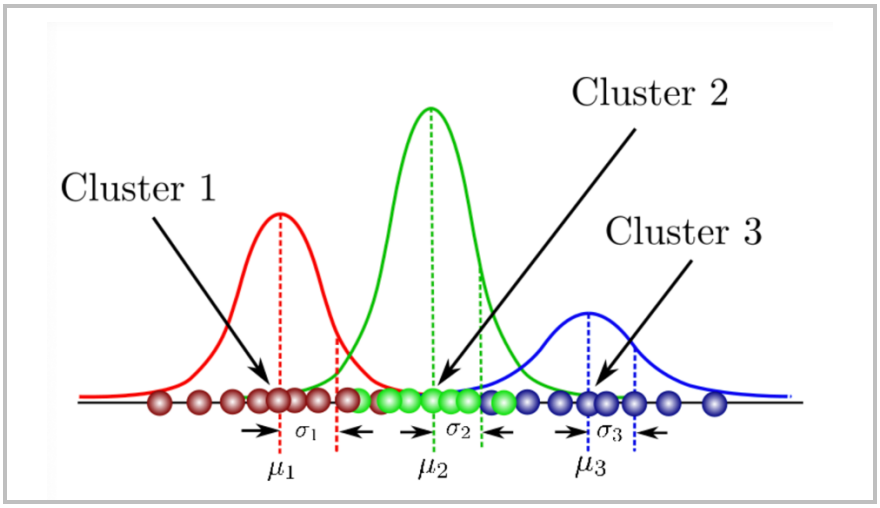

Figure 4.3.1: Gaussian Mixtures [7]

4.3.3 Architecture

In this task, we have used a convolutional autoencoder with the following architecture.

The encoder comprises of 2 convolution layers with 15 and 8 filters respectively. Each convolution layer is followed by a max pool layer and we drop 20% data at each layer to maintain generality of the model. The activation function used is ReLU.

Similarly, the decoder comprises of 2 convolution layers with 8 and 15 filters respectively. Each convolution layer is followed by a up scaling layer and we drop 20% data at each layer to maintain generality of the model. The activation function used is ReLU.

Kernel size used is (3, 3) along with padding to retain the data.

Firstly, the autoencoder is compiled and trained for some 8 epochs. Once the autoencoder is trained and the loss is minimized, the GMM is applied for clustering using the encoded data.

Layer (type)	Output Shape	Param #
conv2d_11 (Conv2D)	(None, 28, 28, 15)	150
max_pooling2d_5 (MaxPooling2	(None, 14, 14, 15)	0
dropout_9 (Dropout)	(None, 14, 14, 15)	0
conv2d_12 (Conv2D)	(None, 14, 14, 8)	1088
max_pooling2d_6 (MaxPooling2	(None, 7, 7, 8)	0
dropout_10 (Dropout)	(None, 7, 7, 8)	0
conv2d_13 (Conv2D)	(None, 7, 7, 8)	584
up_sampling2d_5 (UpSampling2	(None, 14, 14, 8)	0
dropout_11 (Dropout)	(None, 14, 14, 8)	0
conv2d_14 (Conv2D)	(None, 14, 14, 15)	1095
up_sampling2d_6 (UpSampling2	(None, 28, 28, 15)	0
dropout_12 (Dropout)	(None, 28, 28, 15)	0
conv2d_15 (Conv2D)	(None, 28, 28, 1)	136

Total params: 3,053
Trainable params: 3,053
Non-trainable params: 0

Figure 4.3.1: Model summary

4.3.4 Results

The testing accuracy for the Gaussian Mixture Model on 10,000 encoded images is **60.46%**. From the graph below we can conclude that the optimum number of epochs is 10.

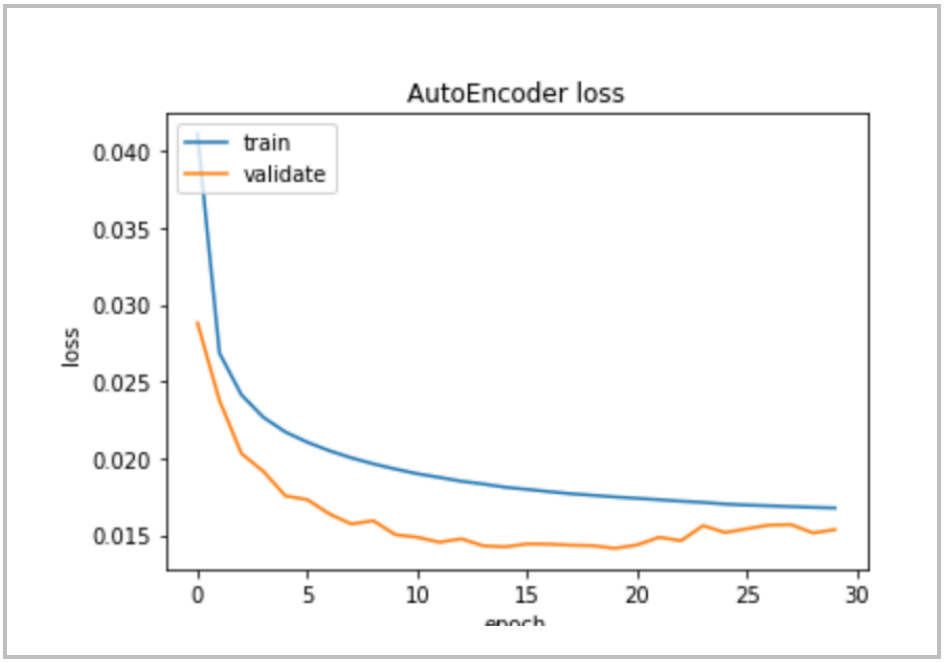

Figure 4.3.2: Autoencoder training and validation loss v/s epochs

```
Confusion matrix :
[[1032   26   40  186    0    6  150    1   25    1]
 [   6 1357   11   95    0    1   25    0    0    0]
 [  21    0 1209   32    0    3  182    0   43    0]
 [ 259  400   33  732    0    2   40    0    5    0]
 [ 140    5 1124  148    0    0   69    0   14    0]
 [   0    0    0    2    0  548   26  800    3   96]
 [ 307   10  716  108    0    4  278    0   47    0]
 [   0    0    0    0    0   30    0 1402    0  110]
 [   4    0   11   78    0   70  123   79 1147    8]
 [   1    0    0    2    0   34   15  154    0 1364]]
```

Figure 4.3.3: Confusion Matrix

6 Conclusion

Number of clusters, number of epochs, number of layers, number of filters are few hyper parameters used to tune the model and improve the accuracy. Either the hyper parameters can be manually tuned by trial error method or the process can be automated. For this problem the hyper parameters are manually trained and the corresponding finding are provided in the form of graphs in the result sections of each classifiers designed.

The naïve KMeans provides a test accuracy of **56.03%.**

The accuracy obtained by KMeans on the encoded data of autoencoder is **55.75%.**

The accuracy obtained by KMeans on the encoded data of autoencoder is **60.46%.**

References

[1] https://www.mathworks.com/discovery/unsupervised-learning.html

[2] https://en.wikipedia.org/wiki/K-means_clustering

[3] https://www.oreilly.com/library/view/statistics-for-machine/

[4] https://www.datacamp.com/community/tutorials/autoencoder-keras-tutorial

[5] https://quantdare.com/outliers-detection-with-autoencoder-neural-network/

[6] https://www.sciencedirect.com/science/article/pii/S0098300418303911

[7] https://towardsdatascience.com/gaussian-mixture-models-explained-6986aaf5a95