# Cluster analysis on fashion MNIST dataset using unsupervised learning

**Shwetasree Chowdhury**

Person Id: 50296995

Department of Computer Science

State University of New York at Buffalo

*shwetasr@buffalo.edu*

## Abstract

In this project, we perform cluster analysis on fashion MNIST dataset using unsupervised learning. Cluster analysis is one of the unsupervised machine learning technique which doesn't require labeled data. The fashion MNIST data as described below comprises of 70000 instances of data with each image being a 28 x 28 grayscale reproduction of any of the 10 items described below. The purpose of this project dictates us to cluster the same using differing clustering algorithms, mainly K-Means and Gaussian Mixture Model, with and without the help of an autoencoder

We will be taking three different approaches to test and verify the accuracy of the algorithms on the dataset using the Scipy and sklearns libraries.

1. Cluster the training and test dataset using kmeans algorithm.
2. Build an Auto-Encoder based K-Means clustering model to cluster the condensed representation of the unlabeled fashion MNIST dataset using Keras and Sklearns library.
3. Build an Auto-Encoder based Gaussian Mixture Model clustering model to cluster the condensed representation of the unlabeled fashion MNIST dataset using Keras and Sklearns library.

Finally, we shall be reporting the accuracy of each of these clustering algorithms.

**Problem Statement**

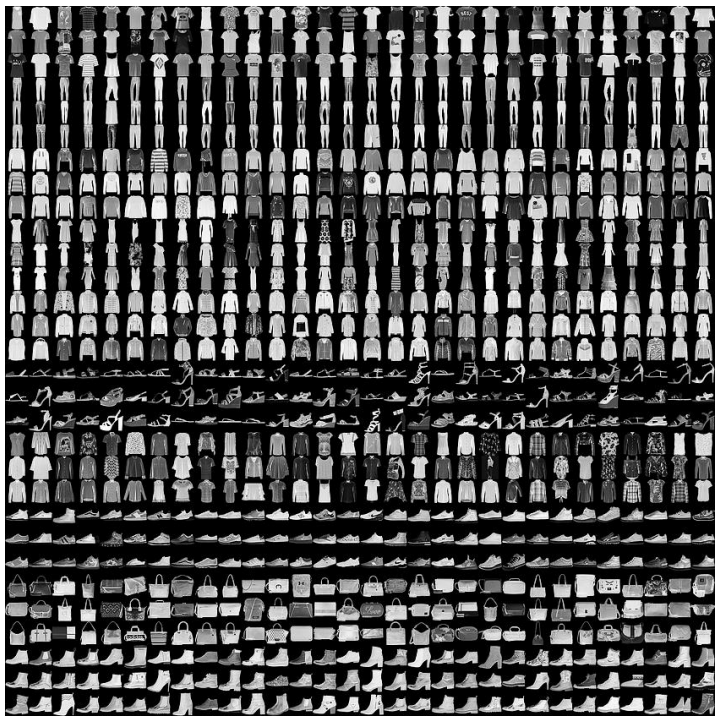
The fashion MNIST is a dataset comprising of 70000 data. Each image is a 28x28 grayscale reproduction, associated with a label from 10 classes. Labels so represented are noted in the next section. We aim to use kmeans and Gaussian Mixture Model- which are both clustering techniques in order to cluster the above dataset based on their likeliness. Initially, we simply use kmeans to cluster the dataset. However, we then parse the dataset through an autoencoder. Autoencoders are an unsupervised learning technique in which we leverage neural networks for the task of **representation learning**. Specifically, we'll design a neural network architecture such that we *impose a bottleneck in the network which forces a* **compressed** *knowledge representation of the original input.* After the compressed/ latent space representation of each image is formed, this is exploited to conduct kmeans clustering and GMM clustering on the same. This is done so as to gauge the performance of normal clustering on data v/s clustering on autoencoded/ compressed images of the same dataset.

# 1. Theory

## 1.1 Dataset Used

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, associated with a label from 10 classes. We intend Fashion-MNIST to serve as a direct **drop-in replacement** for the original MNIST dataset for benchmarking machine learning algorithms. It shares the same image size and structure of training and testing splits.

Here's an example how the data looks (*each class takes three-rows*):



| Label | Description |
| --- | --- |
| 0 | T-shirt/top |
| 1 | Trouser |
| 2 | Pullover |
| 3 | Dress |
| 4 | Coat |
| 5 | Sandal |
| 6 | Shirt |
| 7 | Sneaker |
| 8 | Bag |
| 9 | Ankle boot |

## 1.2 What is clustering / cluster analysis?

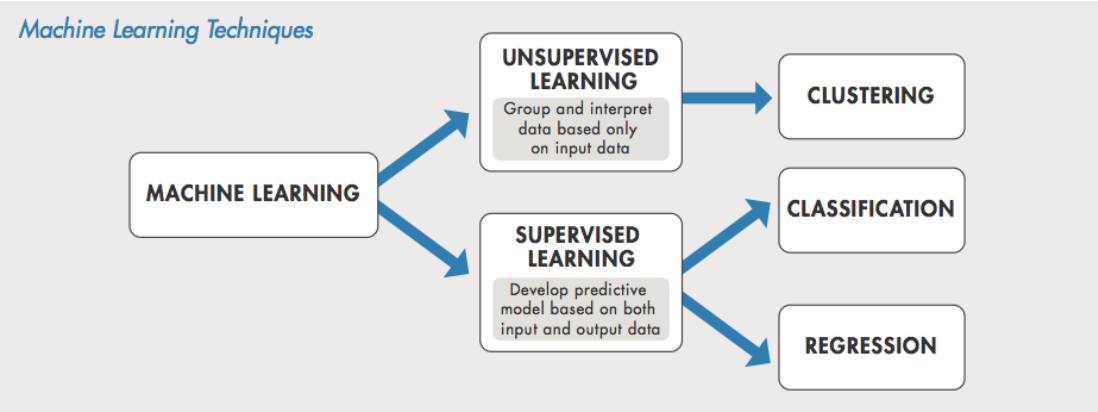
Cluster analysis divides data into groups (clusters) that are meaningful, useful, or both. If meaningful groups are the goal, then the clusters should capture the natural structure of the data. In some cases, however, cluster analysis is used for data summarization in order to reduce the size of the data.

Clusters are Classes, or conceptually meaningful groups of objects that share common characteristics, play an important role in how people analyse and describe the world. The greater the similarity within a group and the greater the difference between groups, the better or more distinct the clustering.

### How is clustering a form of unsupervised learning?

The goal for unsupervised learning is to model the underlying structure or distribution in the data in order to learn more about the data. Clustering is unsupervised since with clustering we try to group some data-points into several groups. We call those groups as clusters. So usually clustering does not look at target/labels instead it groups the data considering the similarities in the features. Therefore, clustering employs a similarity function to measure the similarity between two data-points (e.g. k means clustering measures the euclidean distance).

Unlike supervised learning, there is no correct answers/labels and there is no teacher. Algorithms are left to their own to discover and present the interesting structure in the data.




### Types of clustering

Clustering aims to find useful groups of objects (clusters), where usefulness is defined by the goals of the nature of the data. Some types of clusters are

- **Well-Separated:** A threshold is used to specify that all the objects in a cluster must be sufficiently close (or similar) to one another. This idealistic definition of a cluster is satisfied only when the data contains natural clusters that are quite far from each other.
- **Prototype-Based :** For data with continuous attributes, the prototype of a cluster is often a centroid, i.e., the average (mean) of all the points in the cluster.

- **Graph-Based:** If the data is represented as a graph, where the nodes are objects and the links represent connections among objects, then a cluster can be defined as a connected component; i.e., a group of objects that are connected to one another, but that have no connection to objects outside the group
- **Model-based:** In the model-based clustering approach, the data are viewed as coming from a mixture of probability distributions, each of which represents a different cluster. In other words, in model-based clustering, it is assumed that the data are generated by a mixture of probability distributions in which each component represents a different cluster. Thus a particular clustering method can be expected to work well when the data conform to the model.

**For the purpose of our project, we shall be working with two very common and famous clustering algorithms. One of which is prototype based and the other is model based.**

**K-means Clustering**

K-Means is one of the most popular clustering algorithms. It is an example of a prototype based clustering model. K-means stores "k" centroids that it uses to define clusters. A point is considered to be in a particular cluster if it is closer to that cluster's centroid than any other centroid.
It finds the best centroids by alternating between

1. Assigning data points to clusters based on the current centroids
2. Choosing centroids (points which are the center of a cluster) based on the current assignment of data points to clusters.

**<u>Algorithm:</u>**

1. Select K points as initial centroids
2. Repeat
   a. Form K clusters by assigning each point to its closest centroid
   b. Re-compute the centroid of each cluster
3. Until Centroids do not change

To assign a point to the closest centroid, we need a proximity measure that quantifies the notion of "closest" for the specific data under consideration. Euclidean (L2) distance is often used for data points in Euclidean space, while cosine similarity is more appropriate for documents.

For the purpose of our project, we have used a k-means++ approach (which is the default approach used in SKlearns library implementation)

**Algorithm 7.2** K-means++ initialization algorithm.

1: For the first centroid, pick one of the points at random.
2: **for** $i = 1$ to *number of trials* **do**
3: Compute the distance, $d(x)$, of each point to its closest centroid.
4: Assign each point a probability proportional to each point's $d(x)^2$.
5: Pick new centroid from the remaining points using the weighted probabilities.
6: **end for**

**Gaussian Mixture Model**

Mixture models view the data as a set of observations from a mixture of differing probability distribution. More formally, if there are K distributions and m objects, then we find out the probability of the i-th object in the j-th distribution, which is given by a weight.

If the objects are generated in an independent manner, then the probability of the entire set of objects is just the product of the probabilities of individual x. By using statistical methods- MLE, we can estimate the parameters of these distributions from the data and thus describe these distributions as clusters.

**Algorithm:**

1. Select an initial set of model parameters (As with K means, this can be done randomly)
2. Repeat
   a. **Expectation Step:** For each object calculate the probability that each object belongs to each distribution
   b. **Maximization Step:** Given the probabilities from the estimation step, find the new estimates of the parameters that maximize the expected likelihood.
3. Until parameter stop changing or until convergence is met.

Given a Gaussian mixture model, the goal is to maximize the likelihood function with respect to the parameters comprising the means and covariances of the components and the mixing coefficients).

1. Initialize the means μ , covariances ∑ and mixing coefficients π , and evaluate the initial value of the log likelihood.
2. E step: Evaluate the responsibilities using the current parameter values

$$\gamma_j(\mathbf{x}) = \frac{\pi_k \mathcal{N}(\mathbf{x} \mid \mu_k, \Sigma_k)}{\sum_{j=1}^{K} \pi_j \mathcal{N}(\mathbf{x} \mid \mu_j, \Sigma_j)}$$

3. M step: Re-estimate the parameters using the current responsibilities

$$\mu_j = \frac{\sum_{n=1}^{N} \gamma_j(x_n) x_n}{\sum_{n=1}^{N} \gamma_j(x_n)}$$

$$\Sigma_j = \frac{\sum_{n=1}^{N} \gamma_j(x_n)(x_n - \mu_j)(x_n - \mu_j)^T}{\sum_{n=1}^{N} \gamma_j(x_n)}$$

$$\pi_j = \frac{1}{N} \sum_{n=1}^{N} \gamma_j(x_n)$$

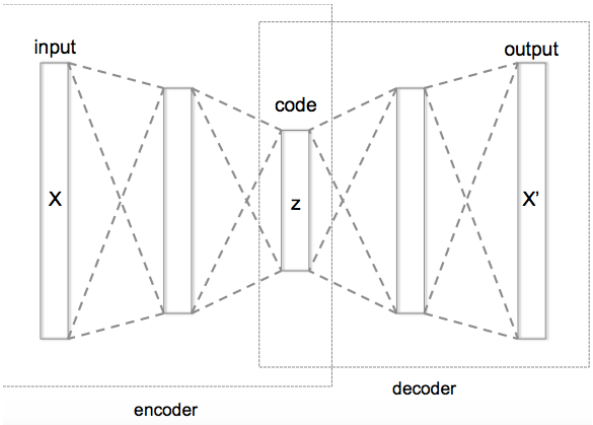
4. Evaluate log likelihood

$$\ln p(X \mid \mu, \Sigma, \pi) = \sum_{n=1}^{N} \ln \left\{ \sum_{k=1}^{K} \pi_k N(x_n \mid \mu_k, \Sigma_k) \right\}$$

If there is no convergence, return to 2.

**Autoencoders**

An **autoencoder** neural network is an unsupervised learning algorithm that applies backpropagation, setting the target values to be equal to the inputs.
An autoencoder is a neural network that is trained to attempt to copy its input to its output. Internally, it has a hidden layer h that describes a code used to represent the input. The network may be viewed as consisting of two parts: an encoder function h=f(x) and a decoder that produces a reconstruction r = g(h).
If an autoencoder succeeds in simply learning to set g(f(x)) =x everywhere, then it is not especially useful. Instead, autoencoders are designed to be unable to learn to copy perfectly. Usually they are restricted in ways that allow them to copy only approximately, and to copy only input that resembles the training data. Because the model is forced to prioritize which aspects of the input should be copied, it often learns useful properties of the data




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.

An auto encoder has two parts, namely the encoder and the decoder. While the encoder down-samples the image to a more reduced dimension, the decoder up-samples the image and reconstructs it to its former state. Auto encoders are useful for noise reduction and image reconstruction. For the purpose of this project, because we are dealing with 28 X 28 sized grayscale images which need to be reconstructed, we shall be using a convoluted autoencoder.

## 2. Implementation

In order to work on about any dataset, there are a few steps that we have to follow.

1. Importing datasets
2. Dataset normalization and preprocessing. Resizing 28 x 28 sized image into a 28 x 28 x 1 matrix to be fed into encoder.
3. Data partitioning into Training, Validation and Testing Sets
4. Run K-Means on Training Data. Subsequent calculation of accuracy using confusion matrix and comparison using NMI (Normalized Mutual Information Score)
5. Train convolutional auto encoder.
6. Run K-Means on latent space representation of encoded images. Subsequent calculation of accuracy using confusion matrix and comparison using NMI (Normalized Mutual Information Score)
7. Run GMM on latent space representation of encoded images. Subsequent calculation of accuracy using confusion matrix and comparison using NMI (Normalized Mutual Information Score)
8. Tuning of hyper-parameters.
9. Testing of each test-dataset against the same hyper-parameters to ascertain which tuning fits well.

### 2.1 Importing Datasets

The Fashion MNIST datasets can be imported using keras library by invoking *keras.datasets.fashion_mnist.load_*data(). It can also be directly downloaded from the official git page of fashion MNIST dataset as a .gz file and imported locally. We have used the latter approach. Four different datasets comprise of the fashion MNIST package. Train data, train label, test data and test label, i.e the label for the test and the train data are already bifurcated into separate datasets. Given below are the dataset sizes.

```
train_data shape: (60000, 28, 28) train_label shape: (60000,)
test_data shape: (10000, 28, 28) test_label shape: (10000,)
```

## 2.2 Dataset normalization and Preprocessing

The goal of normalization is to change the values of numeric columns in the dataset to a common scale, without distorting differences in the ranges of values, usually required for datasets with different kinds of values. This is required since if the features in the data set have values in different ranges, it takes the model a long time to converge and achieve the global/local minima. Hence, we do normalization to get the feature values in similar ranges and help the model to converge more quickly.

- For the purpose of this project, normalisation is done by dividing each image by 255 (range of pixels). The data is then concatenated together and shuffled using a common seed for the model to train well and learn on randomised data.
- The train data and test data are then reshaped into 28 x 28 x1 matrix in order to be fed into the auto encoder.

```
train_data = train_data.reshape(-1, 28,28, 1)
test_data = test_data.reshape(-1, 28,28, 1)
```

## 2.3 Data partitioning into Training, Validation and Testing

The data sets in this project are already separated into training and test data sets and labels.

**Training Data**: 48,000 rows, 784 labels | **Testing data**: 10000 rows, 784 labels | **Validation Data**: 12000 rows, 784 labels.

## 2.4 K- means implementation on Training Dataset

- We use the sklearns.cluster.Kmeans library to compute the K-means for the fashion mnist dataset, using all default values and specifying the no of clusters as 10. Method for initialization, defaults to 'k-means++': selects initial cluster centres for k-mean clustering in a smart way to speed up convergence

```
kmeans_data= train_data.reshape(len(train_data),784)
kmeans = KMeans(n_clusters=10, init='k-means++', max_iter=300, n_init=10, random_state=0)
kmeans_predicted_labels = kmeans.fit_predict(kmeans_data)
```

- Next we find the unique labels in each cluster. This is for us to understand that in the original dataset, we had 6000 images in each cluster, so as to draw a fair comparison (before and after the kmeans calculation).
- The scipy.stats.mode function calculates the mode of the array elements along the specified axis of the array. We simply find the indexes of the first cluster, for example, and then compare it with same indices of the predicted label from kmeans. The mode obtained from the original cluster becomes the final cluster label of the predicted list. This is reiteratively done for all clusters in the list
- The accuracy is then calculated using the confusion matrix.
- The normalized mutual information score is represented as follows

```
print(formatting.BOLD + "Normalized Mutual Information Score: " + formatting.END,
      NMI(train_label, predicted_kmean_train,average_method="arithmetic")*100)
```

Normalized Mutual Information (NMI) is a normalization of the Mutual Information (MI) score to scale the results between 0 (no mutual information) and 1 (perfect correlation). In this function, mutual information is normalized by some generalized mean of H(labels_true) and H(labels_pred)), defined by the average method. It is also a means to generate clustering accuracy for unsupervised learning. The NMI is calculated in order to base our accuracies produced.

- We then apply PCA on the original dataset (train data and test data) to reduce the 10 feature dimension to a 2 component representation for better illustration.
- The scatter plots from PCA are plotted for the original labels and the clustered labels.

### 2.5 Train using Auto Encoder network

As previously mentioned, an Autoencoder can be broken in to three parts:

**Encoder**: this part of the network compresses or downsamples the input into a fewer number of bits.

The space represented by these fewer number of bits is often called the **latent-space** or bottleneck. The bottleneck is also called the "maximum point of compression" since at this point the input is compressed the maximum

**Decoder**: this part of the network tries to reconstruct the input using only the encoding of the input.

- For the purpose of our project, we constructed a deep convolutional autoencoder with 14 layers- inclusive of the encoder and decoder. The encoder layers are of the following sizes. The image (28 x28 x 1) is first made thicker (28 x 28 x 64) and then gradually downsampled into 7 x 7 x 7 which is the encoded image. Likewise, the encoded image with dimension 7 x 7 x 7 is upsampled in the decoder and reconstructed as a 28 x 28 x 1 image at the end of the model. The function getautoencoder() is constructed for this purpose. We are also taking help from the keras library to build the auto encoder network.
- The model is then trained with batch size and epochs and other parameters specified
- Model is compiled with mean squared error as loss type and adam optimizer.
- The performance (loss and accuracy is plotted for each epoch)
- Finally, after successful training, we evaluate the test data on the trained autoencoder using the autoencoder.evaluate and autoencoder.predict functions
- The reconstructed images from the test data are then plotted so as to visually analyse the differences between the original and the reconstructed set.
- In order to assess the performances of the clustering algorithms on the latent space representations or the encoded images, we extract the encoder function and use the encoded images so produced for the next parts of the project.

### 2.6 Using Kmeans on Latent Space

- we extract the encoder function and use the encoded images so produced in order to cluster them using K means.
- We follow the same steps as in section 2.4 with the only difference being the change in input and input dimension.
- Because we are sending in encoded images of the test set, the dimensions of the input label are different from the dataset label.

```
encoded_images = encoder_tr.predict(test_data)
encoded_images = encoded_images.reshape(-1,7*7*7)
print(encoded_images.shape)
```

```
(10000, 343)
```

### 2.6 Using GMM on Latent Space

- we extract the encoder function and use the encoded images so produced in order to cluster them using GMM
- Because we are sending in encoded images of the test set, the dimensions of the input label are different from the dataset label.

```
encoded_images = encoder_tr.predict(test_data)
encoded_images = encoded_images.reshape(-1,7*7*7)
print(encoded_images.shape)
```

```
(10000, 343)
```

- we use the sklearn.mixture.GaussianMixture library in order to use GMM on the encoded images.
- The parameters are mostly kept default, with the only exception being that of n_components where we mention the number of clusters we want and the max no of iterations to be 300.

```
gmm = GaussianMixture(n_components=10,max_iter=300,n_init=10).fit(encoded_images)
labels_gmm = gmm.predict(encoded_images)
```

- The confusion matrix and NMI are calculated in the same manner as mentioned above.
- Scatter plots representing the same are formulated and print.

### 2.5 Tuning of Hyper-parameter

Parameters are tuned to bring out the change in the accuracy, loss, clusters and to compare relative performances amongst each other

- Epochs and number of layers in deep autoenocder are tuned and analysed.

- Confusion matrix is calculated using the inbuilt sklearns metrics library and the accuracy, is calculated for the test set.
  Thus represented as

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

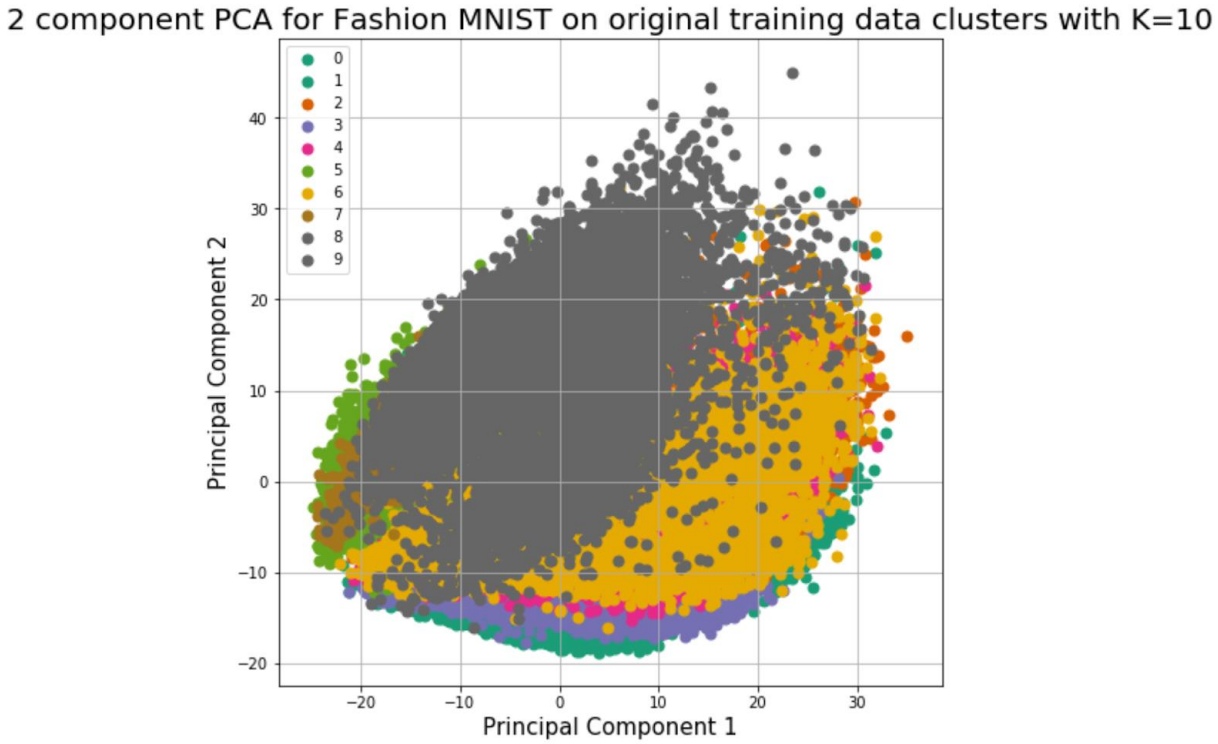
$$Precision = \frac{TP}{TP + FP}$$

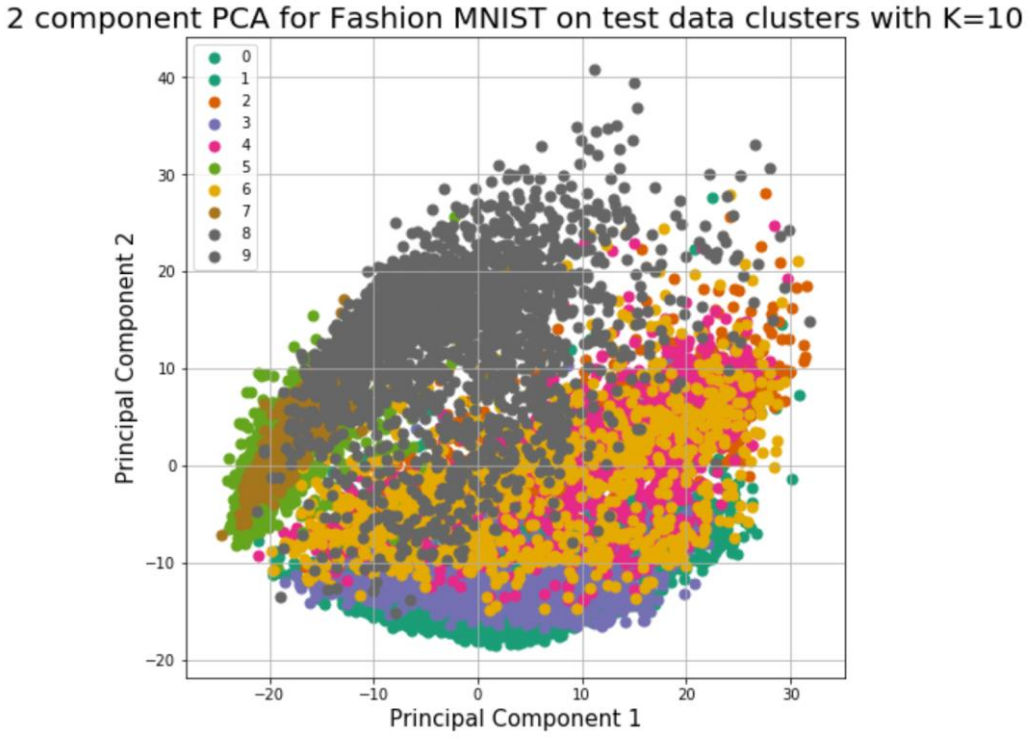
$$Recall = \frac{TP}{TP + FN}$$

Where TP = true positive, TN = true negative, FP = false positive and FN = false negative.

- NMI is also calculated, so as to double check our accuracies. NMI, as previously mentioned is also a good measure of unsupervised learning efficiency.
- The results are reported and analysed.

**Training Data and Test Data representation with their original labels**



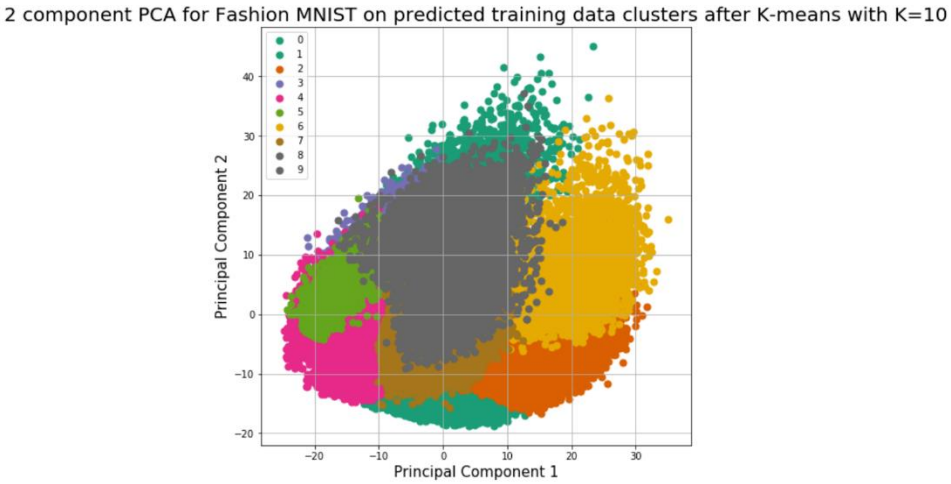



## KMeans Clustering on Training Data

**Accuracy:** 55.33166666666667
**Normalized Mutual Information Score:** 51.16483920105648

**Confusion Matrix:**

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 3404 | 201 | 0 | 0 | 167 | 594 | 1582 | 2 | 50 | 0 |
| **1** | 236 | 5413 | 0 | 0 | 63 | 156 | 129 | 0 | 3 | 0 |
| **2** | 115 | 9 | 0 | 0 | 3519 | 515 | 1787 | 1 | 54 | 0 |
| **3** | 1684 | 3209 | 0 | 0 | 49 | 523 | 523 | 0 | 12 | 0 |
| **4** | 873 | 154 | 0 | 0 | 3596 | 252 | 1081 | 0 | 44 | 0 |
| **5** | 2 | 1 | 0 | 0 | 0 | 3766 | 29 | 1444 | 17 | 741 |
| **6** | 1053 | 62 | 0 | 0 | 1954 | 774 | 2071 | 6 | 79 | 1 |
| **7** | 0 | 0 | 0 | 0 | 0 | 508 | 0 | 4680 | 1 | 811 |
| **8** | 22 | 28 | 0 | 0 | 269 | 494 | 228 | 237 | 4650 | 72 |
| **9** | 2 | 2 | 0 | 0 | 1 | 170 | 38 | 164 | 4 | 5619 |

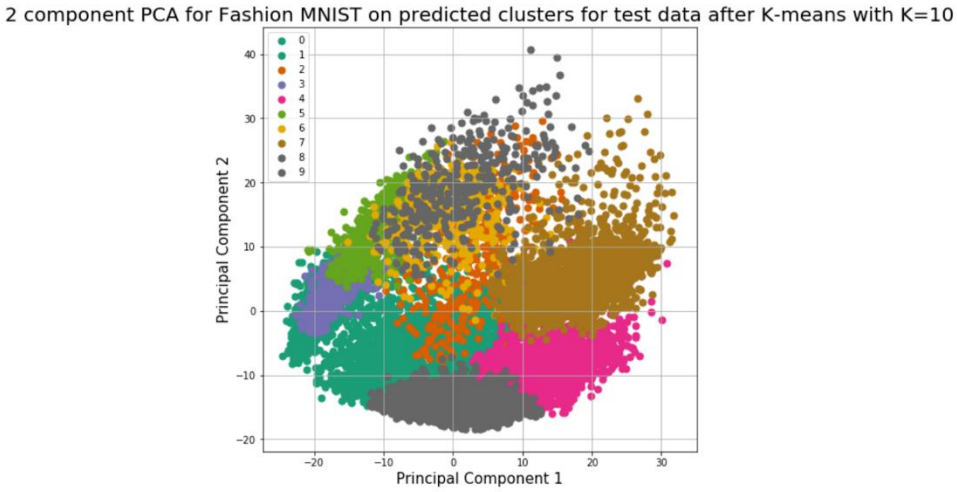



## KMeans Clustering on Test Data

**Accuracy:** 56.03
**Normalized Mutual Information Score:** 51.45761149609541

**Confusion Matrix:**

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 594 | 29 | 0 | 0 | 35 | 80 | 251 | 1 | 10 | 0 |
| **1** | 52 | 890 | 0 | 0 | 9 | 20 | 29 | 0 | 0 | 0 |
| **2** | 24 | 4 | 0 | 0 | 568 | 49 | 347 | 0 | 8 | 0 |
| **3** | 276 | 506 | 0 | 0 | 9 | 84 | 120 | 0 | 5 | 0 |
| **4** | 148 | 27 | 0 | 0 | 619 | 39 | 158 | 0 | 9 | 0 |
| **5** | 0 | 0 | 0 | 0 | 0 | 663 | 6 | 221 | 0 | 110 |
| **6** | 190 | 12 | 0 | 0 | 317 | 108 | 359 | 0 | 14 | 0 |
| **7** | 0 | 0 | 0 | 0 | 0 | 66 | 0 | 774 | 0 | 160 |
| **8** | 3 | 6 | 0 | 0 | 63 | 78 | 44 | 35 | 762 | 9 |
| **9** | 0 | 0 | 0 | 0 | 0 | 29 | 4 | 23 | 2 | 942 |

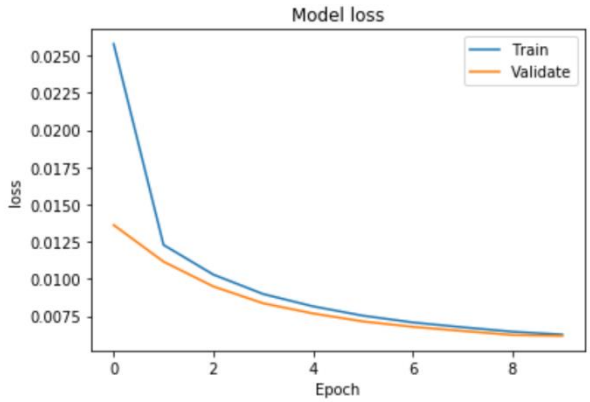


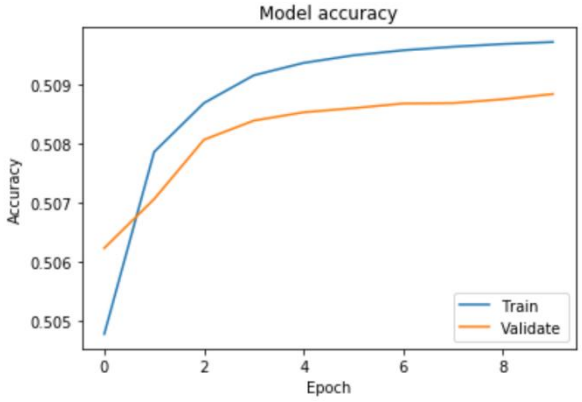

## AutoEncoder with KMeans and GMM

### Case 1

Batch Size: 128
Epochs: 10
Encoder Layers:

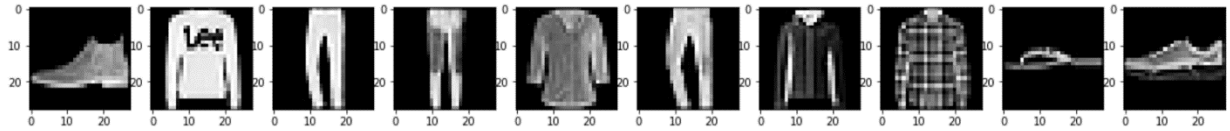
| Layer (type) | Output Shape | Param # |
|---|---|---|
| input_28 (InputLayer) | (None, 28, 28, 1) | 0 |
| conv2d_164 (Conv2D) | (None, 28, 28, 64) | 640 |
| max_pooling2d_45 (MaxPooling | (None, 14, 14, 64) | 0 |
| conv2d_165 (Conv2D) | (None, 14, 14, 32) | 18464 |
| max_pooling2d_46 (MaxPooling | (None, 7, 7, 32) | 0 |
| conv2d_166 (Conv2D) | (None, 7, 7, 16) | 4624 |
| conv2d_167 (Conv2D) | (None, 7, 7, 7) | 1015 |



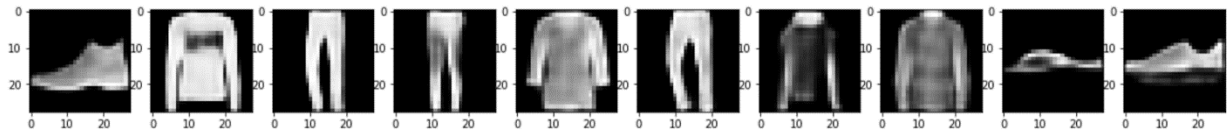




**Autoencoder accuracy for training set:** 50.97190702756246
**Autoencoderloss for training set:** 0.0062704156339168544
**Autoencoder accuracy for validation set:** 50.884002923965454
**Autoencoder loss for validation set:** 0.00618240815276901
**Autoencoder accuracy for Test set:** 50.73575259208679
**Autoencoder loss for Test set:** 0.006218375788629055

Test Images



Reconstruction of Test Images

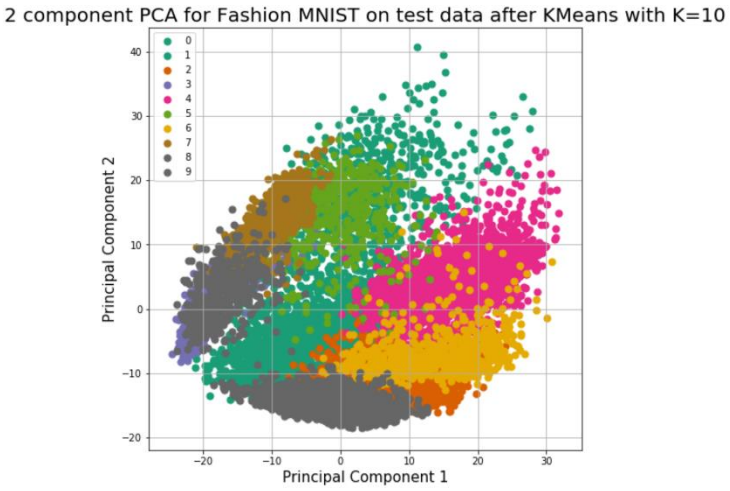


## **KMean** on Encodeds Images

**Accuracy:** 57.25
**Normalized Mutual Information Score:** 56.25728319297452

**Confusion Matrix:**

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 578 | 13 | 0 | 106 | 42 | 1 | 247 | 2 | 11 | 0 |
| **1** | 2 | 860 | 0 | 103 | 12 | 0 | 22 | 0 | 1 | 0 |
| **2** | 3 | 1 | 0 | 18 | 613 | 0 | 342 | 0 | 23 | 0 |
| **3** | 11 | 471 | 0 | 369 | 20 | 0 | 126 | 0 | 3 | 0 |
| **4** | 0 | 10 | 0 | 144 | 692 | 0 | 142 | 0 | 12 | 0 |
| **5** | 0 | 0 | 0 | 0 | 0 | 389 | 5 | 510 | 1 | 95 |
| **6** | 124 | 5 | 0 | 114 | 348 | 0 | 382 | 1 | 26 | 0 |
| **7** | 0 | 0 | 0 | 0 | 0 | 5 | 0 | 803 | 0 | 192 |
| **8** | 1 | 8 | 0 | 33 | 42 | 8 | 161 | 41 | 705 | 1 |
| **9** | 0 | 0 | 0 | 1 | 0 | 35 | 0 | 16 | 1 | 947 |

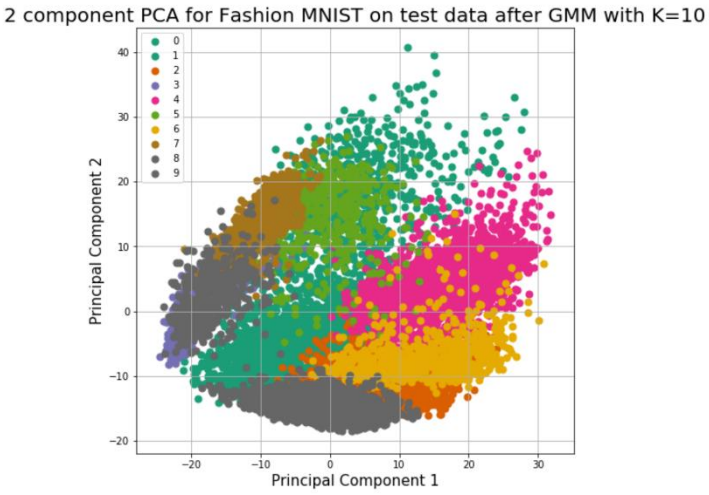



## **GMM** on Encode Images

**Accuracy:** 61.029999999999994
**Normalized Mutual Information Score:** 59.38089706750023

**Confusion Matrix:**

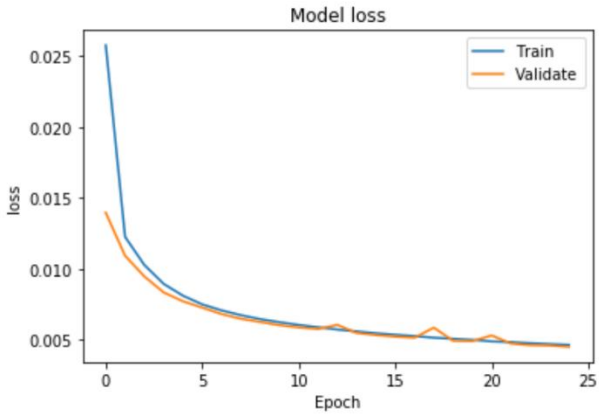
| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 673 | 2 | 38 | 96 | 32 | 2 | 145 | 1 | 11 | 0 |
| **1** | 0 | 808 | 9 | 129 | 33 | 0 | 20 | 0 | 1 | 0 |
| **2** | 6 | 1 | 816 | 3 | 57 | 0 | 105 | 0 | 12 | 0 |
| **3** | 13 | 238 | 11 | 471 | 172 | 0 | 92 | 0 | 3 | 0 |
| **4** | 0 | 1 | 670 | 36 | 227 | 0 | 59 | 0 | 7 | 0 |
| **5** | 0 | 0 | 0 | 1 | 0 | 402 | 3 | 535 | 2 | 57 |
| **6** | 156 | 4 | 517 | 50 | 84 | 0 | 166 | 1 | 22 | 0 |
| **7** | 0 | 0 | 0 | 0 | 0 | 6 | 0 | 892 | 0 | 102 |
| **8** | 1 | 1 | 13 | 21 | 63 | 6 | 133 | 31 | 730 | 1 |
| **9** | 0 | 0 | 0 | 3 | 0 | 26 | 2 | 49 | 2 | 918 |

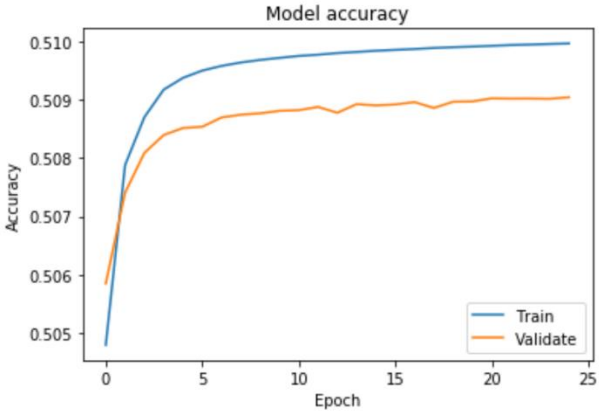



## Case 2

Batch Size: 128
Epochs: 25
Encoder Layers:

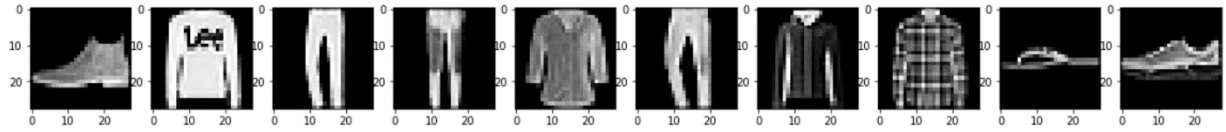
| Layer (type) | Output Shape | Param # |
|---|---|---|
| input_31 (InputLayer) | (None, 28, 28, 1) | 0 |
| conv2d_191 (Conv2D) | (None, 28, 28, 64) | 640 |
| max_pooling2d_51 (MaxPooling | (None, 14, 14, 64) | 0 |
| conv2d_192 (Conv2D) | (None, 14, 14, 32) | 18464 |
| max_pooling2d_52 (MaxPooling | (None, 7, 7, 32) | 0 |
| conv2d_193 (Conv2D) | (None, 7, 7, 16) | 4624 |
| conv2d_194 (Conv2D) | (None, 7, 7, 7) | 1015 |



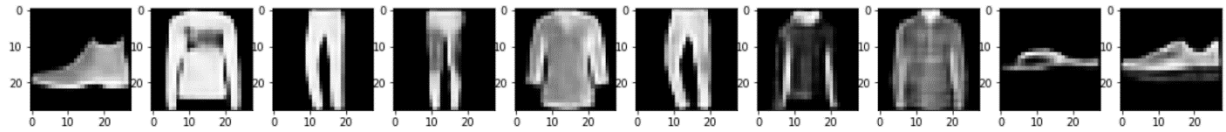




**Autoencoder accuracy for training set:** 50.99693876107534
**Autoencoderloss for training set:** 0.004620865081126491
**Autoencoder accuracy for validation set:** 50.904559850692756
**Autoencoder loss for validation set:** 0.0044796687203149
**Autoencoder accuracy for Test set:** 50.756249971389764
**Autoencoder loss for Test set:** 0.004517494411021471

Test Images



Reconstruction of Test Images

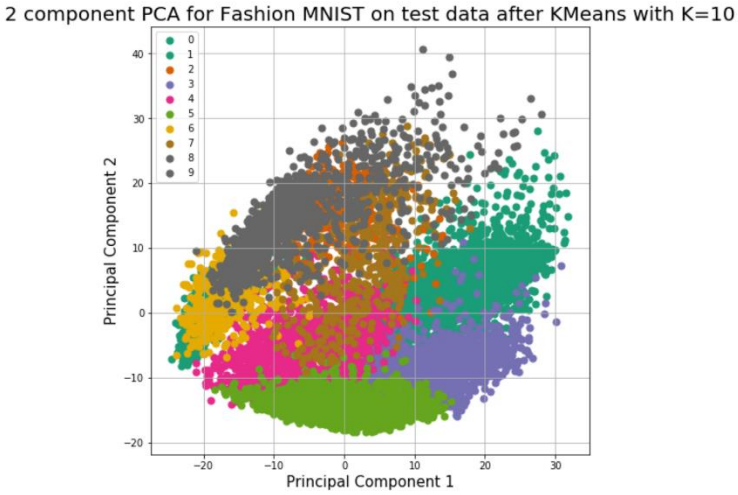


## **KMean** on Encode Images

**Accuracy:** 56.08
**Normalized Mutual Information Score:** 56.48724338980167

**Confusion Matrix:**

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 625 | 47 | 0 | 0 | 45 | 2 | 268 | 1 | 12 | 0 |
| **1** | 42 | 919 | 0 | 0 | 12 | 0 | 26 | 0 | 1 | 0 |
| **2** | 15 | 3 | 0 | 0 | 628 | 0 | 347 | 0 | 7 | 0 |
| **3** | 270 | 581 | 0 | 0 | 16 | 0 | 130 | 0 | 3 | 0 |
| **4** | 121 | 23 | 0 | 0 | 700 | 0 | 147 | 0 | 9 | 0 |
| **5** | 0 | 0 | 0 | 0 | 0 | 391 | 6 | 509 | 1 | 93 |
| **6** | 211 | 18 | 0 | 0 | 350 | 0 | 399 | 1 | 21 | 0 |
| **7** | 0 | 0 | 0 | 0 | 0 | 4 | 0 | 784 | 0 | 212 |
| **8** | 3 | 14 | 0 | 0 | 42 | 10 | 54 | 32 | 843 | 2 |
| **9** | 1 | 0 | 0 | 0 | 0 | 35 | 0 | 16 | 1 | 947 |

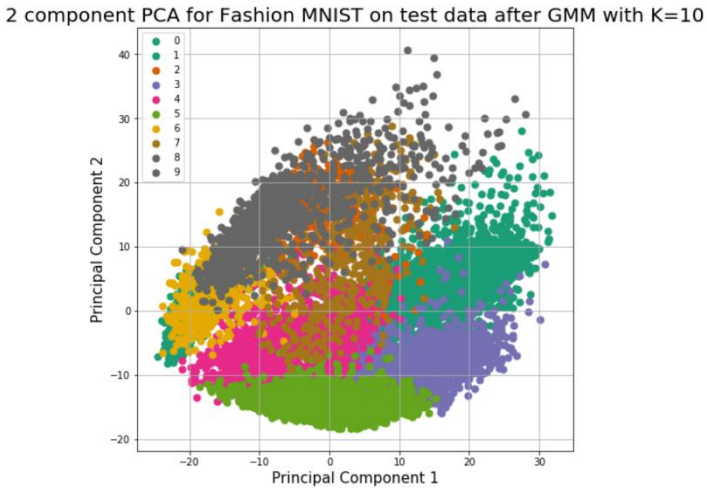



## **GMM** on Encode Images

**Accuracy:** 59.36
**Normalized Mutual Information Score:** 60.088127806780975

**Confusion Matrix:**

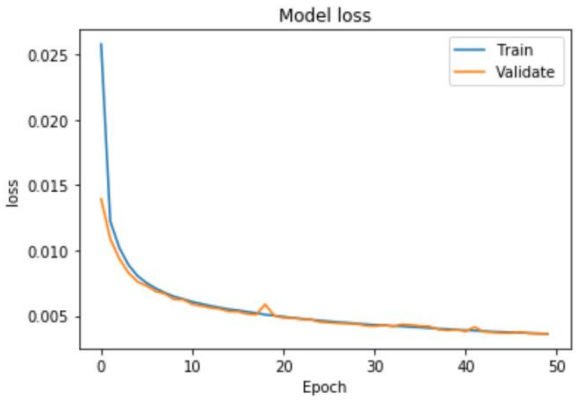
| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 692 | 0 | 43 | 126 | 0 | 0 | 125 | 2 | 11 | 1 |
| **1** | 0 | 866 | 9 | 106 | 0 | 0 | 18 | 0 | 1 | 0 |
| **2** | 7 | 0 | 835 | 46 | 0 | 0 | 102 | 0 | 10 | 0 |
| **3** | 15 | 4 | 17 | 886 | 0 | 0 | 75 | 0 | 3 | 0 |
| **4** | 0 | 1 | 714 | 217 | 0 | 0 | 61 | 0 | 7 | 0 |
| **5** | 0 | 0 | 0 | 0 | 0 | 0 | 5 | 527 | 0 | 468 |
| **6** | 169 | 1 | 534 | 130 | 0 | 0 | 144 | 1 | 21 | 0 |
| **7** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 791 | 0 | 209 |
| **8** | 1 | 1 | 14 | 67 | 0 | 0 | 142 | 30 | 744 | 1 |
| **9** | 0 | 0 | 0 | 2 | 0 | 0 | 0 | 18 | 2 | 978 |

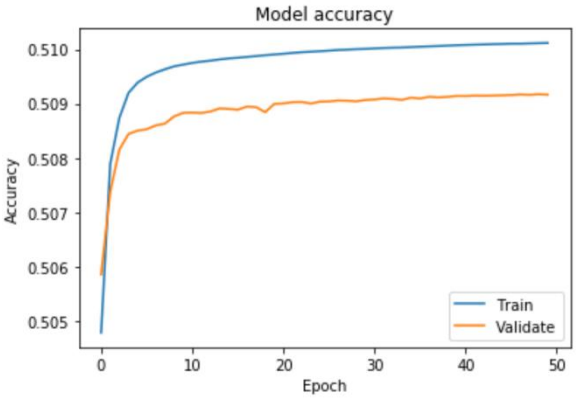



## Case 3

Batch Size: 128
Epochs: 50
Encoder Layers:

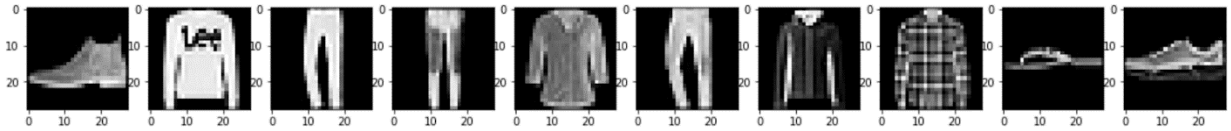
| Layer (type) | Output Shape | Param # |
|---|---|---|
| input_29 (InputLayer) | (None, 28, 28, 1) | 0 |
| conv2d_173 (Conv2D) | (None, 28, 28, 64) | 640 |
| max_pooling2d_47 (MaxPooling | (None, 14, 14, 64) | 0 |
| conv2d_174 (Conv2D) | (None, 14, 14, 32) | 18464 |
| max_pooling2d_48 (MaxPooling | (None, 7, 7, 32) | 0 |
| conv2d_175 (Conv2D) | (None, 7, 7, 16) | 4624 |
| conv2d_176 (Conv2D) | (None, 7, 7, 7) | 1015 |



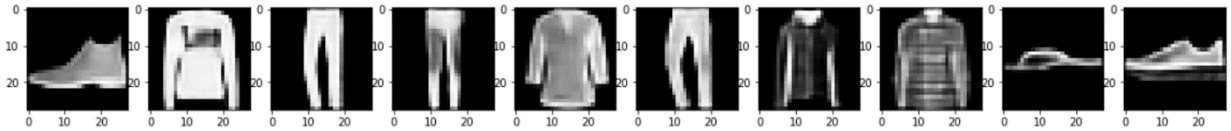




**Autoencoder accuracy for training set:** 51.01163368225098
**Autoencoderloss for training set:** 0.003611553032572071
**Autoencoder accuracy for validation set:** 50.91647534370423
**Autoencoder loss for validation set:** 0.0036327683261285224
**Autoencoder accuracy for Test set:** 50.77005103111267
**Autoencoder loss for Test set:** 0.0036662487968802454

Test Images



Reconstruction of Test Images

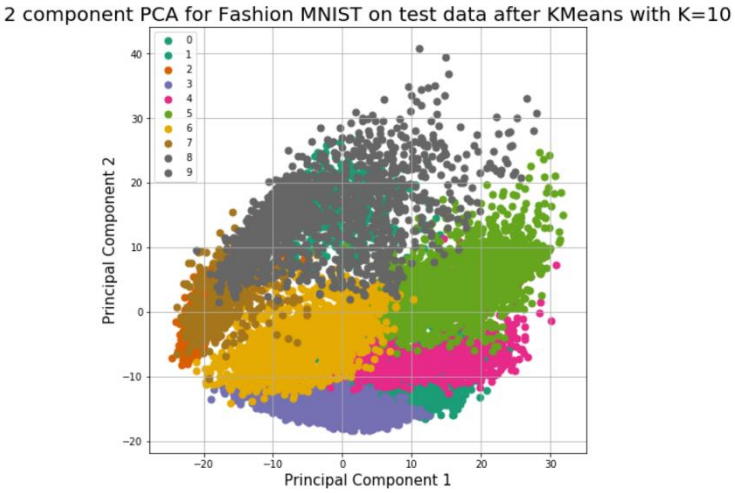


## **KMean** on Encode Images

**Accuracy:** 57.89
**Normalized Mutual Information Score:** 56.55528651892057

**Confusion Matrix:**

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 589 | 19 | 0 | 107 | 45 | 1 | 227 | 1 | 11 | 0 |
| **1** | 4 | 884 | 0 | 78 | 13 | 0 | 20 | 0 | 1 | 0 |
| **2** | 5 | 1 | 0 | 15 | 620 | 0 | 339 | 0 | 20 | 0 |
| **3** | 11 | 466 | 0 | 408 | 21 | 0 | 91 | 0 | 3 | 0 |
| **4** | 0 | 10 | 0 | 134 | 702 | 0 | 144 | 0 | 10 | 0 |
| **5** | 0 | 0 | 0 | 0 | 0 | 405 | 4 | 510 | 0 | 81 |
| **6** | 135 | 7 | 0 | 107 | 344 | 0 | 375 | 1 | 31 | 0 |
| **7** | 0 | 0 | 0 | 0 | 0 | 7 | 0 | 792 | 0 | 201 |
| **8** | 1 | 11 | 0 | 29 | 40 | 13 | 169 | 47 | 687 | 3 |
| **9** | 0 | 0 | 0 | 2 | 0 | 31 | 0 | 19 | 1 | 947 |

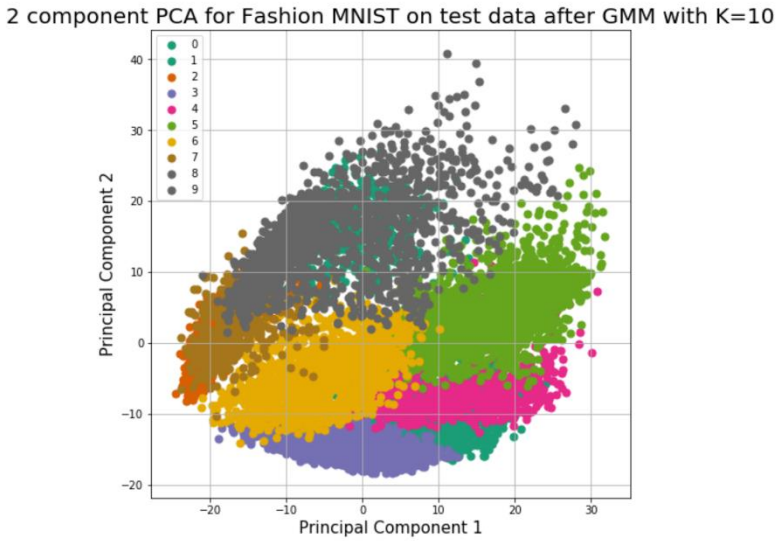



## **GMM** on Encode Images

**Accuracy:** 59.64
**Normalized Mutual Information Score:** 60.2655374956955

**Confusion Matrix:**

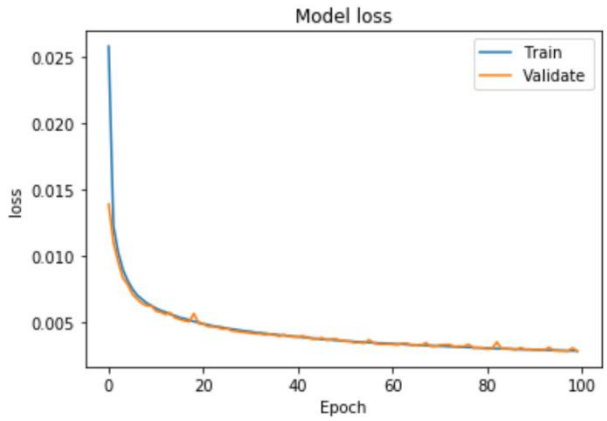
| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 698 | 1 | 42 | 131 | 0 | 0 | 113 | 2 | 13 | 0 |
| **1** | 2 | 866 | 8 | 106 | 0 | 0 | 17 | 0 | 1 | 0 |
| **2** | 6 | 0 | 843 | 43 | 0 | 0 | 97 | 0 | 11 | 0 |
| **3** | 12 | 11 | 15 | 899 | 0 | 0 | 60 | 0 | 3 | 0 |
| **4** | 0 | 1 | 715 | 219 | 0 | 0 | 58 | 0 | 7 | 0 |
| **5** | 0 | 0 | 0 | 0 | 0 | 0 | 3 | 577 | 0 | 420 |
| **6** | 168 | 2 | 528 | 127 | 0 | 0 | 143 | 1 | 31 | 0 |
| **7** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 808 | 0 | 192 |
| **8** | 1 | 1 | 12 | 71 | 0 | 0 | 142 | 35 | 735 | 3 |
| **9** | 0 | 0 | 0 | 2 | 0 | 0 | 0 | 24 | 2 | 972 |

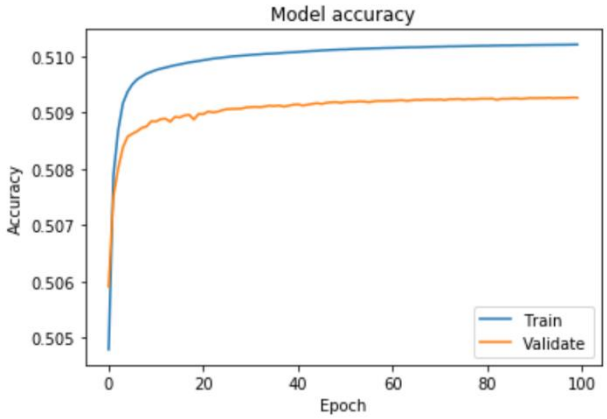



## Case 4

Batch Size: 128
Epochs: 100
Encoder Layers:

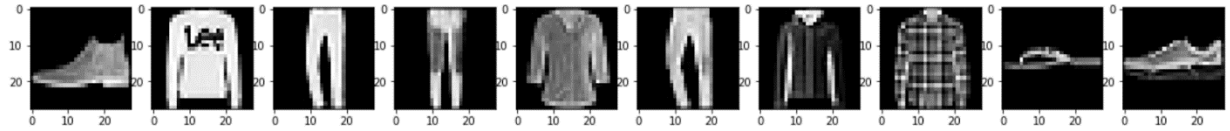
| Layer (type) | Output Shape | Param # |
|---|---|---|
| input_30 (InputLayer) | (None, 28, 28, 1) | 0 |
| conv2d_182 (Conv2D) | (None, 28, 28, 64) | 640 |
| max_pooling2d_49 (MaxPooling | (None, 14, 14, 64) | 0 |
| conv2d_183 (Conv2D) | (None, 14, 14, 32) | 18464 |
| max_pooling2d_50 (MaxPooling | (None, 7, 7, 32) | 0 |
| conv2d_184 (Conv2D) | (None, 7, 7, 16) | 4624 |
| conv2d_185 (Conv2D) | (None, 7, 7, 7) | 1015 |



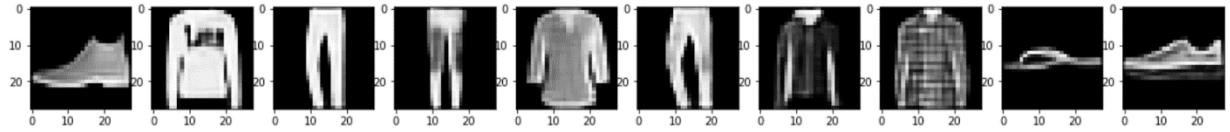




**Autoencoder accuracy for training set:** 51.02073504924775
**Autoencoderloss for training set:** 0.0028501253140469393
**Autoencoder accuracy for validation set:** 50.92626509666442
**Autoencoder loss for validation set:** 0.0028193462590376537
**Autoencoder accuracy for Test set:** 50.779451599121096
**Autoencoder loss for Test set:** 0.0028411265186965465

Test Images



Reconstruction of Test Images

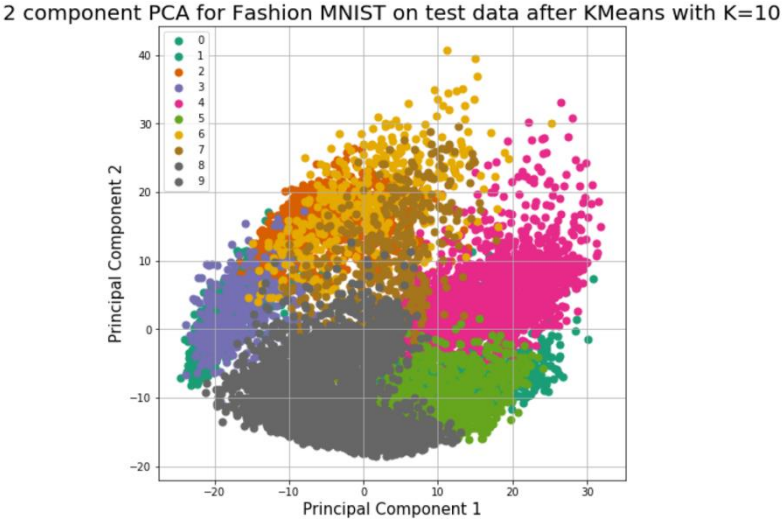


## **KMean** on Encode Images

**Accuracy:** 58.60999999999999
**Normalized Mutual Information Score:** 57.74294012297651

**Confusion Matrix:**

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 574 | 30 | 0 | 101 | 47 | 1 | 236 | 1 | 10 | 0 |
| **1** | 4 | 892 | 0 | 69 | 10 | 0 | 24 | 0 | 1 | 0 |
| **2** | 4 | 1 | 0 | 23 | 610 | 0 | 357 | 0 | 5 | 0 |
| **3** | 12 | 536 | 0 | 343 | 17 | 0 | 91 | 0 | 1 | 0 |
| **4** | 0 | 14 | 0 | 166 | 659 | 0 | 153 | 0 | 8 | 0 |
| **5** | 0 | 0 | 0 | 0 | 0 | 401 | 5 | 542 | 2 | 50 |
| **6** | 126 | 11 | 0 | 111 | 349 | 0 | 387 | 2 | 14 | 0 |
| **7** | 0 | 0 | 0 | 0 | 0 | 6 | 0 | 908 | 0 | 86 |
| **8** | 2 | 19 | 0 | 11 | 62 | 8 | 58 | 37 | 801 | 2 |
| **9** | 0 | 0 | 0 | 5 | 0 | 27 | 4 | 67 | 1 | 896 |

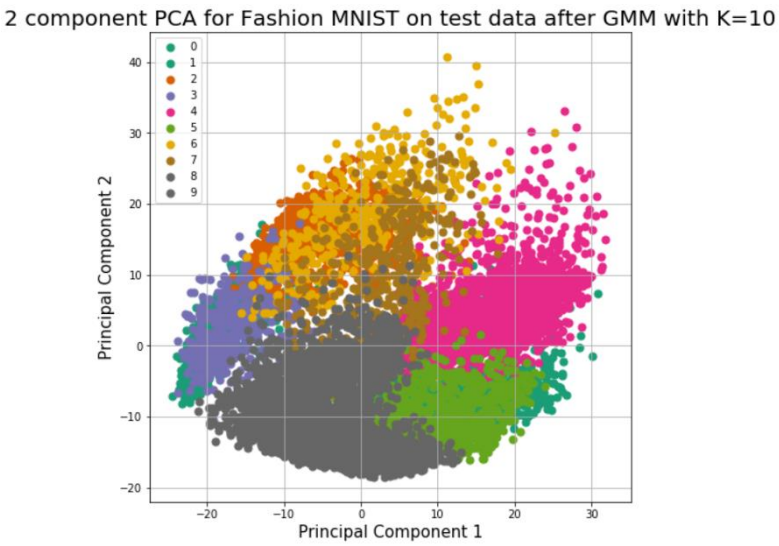



## **GMM** on Encode Images

**Accuracy:** 58.98
**Normalized Mutual Information Score:** 59.57059036178043

**Confusion Matrix:**

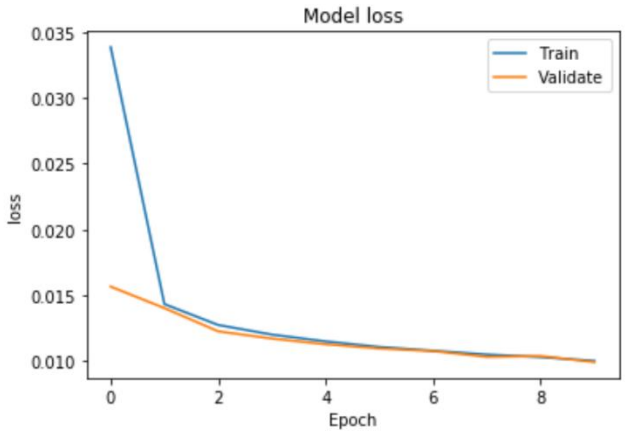
| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 699 | 37 | 48 | 98 | 0 | 1 | 0 | 1 | 116 | 0 |
| **1** | 4 | 902 | 8 | 63 | 0 | 0 | 0 | 0 | 23 | 0 |
| **2** | 5 | 1 | 878 | 22 | 0 | 0 | 0 | 0 | 94 | 0 |
| **3** | 12 | 524 | 21 | 371 | 0 | 0 | 0 | 0 | 72 | 0 |
| **4** | 0 | 11 | 756 | 164 | 0 | 0 | 0 | 0 | 69 | 0 |
| **5** | 0 | 0 | 0 | 0 | 0 | 398 | 0 | 516 | 4 | 82 |
| **6** | 169 | 14 | 543 | 116 | 0 | 0 | 0 | 2 | 156 | 0 |
| **7** | 0 | 0 | 0 | 0 | 0 | 4 | 0 | 790 | 0 | 206 |
| **8** | 2 | 14 | 16 | 21 | 0 | 9 | 0 | 31 | 903 | 4 |
| **9** | 0 | 0 | 0 | 2 | 0 | 21 | 0 | 18 | 2 | 957 |

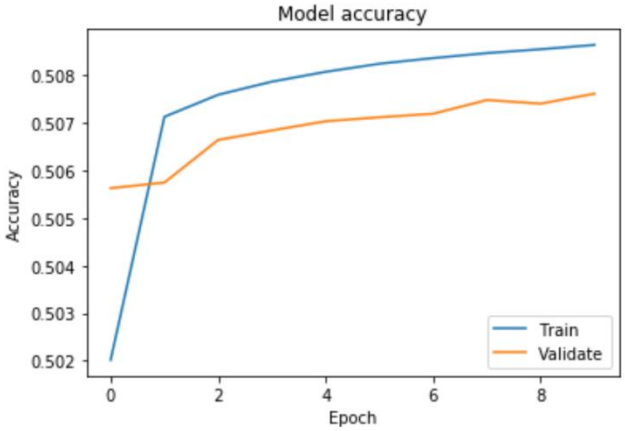



## Case 5

Batch Size: 128
Epochs: 10
Encoder Layers:

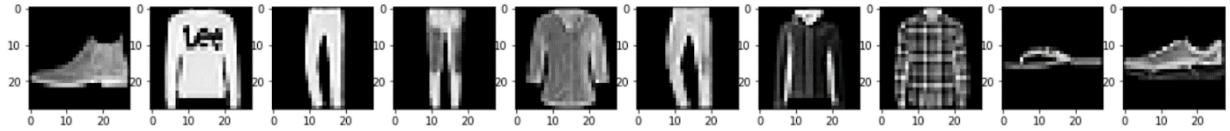
| Layer (type) | Output Shape | Param # |
|---|---|---|
| input_33 (InputLayer) | (None, 28, 28, 1) | 0 |
| conv2d_209 (Conv2D) | (None, 28, 28, 64) | 640 |
| max_pooling2d_55 (MaxPooling | (None, 14, 14, 64) | 0 |
| conv2d_210 (Conv2D) | (None, 14, 14, 32) | 18464 |
| max_pooling2d_56 (MaxPooling | (None, 7, 7, 32) | 0 |
| conv2d_211 (Conv2D) | (None, 7, 7, 16) | 4624 |
| conv2d_212 (Conv2D) | (None, 7, 7, 7) | 1015 |
| conv2d_213 (Conv2D) | (None, 7, 7, 1) | 64 |



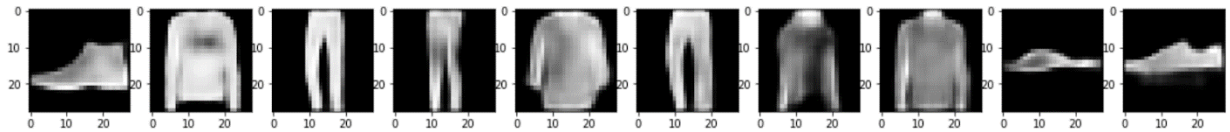




**Autoencoder accuracy for training set:** 50.86418205897013
**Autoencoderloss for training set:** 0.010009482945005099
**Autoencoder accuracy for validation set:** 50.761585744222
**Autoencoder loss for validation set:** 0.009918003735442956
**Autoencoder accuracy for Test set:** 50.616989879608155
**Autoencoder loss for Test set:** 0.009940810371935367

Test Images



Reconstruction of Test Images

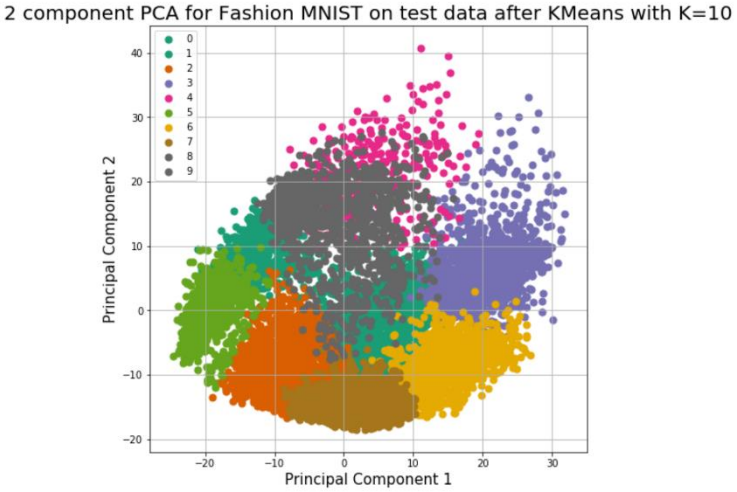


## **KMean** on Encode Images

**Accuracy:** 50.46000000000001
**Normalized Mutual Information Score:** 48.202021408613035

**Confusion Matrix:**

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 446 | 37 | 215 | 0 | 49 | 12 | 233 | 2 | 6 | 0 |
| **1** | 45 | 794 | 21 | 0 | 5 | 0 | 134 | 0 | 1 | 0 |
| **2** | 21 | 4 | 358 | 0 | 421 | 4 | 184 | 0 | 8 | 0 |
| **3** | 260 | 441 | 64 | 0 | 8 | 2 | 222 | 0 | 3 | 0 |
| **4** | 128 | 29 | 261 | 0 | 475 | 6 | 92 | 0 | 9 | 0 |
| **5** | 0 | 0 | 1 | 0 | 0 | 760 | 16 | 189 | 0 | 34 |
| **6** | 138 | 24 | 301 | 0 | 238 | 25 | 261 | 2 | 11 | 0 |
| **7** | 0 | 0 | 0 | 0 | 0 | 403 | 0 | 581 | 0 | 16 |
| **8** | 2 | 6 | 15 | 0 | 53 | 27 | 88 | 110 | 699 | 0 |
| **9** | 3 | 0 | 3 | 0 | 2 | 27 | 2 | 288 | 3 | 672 |

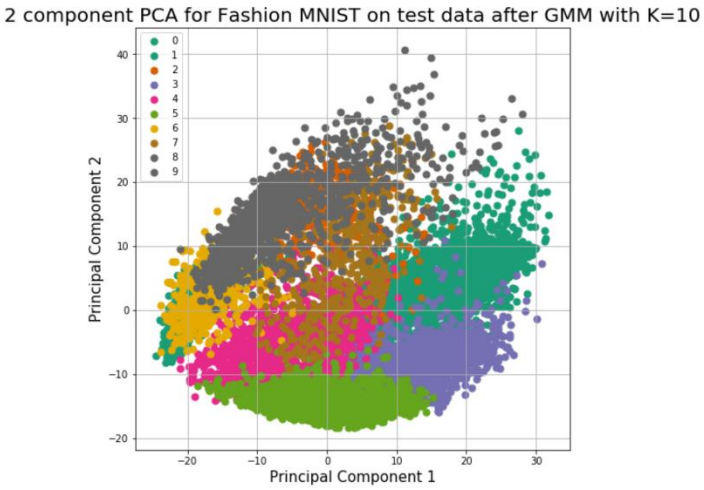



## **GMM** on Encode Images

**Accuracy:** 64.03999999999999
**Normalized Mutual Information Score:** 61.252293036939356

**Confusion Matrix:**

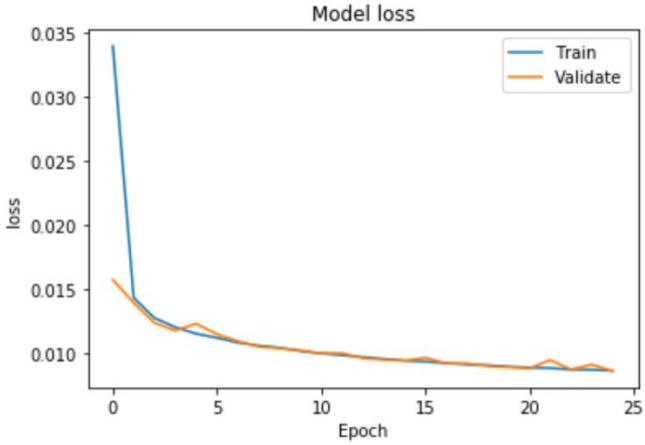
| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 689 | 6 | 5 | 100 | 41 | 0 | 0 | 0 | 159 | 0 |
| **1** | 0 | 878 | 0 | 94 | 9 | 0 | 0 | 0 | 19 | 0 |
| **2** | 9 | 0 | 575 | 8 | 312 | 0 | 0 | 0 | 96 | 0 |
| **3** | 40 | 182 | 2 | 616 | 109 | 0 | 0 | 0 | 51 | 0 |
| **4** | 1 | 0 | 450 | 42 | 452 | 0 | 0 | 0 | 55 | 0 |
| **5** | 0 | 0 | 0 | 0 | 0 | 514 | 0 | 474 | 6 | 6 |
| **6** | 195 | 4 | 381 | 88 | 200 | 0 | 0 | 0 | 132 | 0 |
| **7** | 0 | 0 | 0 | 0 | 0 | 45 | 0 | 837 | 1 | 117 |
| **8** | 0 | 1 | 0 | 5 | 2 | 2 | 0 | 2 | 987 | 1 |
| **9** | 0 | 0 | 0 | 0 | 0 | 103 | 0 | 39 | 2 | 856 |

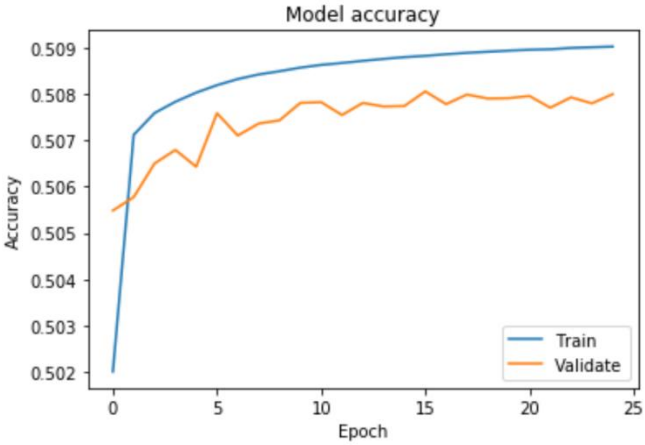



## Case 6

Batch Size: 128
Epochs: 25
Encoder Layers:

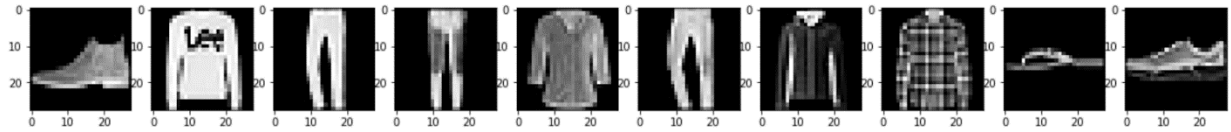
| Layer (type) | Output Shape | Param # |
|---|---|---|
| input_34 (InputLayer) | (None, 28, 28, 1) | 0 |
| conv2d_220 (Conv2D) | (None, 28, 28, 64) | 640 |
| max_pooling2d_57 (MaxPooling | (None, 14, 14, 64) | 0 |
| conv2d_221 (Conv2D) | (None, 14, 14, 32) | 18464 |
| max_pooling2d_58 (MaxPooling | (None, 7, 7, 32) | 0 |
| conv2d_222 (Conv2D) | (None, 7, 7, 16) | 4624 |
| conv2d_223 (Conv2D) | (None, 7, 7, 7) | 1015 |
| conv2d_224 (Conv2D) | (None, 7, 7, 1) | 64 |



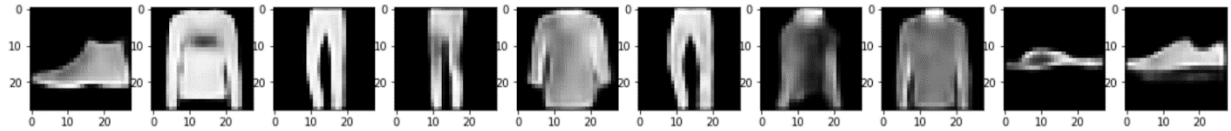




**Autoencoder accuracy for training set:** 50.902598937352494
**Autoencoderloss for training set:** 0.00861988844225804
**Autoencoder accuracy for validation set:** 50.79972364107768
**Autoencoder loss for validation set:** 0.008580441477398077
**Autoencoder accuracy for Test set:** 50.65727038383484
**Autoencoder loss for Test set:** 0.008601331052184105

Test Images



Reconstruction of Test Images

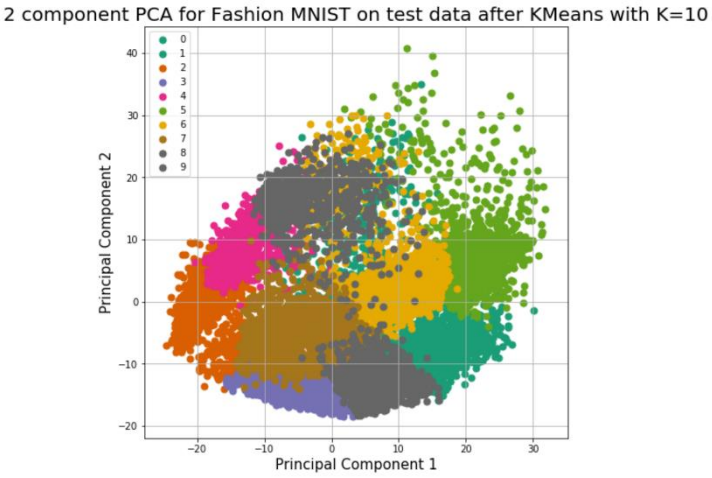


## **KMean** on Encode Images

**Accuracy:** 50.05
**Normalized Mutual Information Score:** 46.33992717710239

**Confusion Matrix:**

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 488 | 20 | 41 | 95 | 30 | 37 | 285 | 2 | 2 | 0 |
| **1** | 24 | 722 | 12 | 221 | 3 | 3 | 15 | 0 | 0 | 0 |
| **2** | 28 | 2 | 315 | 7 | 347 | 18 | 281 | 0 | 2 | 0 |
| **3** | 146 | 278 | 13 | 428 | 2 | 26 | 106 | 0 | 1 | 0 |
| **4** | 136 | 11 | 305 | 53 | 357 | 16 | 116 | 0 | 6 | 0 |
| **5** | 0 | 0 | 0 | 0 | 0 | 769 | 18 | 178 | 1 | 34 |
| **6** | 137 | 7 | 225 | 54 | 186 | 52 | 337 | 2 | 0 | 0 |
| **7** | 0 | 0 | 0 | 0 | 0 | 437 | 0 | 550 | 0 | 13 |
| **8** | 2 | 6 | 194 | 9 | 114 | 30 | 94 | 159 | 390 | 2 |
| **9** | 0 | 0 | 0 | 2 | 2 | 28 | 4 | 314 | 1 | 649 |

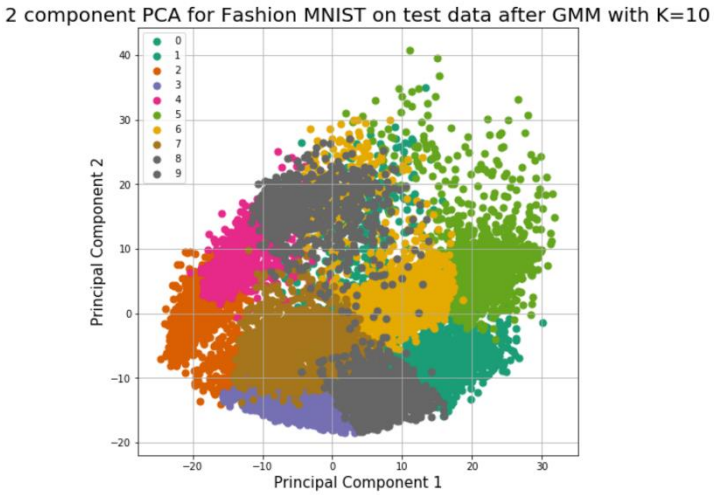



## **GMM** on Encode Images

**Accuracy:** 58.96
**Normalized Mutual Information Score:** 60.482943802890965

**Confusion Matrix:**

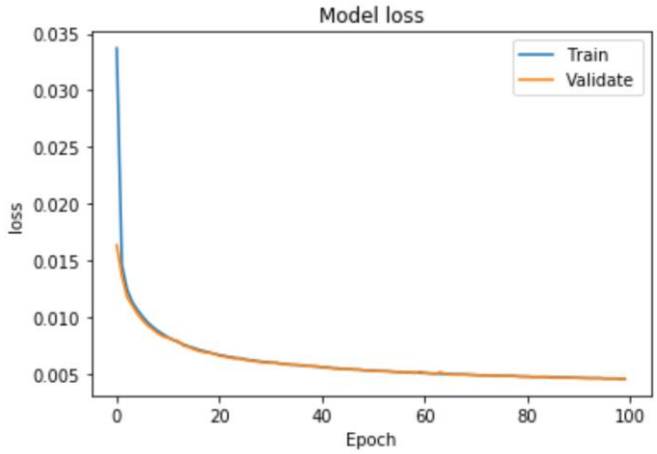
| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 789 | 1 | 2 | 84 | 0 | 0 | 116 | 0 | 8 | 0 |
| **1** | 67 | 915 | 0 | 14 | 0 | 0 | 4 | 0 | 0 | 0 |
| **2** | 136 | 0 | 746 | 13 | 0 | 0 | 105 | 0 | 0 | 0 |
| **3** | 149 | 86 | 2 | 685 | 0 | 0 | 77 | 0 | 1 | 0 |
| **4** | 105 | 0 | 628 | 105 | 0 | 0 | 162 | 0 | 0 | 0 |
| **5** | 4 | 0 | 0 | 0 | 0 | 0 | 0 | 423 | 5 | 568 |
| **6** | 247 | 1 | 449 | 69 | 0 | 0 | 229 | 0 | 5 | 0 |
| **7** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 714 | 1 | 285 |
| **8** | 100 | 1 | 0 | 1 | 0 | 0 | 66 | 1 | 828 | 3 |
| **9** | 2 | 0 | 0 | 0 | 0 | 0 | 0 | 8 | 0 | 990 |

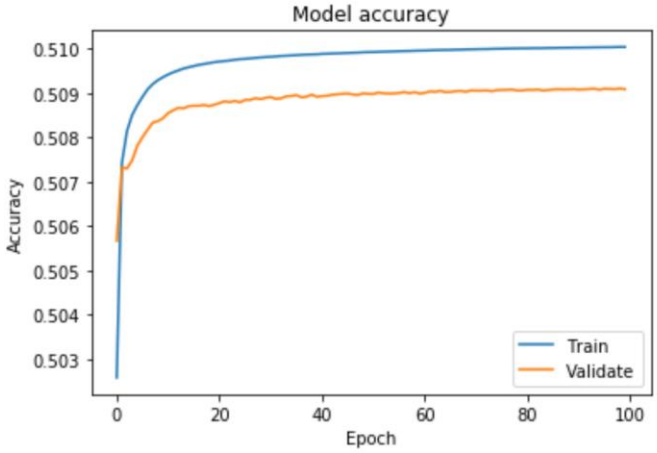



## Case 7

Batch Size: 128
Epochs: 100
Encoder Layers:

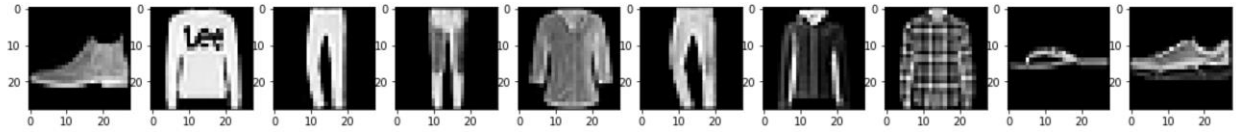
| Layer (type) | Output Shape | Param # |
|---|---|---|
| input_37 (InputLayer) | (None, 28, 28, 1) | 0 |
| conv2d_245 (Conv2D) | (None, 28, 28, 32) | 320 |
| max_pooling2d_63 (MaxPooling | (None, 14, 14, 32) | 0 |
| conv2d_246 (Conv2D) | (None, 14, 14, 16) | 4624 |
| max_pooling2d_64 (MaxPooling | (None, 7, 7, 16) | 0 |
| conv2d_247 (Conv2D) | (None, 7, 7, 7) | 1015 |



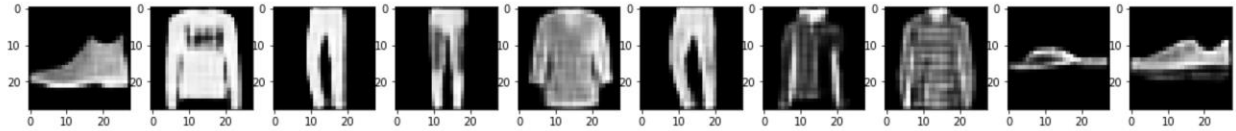




**Autoencoder accuracy for training set:** 51.00404439767202
**Autoencoderloss for training set:** 0.004545479162906607
**Autoencoder accuracy for validation set:** 50.90880112648011
**Autoencoder loss for validation set:** 0.004527053128927946
**Autoencoder accuracy for Test set:** 50.76146686553955
**Autoencoder loss for Test set:** 0.004563230994343757

Test Images



Reconstruction of Test Images

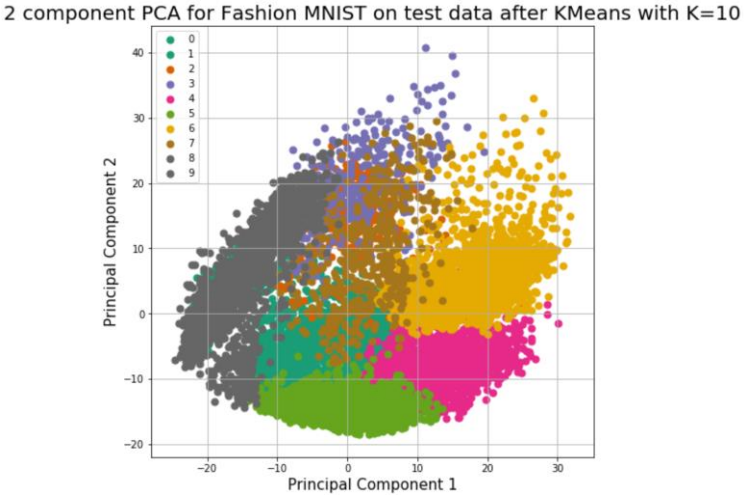


## **KMean** on Encode Images

**Accuracy:** 55.45
**Normalized Mutual Information Score:** 52.46180545623302

**Confusion Matrix:**

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 596 | 36 | 0 | 0 | 49 | 45 | 263 | 1 | 10 | 0 |
| **1** | 59 | 899 | 0 | 0 | 10 | 7 | 24 | 0 | 1 | 0 |
| **2** | 36 | 3 | 0 | 0 | 581 | 16 | 356 | 0 | 8 | 0 |
| **3** | 275 | 557 | 0 | 0 | 8 | 40 | 115 | 0 | 5 | 0 |
| **4** | 172 | 25 | 0 | 0 | 617 | 15 | 163 | 0 | 8 | 0 |
| **5** | 0 | 0 | 0 | 0 | 0 | 546 | 6 | 322 | 1 | 125 |
| **6** | 195 | 17 | 0 | 0 | 342 | 52 | 381 | 1 | 12 | 0 |
| **7** | 0 | 0 | 0 | 0 | 0 | 35 | 0 | 808 | 1 | 156 |
| **8** | 3 | 11 | 0 | 0 | 74 | 38 | 73 | 42 | 758 | 1 |
| **9** | 0 | 2 | 0 | 0 | 0 | 29 | 0 | 28 | 1 | 940 |

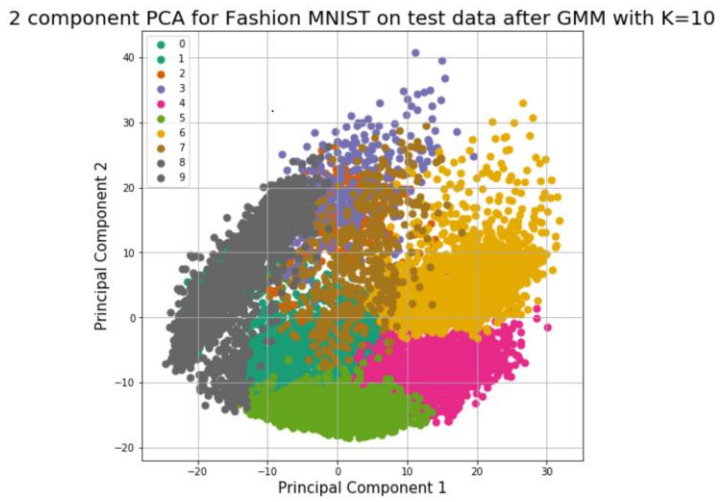



## **GMM** on Encode Images

**Accuracy:** 59.830000000000005
**Normalized Mutual Information Score:** 60.24886018302886

**Confusion Matrix:**

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 736 | 37 | 46 | 96 | 0 | 11 | 0 | 1 | 73 | 0 |
| **1** | 2 | 902 | 14 | 71 | 0 | 2 | 0 | 0 | 9 | 0 |
| **2** | 7 | 3 | 921 | 26 | 0 | 5 | 0 | 0 | 38 | 0 |
| **3** | 14 | 467 | 25 | 437 | 0 | 5 | 0 | 0 | 52 | 0 |
| **4** | 0 | 8 | 759 | 206 | 0 | 3 | 0 | 0 | 24 | 0 |
| **5** | 0 | 0 | 0 | 0 | 0 | 330 | 0 | 514 | 4 | 152 |
| **6** | 183 | 15 | 579 | 120 | 0 | 10 | 0 | 1 | 92 | 0 |
| **7** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 769 | 0 | 231 |
| **8** | 1 | 6 | 15 | 11 | 0 | 25 | 0 | 22 | 915 | 5 |
| **9** | 0 | 1 | 0 | 1 | 0 | 6 | 0 | 17 | 2 | 973 |


2 component PCA for Fashion MNIST on test data after GMM with K=10
0
1
2
3
4
5
6
7
8
9
Principal Component 2
Principal Component 1
40
30
20
10
0
−10
−20
−20
−10
0
10
20
30

## Evaluations & Observations

1. GMM is a better clustering algorithm as compared to K-means for such type of datasets as we have used (datasets containing related data). It tends to perform better as it is a soft clustering approach compared to a hard one as Kmeans.
2. Kmeans on the original training dataset gives an accuracy of 55.33%, NMI: 51.16%
3. Kmeans on original test dataset gives an accuracy of 56.03%, NMI 51.46%
4. As observed, both Kmeans and GMM perform better after the dimensionality reduction using auto encoder.
5. As observed from the above results and test cases, GMM tends to perform better for Kmeans.
6. As number of epochs progresses, loss is reduced and more accurate images are produced from the auto encoder.
7. Also, as the number of epochs increases, accuracy for Kmeans increases. Also, the accuracy for GMM decreases slightly with the increase in epochs, but it still performs better than Kmeans.
8. From case 1-4 where we are reducing the dimensions to 7*7*7 representation, both the performances of Kmeans and GMM clustering have comparable performances and do not have a lot of gap in between their measures. But, GMM still performs better than KMeans Clustering with accuracy reaching to 60%.

   With a 14 Layer network with dimensionality reduction to 7x7x7 following are the accuracies:

| Epochs | K-Means | GMM |
|---|---|---|
| 10 | 57.25 | 61.02 |
| 25 | 56.08 | 59.36 |
| 50 | 57.89 | 59.64 |
| 100 | 58.60 | 58.98 |

9. The best performances can be observed between epochs of 50-100, which shows a constant rise in accuracy of both the models and a constant decrease in loss.
10. On reducing the dimensionality of the encoded image to 7 * 7*1 instead of 7*7*7 by adding more convolutional layers, we get the following accuracies.

| Epochs | Kmeans | GMM |
|---|---|---|
| 10 | 50 | 64 |
| 25 | 50.05 | 58.96 |

11. Accuracy of GMM increases in the above case, but Kmeans decreases. Since, the accuracy of Kmeans is pretty low, this network is not used and the network with dimensionality reduction to 7x7x7 is used.
12. In Case 7, where the number of total layers are 12 and the dimensionality reduction is still 7x7x7 both KMeans and GMM perform well and give an accuracy of **55.45%** and **59.83%** respectively.
13. Also, from the PCA representations of the clustering algorithms and comparing it with original, we can see that the images are clustered better after applying the autoencoder network.

## Acknowledgments

Thankful to Professor Srihari and the group of Teaching Assistants who have helped us ace the projects.

## References

1. https://scikit-learn.org/stable/modules/generated/sklearn.metrics.normalized_mutual_info_score.html
2. https://www.datacamp.com/community/tutorials/autoencoder-keras-tutorial
3. https://www.deeplearningbook.org/contents/autoencoders.html
4. https://www-users.cs.umn.edu/~kumar001/dmbook/index.php
5. https://towardsdatascience.com/the-5-clustering-algorithms-data-scientists-need-to-know-a36d136ef68
6. https://www.datacamp.com/community/tutorials/k-means-clustering-r
7. https://www.kaggle.com/s00100624/digit-image-clustering-via-autoencoder-kmeans