

# Clustering Analysis using K-Means and Gaussian Mixture Models

---

**Upasana Ghosh**

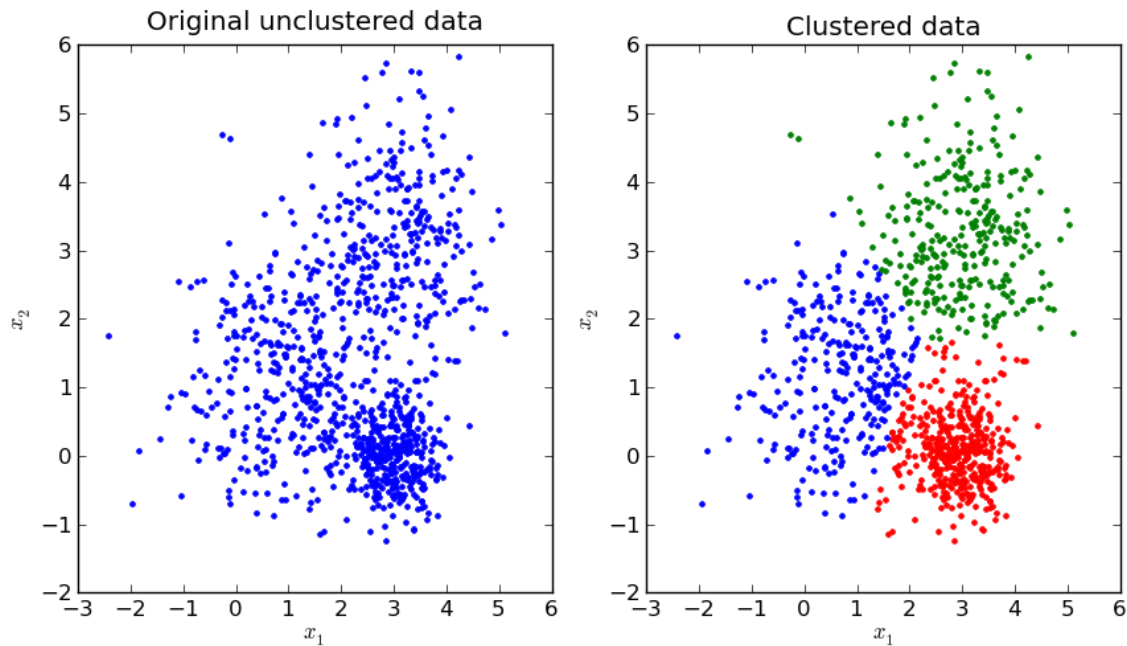
Department of Computer Science  
University at Buffalo  
Buffalo, NY 14214  
upasanag@buffalo.edu

## Abstract:

The purpose of this project is to perform cluster analysis on a dataset using unsupervised learning techniques. Cluster analysis is one of the unsupervised machine learning techniques which doesn't require labeled data. We are provided with the Fashion-MNIST dataset. Fashion-MNIST is a dataset of Zalando's article images, consisting of a total 70,000 examples. Each example is a 28x28 grayscale image. We need to cluster the images into ten different clusters using the K-Means Clustering algorithm. The K-Means algorithm is implemented using the functionalities provided by the Sklearn library. As the K-Means algorithm fails for higher dimensional datasets, in the second approach we implement an Auto-Encoder to compress the dataset and then apply the K-Means clustering using Keras and Sklearn library. In the third approach, we implement an Auto-Encoder based Gaussian Mixture Model as our clustering model to cluster the condensed representation of the dataset using Keras and Sklearn library. The Confusion Matrix and the Classification Report has been used to measure the accuracy of each of the Clustering Models.

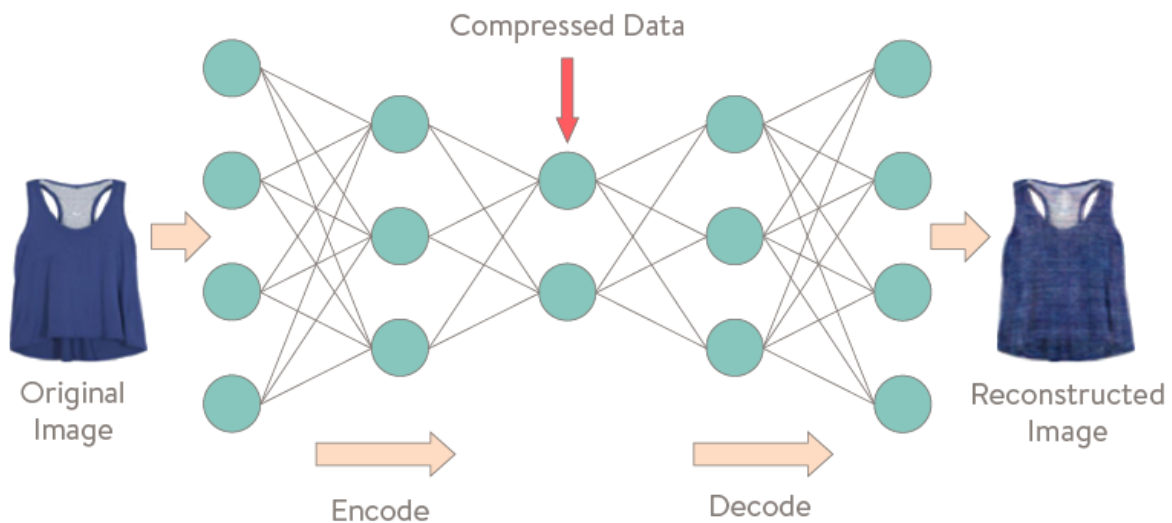
## Introduction:

Unsupervised learning is a machine learning technique, where we do not need to supervise the model. Instead, we allow the model to work on its own to discover information. Unsupervised learning mainly deals with unlabeled data. Clustering is an important concept when it comes to unsupervised learning. It mainly deals with finding a structure or pattern in a collection of uncategorized data. The K-Means algorithm clusters data by trying to separate the dataset in 'K' groups of equal variances, by minimizing a criterion known as the inertia or within-cluster sum-of-squares. This algorithm requires the number of clusters (denoted by the letter 'K') to be specified beforehand.



The K-Means algorithm aims to choose cluster centroids that minimizes the Euclidean distance between the data samples in a given cluster. This Euclidean distance tends to become inflated when the data belong to very high dimensional spaces. To alleviate this problem and speed up computations, a dimensionality reduction algorithm such as the Principal Component Analysis (PCA) or Auto-encoder can be run on the data prior to the K-means clustering.

We have implemented an Auto-encoder to handle this issue for this project. ‘Autoencoding’ is a data compression algorithm where the compression and decompression functions are data-specific, lossy and learned automatically from examples rather than engineered by a human. An Autoencoder uses compression and decompression functions which are implemented using neural networks:

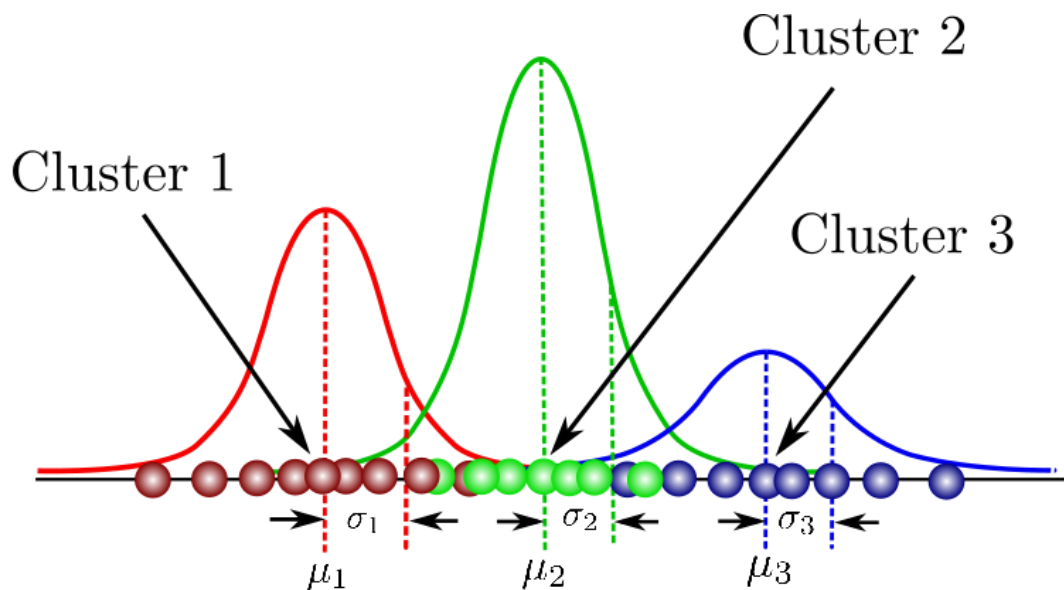


Building an autoencoder requires implementing three components:

1. An encoding function (Encoder)
2. A decoding function (Decoder)
3. A distance function (or a 'Loss' function) that calculates the amount of information loss between the compressed and the decompressed representation of our data.

The encoder and decoder are generally chosen to be parametric functions (typically neural networks) and needs to be differentiable with respect to the distance function, so that the parameters of the encoding/decoding functions can be optimized to minimize the reconstruction loss, using Stochastic Gradient Descent.

K-Means assumes spherical shape of the data while clustering hence fails to cluster appropriately when data takes on different shapes. To handle such situations, we use the Gaussian Mixture Model. A Gaussian mixture model is a probabilistic model that assumes that all the data points are generated from a mixture of a finite number of Gaussian distributions with unknown parameters. The Gaussian Mixture object implements the expectation-maximization (EM) algorithm for fitting a mixture-of-Gaussian models to represent the data distribution:



It can also draw confidence ellipsoids for multivariate models and compute the Bayesian Information Criterion to assess the number of clusters in the data.

### Dataset:

We are using the Fashion-MNIST dataset for training and testing our clustering models. The 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. Each example is a 28x28 grayscale image. Each image is 28 pixels in height and

28 pixels in width, for a total of 784 pixels in total. Each pixel has a single pixel-value associated with it, indicating the lightness or darkness of that pixel, with higher numbers meaning darker. This pixel-value is an integer between 0 and 255. The training and the test data sets have 785 columns. The first column consists of the class labels (Fig 3 below) and represents the article of clothing. The rest of the columns contain the pixel-values of the associated image. We would be assigning each training and test example to one of the clusters based on the labels.

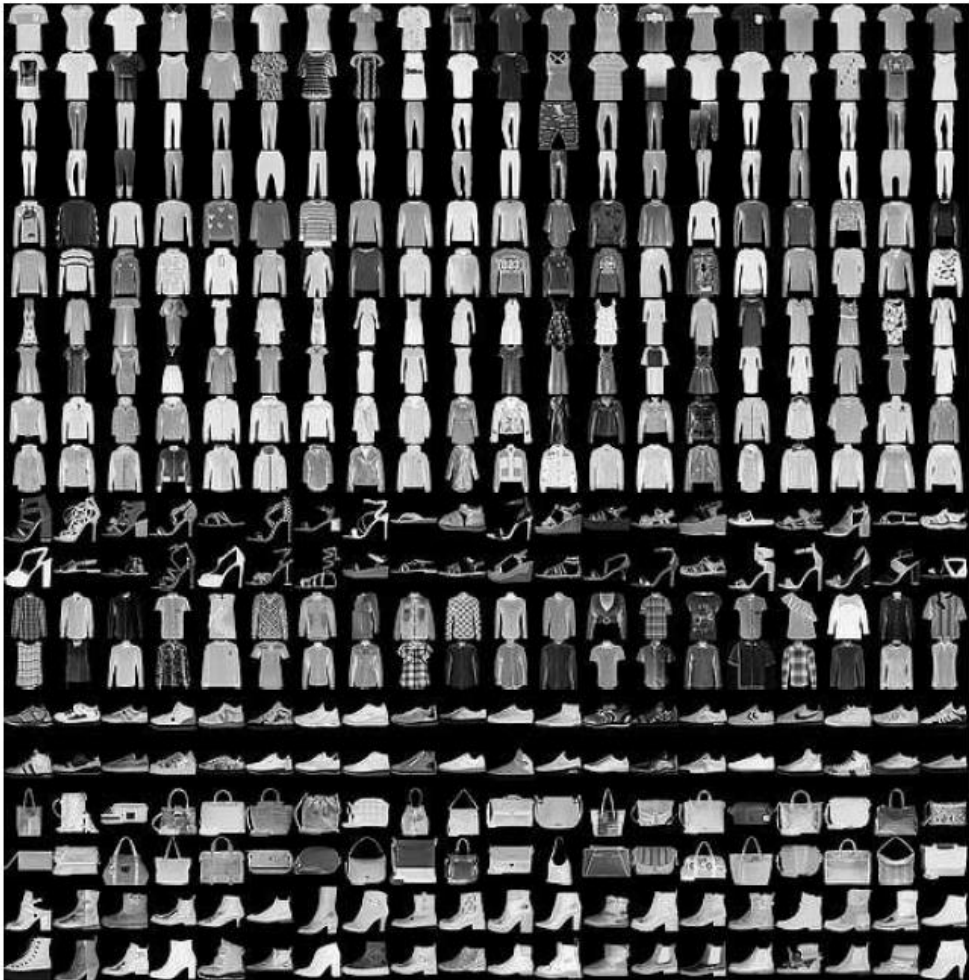
Label	Description	Examples
0	T-Shirt/Top	
1	Trouser	
2	Pullover	
3	Dress	
4	Coat	
5	Sandals	
6	Shirt	
7	Sneaker	
8	Bag	
9	Ankle boots	

Fig 3: Example of MNIST Fashion Dataset

### Preprocessing:

The preprocessing steps that were performed on the dataset for this project are as follows:

1. We extract the feature vector and the labels from the given dataset and sample the dataset randomly into training and testing data sets.

2. The feature vectors `X_train` and `X_test` are normalized by dividing each of the values in the dataset by 255 as the given input dataset consists of only images with pixels ranging from 0 to 255. This is done to maintain a uniform mean and standard deviation throughout the pixel distribution.
3. We have also rescaled the training and the testing data sets to range between 0 and 1 by using the maximum pixel value available in the training and the testing data sets.

## Architecture:

- **Simple K-Means Clustering Model**

The K-means algorithm divides a set of samples into disjoint clusters based on the mean of the samples in the cluster, commonly termed as the cluster centroids. It scales well to large number of samples and has been used across a large range of application areas in many different fields. In this project, we have implemented this simple unsupervised learning clustering algorithm with the help of the `KMeans` method provided by the `sklearn.cluster` library. We have trained our model by taking ten epochs or ten clusters in this case. The `kmeans.fit` method has been used for training the model using the training data set and the `kmeans.fit_predict` method has been used to predict the clusters for the test data. The Sum of Squared Errors (SSE) calculated at the number of clusters level shows that the error is least at the `k=10` clusters:

```
SSE for #cluster = 1 is 4092975.6596677564
SSE for #cluster = 2 is 3233032.0676284176
SSE for #cluster = 3 is 2766659.5236280463
SSE for #cluster = 4 is 2505225.873973058
SSE for #cluster = 5 is 2352605.66958883
SSE for #cluster = 6 is 2202642.5083292704
SSE for #cluster = 7 is 2099373.5016917875
SSE for #cluster = 8 is 2027691.0161877794
SSE for #cluster = 9 is 1963211.0586392875
SSE for #cluster = 10 is 1906652.6329259197
```

This is the level where we stop training our model and feed the test data to verify if our model can predict the correct labels.

- **Auto Encoder:**

An autoencoder is an unsupervised machine learning algorithm that takes an image as input and tries to reconstruct it using fewer number of bits from the bottleneck also known as latent space. The image is majorly compressed at the bottleneck. Autoencoders are similar to dimensionality reduction techniques like Principal Component Analysis (PCA). They project the data from a

higher dimension to a lower dimension using linear transformation and try to preserve the important features of the data while removing the non-essential parts. A simple Auto-encoder primarily has the following three components:

1. An Encoder
2. A Decoder
3. A distance function (or a 'Loss' function) that calculates the amount of information loss between the compressed and the decompressed representation of our data.

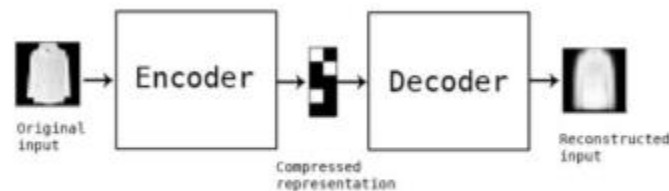


Figure 2: AutoEncoder

In this project, we have implemented a Convolutional Auto-encoder. This Auto-encoder makes use of the convolutional neural networks to implement the encoder and the decoder block. The encoder block has three hidden layers while the decoder has 4 hidden layers. The encoder function uses a ReLU activation function, while the decoder function uses a sigmoid activation function:

```
# Encoder Layers
autoencoder.add(Conv2D(16, (3, 3), activation='relu', padding='same', input_shape=X_train.shape[1:]))
autoencoder.add(MaxPooling2D((2, 2), padding='same'))
autoencoder.add(Conv2D(8, (3, 3), activation='relu', padding='same'))
autoencoder.add(MaxPooling2D((2, 2), padding='same'))
autoencoder.add(Conv2D(8, (3, 3), strides=(2,2), activation='relu', padding='same'))

# Flatten encoding for visualization
autoencoder.add(Flatten())
autoencoder.add(Reshape((4, 4, 8)))

# Decoder Layers
autoencoder.add(Conv2D(8, (3, 3), activation='relu', padding='same'))
autoencoder.add(UpSampling2D((2, 2)))
autoencoder.add(Conv2D(8, (3, 3), activation='relu', padding='same'))
autoencoder.add(UpSampling2D((2, 2)))
autoencoder.add(Conv2D(16, (3, 3), activation='relu'))
autoencoder.add(UpSampling2D((2, 2)))
autoencoder.add(Conv2D(1, (3, 3), activation='sigmoid', padding='same'))
```

Compression factor: 0.875  
Model: "sequential\_6"

Layer (type)	Output Shape	Param #
conv2d_29 (Conv2D)	(None, 28, 28, 16)	160
max_pooling2d_11 (MaxPooling)	(None, 14, 14, 16)	0
conv2d_30 (Conv2D)	(None, 14, 14, 8)	1160
max_pooling2d_12 (MaxPooling)	(None, 7, 7, 8)	0
conv2d_31 (Conv2D)	(None, 4, 4, 8)	584
flatten_6 (Flatten)	(None, 128)	0
reshape_5 (Reshape)	(None, 4, 4, 8)	0
conv2d_32 (Conv2D)	(None, 4, 4, 8)	584
up_sampling2d_10 (UpSampling)	(None, 8, 8, 8)	0
conv2d_33 (Conv2D)	(None, 8, 8, 8)	584
up_sampling2d_11 (UpSampling)	(None, 16, 16, 8)	0
conv2d_34 (Conv2D)	(None, 14, 14, 16)	1168
up_sampling2d_12 (UpSampling)	(None, 28, 28, 16)	0
conv2d_35 (Conv2D)	(None, 28, 28, 1)	145
Total params: 4,385		
Trainable params: 4,385		
Non-trainable params: 0		

We have used 'adam' as the optimizer and 'binary\_crossentropy' as the loss function for the Autoencoder. The auto-encoder has been trained using the training data set by taking the epoch as 50 and the batch size as 128. The test data set has been used for validation.

- [Auto-Encoder based K-Means Clustering Model](#)

The K-means algorithm aims to choose centroids that minimize the inertia, or within-cluster sum-of-squares criterion:

$$\sum_{i=0}^n \min_{\mu_j \in C} (||x_i - \mu_j||^2)$$

Inertia can be defined as the measure of how internally coherent the clusters are. Inertia is not a normalized metric and in very high dimensional spaces, the Euclidean distances tend to become inflated (this is an instance of the 'Curse of dimensionality'). To alleviate this problem and speed up computations, we have encoded the samples using the Auto-encoder prior to the K-means clustering.



The Auto-encoder based K-Means Clustering Model has been implemented by leveraging the same Kmeans() method provided by the sklearn.cluster library as we did for the simple K-Means model. For encoding the data before performing the K-Means cluster, we are extracting the encoder layer of the Auto-encoder. Passing the data through this layer provides us with the lower dimensional or the compressed data which is then passed to Kmeans() for the clustering. We notice that the performance of the K-Means algorithm improves relatively after compressing the data.

- **Auto-Encoder based Gaussian Mixture Model**

A Gaussian mixture model (GMM) is a category of probabilistic model which states that all generated data points are derived from a mixture of a finite Gaussian distributions that has no known parameters. K-Means algorithm fails during clustering data distributions that are non-spherical in nature. To handle such situations, we use GMM to cluster the data samples. Unlike K-Means, GMM performs soft clustering on the data sets and leverages multiple Gaussian distributions to cluster the given dataset. The Gaussian Mixture object implements the expectation-maximization (EM) algorithm for fitting mixture-of-Gaussian models. It can also draw confidence ellipsoids for multivariate models and compute the Bayesian Information Criterion to assess the number of clusters in the data.

In this project, we have used the GaussianMixture.fit method provided by Sklearn to help the model learn a Gaussian Mixture Model from the training dataset. The GaussianMixture.fit\_predict method has been used by the model to assign to each sample the Gaussian it mostly probably belongs to. To improve the performance of the GMM model, we have reduced the dimensions of the both the training and the test data sets using the encoder layer of the Auto-encoder.

We have used a confusion matrix to evaluate the model. The Confusion Matrix gives us the count of the True Positive (TP), True Negative (TN), False Positive (FP) and False Negative (FN) in the predicted output. We can define the TP, TN, FP and FN as follows:

		PREDICTED	
ACTUAL		Positive	Negative
	Positive	TP	FN
	Negative	FP	TN

Using these values, we calculate the Accuracy, Precision and Recall for the training, validation and testing data sets with the help of the following formulae:



$$\text{Accuracy} = \frac{TP+TN}{TP+TN+FP+FN}$$

$$\text{Precision} = \frac{TP}{TP+FP}$$

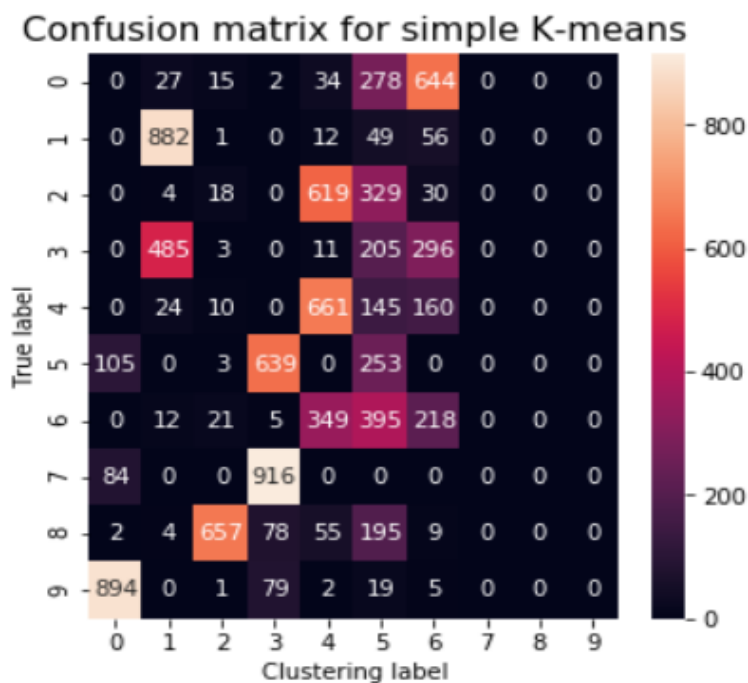
$$\text{Recall} = \frac{TP}{TP+FN}$$

The trend in the accuracy, precision and recall values show us whether we are tuning the hyperparameters correctly and finally help us judge if our model is trained enough to predict the correct output for an unknown dataset in the future. The 'normalized\_mutual\_info\_score' has been used to measure the accuracy of all the models.

## Results:

- Simple K-Means Clustering Model:

- Confusion Matrix:



- Accuracy:

SIMPLE K-MEANS MODEL ACCURACY: 0.5160993782137807

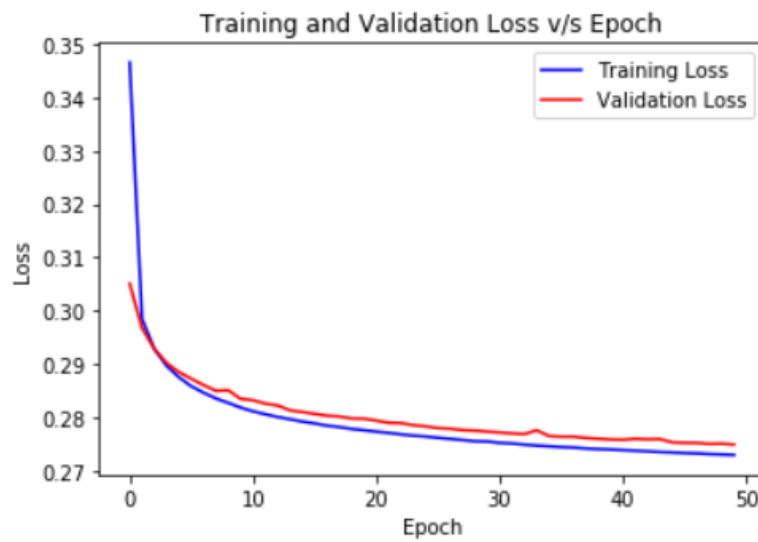
- **Classification Report:**

CLASSIFICATION REPORT

	precision	recall	f1-score	support
0	0.00	0.00	0.00	1000
1	0.61	0.88	0.72	1000
2	0.02	0.02	0.02	1000
3	0.00	0.00	0.00	1000
4	0.38	0.66	0.48	1000
5	0.14	0.25	0.18	1000
6	0.15	0.22	0.18	1000
7	0.00	0.00	0.00	1000
8	0.00	0.00	0.00	1000
9	0.00	0.00	0.00	1000
accuracy			0.20	10000
macro avg	0.13	0.20	0.16	10000
weighted avg	0.13	0.20	0.16	10000

- **Auto-Encoder:**

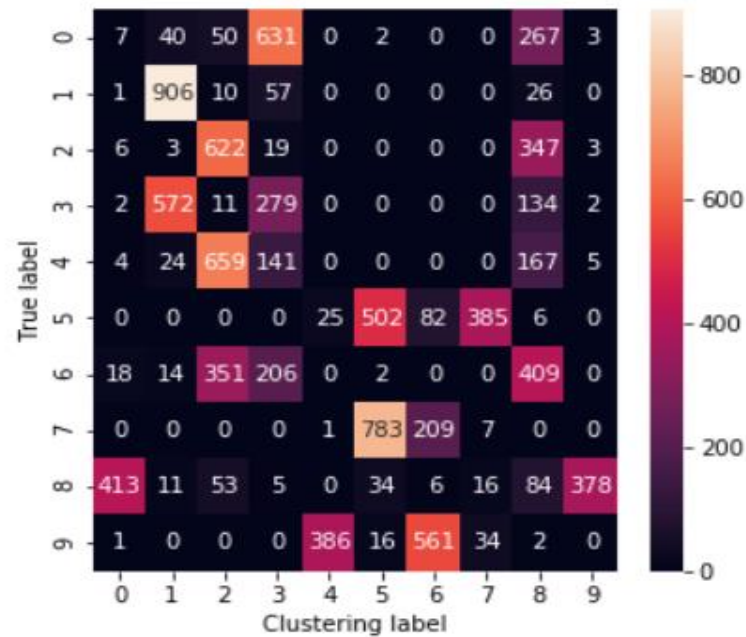
- **Loss v/s Epoch graph for Training and Validation Loss:**



- Auto-Encoder based K-Means Clustering Model:

- Confusion Matrix:

Confusion Matrix for Auto-encoder based K-Means model



- Accuracy:

AUTO-ENCODER BASED K-MEANS MODEL ACCURACY: 0.5461585839613244

- Classification Report:

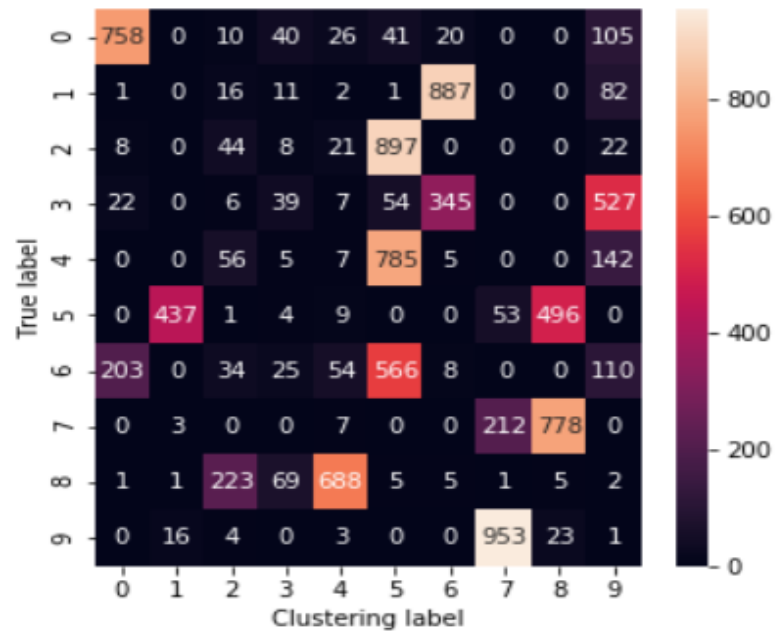
CLASSIFICATION REPORT

	precision	recall	f1-score	support
0	0.03	0.04	0.03	1000
1	0.08	0.09	0.08	1000
2	0.00	0.00	0.00	1000
3	0.04	0.04	0.04	1000
4	0.00	0.00	0.00	1000
5	0.00	0.00	0.00	1000
6	0.03	0.04	0.03	1000
7	0.00	0.00	0.00	1000
8	0.94	0.36	0.53	1000
9	0.65	0.60	0.62	1000
accuracy			0.12	10000
macro avg	0.18	0.12	0.13	10000
weighted avg	0.18	0.12	0.13	10000

- Auto-Encoder based Gaussian Mixture Model:

- Confusion Matrix:

Confusion Matrix for Auto-encoder based GMM model



- Accuracy:

AUTO-ENCODER BASED GMM MODEL ACCURACY: 0.5912614497142512

- Classification Report:

CLASSIFICATION REPORT

	precision	recall	f1-score	support
0	0.10	0.10	0.10	1000
1	0.00	0.00	0.00	1000
2	0.01	0.01	0.01	1000
3	0.00	0.00	0.00	1000
4	0.00	0.01	0.00	1000
5	0.00	0.00	0.00	1000
6	0.12	0.03	0.04	1000
7	0.00	0.00	0.00	1000
8	0.00	0.00	0.00	1000
9	0.64	0.90	0.75	1000
accuracy			0.10	10000
macro avg	0.09	0.10	0.09	10000
weighted avg	0.09	0.10	0.09	10000

## Conclusion:

This project to perform clustering analysis using K-Means, Gaussian Mixture Model and Auto-encoder provides us with an insight about the improvements on the clustering accuracy that each of the model has on the other. This gives us an opportunity to compare the performance of the different models when applied to the same dataset. From the Confusion matrix and the Classification report in our result, we can conclude the following:

- The performance of the model improves as we move from the simple K-Means model to the Auto-encoder based K-Means model. This is because the dimensions of the images in our datasets get reduced as we encode them using the Auto-encoder. This tells us that the performance of the simple K-Means algorithm suffers as the dimensions of the data increases to a great extent.
- K-Means assume the data clusters to be spherical in shape. This assumption hampers the performance of the model when the data distribution changes shape. When we move from K-Means to GMM, we observe that GMM resolves this issue by performing soft clustering unlike K-Means that performs hard clustering.

## Acknowledgement:

1. An Introduction to Statistical Learning by Gareth James, Daniela Witten, Trevor Hastie and Robert Tibshirani
2. <https://towardsdatascience.com/kmeans-clustering-for-classification-74b992405d0a>
3. <https://mubaris.com/posts/kmeans-clustering/>
4. <https://scikit-learn.org/stable/modules/generated/sklearn.cluster.KMeans.html>
5. <https://www.guru99.com/unsupervised-machine-learning.html>
6. <https://jakevdp.github.io/PythonDataScienceHandbook/05.11-k-means.html>
7. <https://www.techopedia.com/definition/30331/gaussian-mixture-model-gmm>