CSE 574: Introduction to Machine Learning

Report for Project 3:

Cluster Analysis using Unsupervised Learning on Fashion MNIST Image Dataset

Submitted by:

Vishva Nitin Patel

UB # 50318625 || vishvani@buffalo.edu

ABSTRACT:

Presented in this report is the implementation of (A) K-means algorithm, (B) Auto-Encoder based K-means clustering model and (C) Auto-Encoder based Gaussian Mixture Model clustering model in order to perform clustering and cluster analysis on the Fashion MNIST Dataset. The clustering task will be that of recognizing an image and identifying it as one of ten classes (i.e. clusters). I have implemented the clustering program(s) in python from scratch and findings of the same (accuracy and loss) along with the related code snippets are attached herewith.

1. Introduction:

K-means clustering is one of the simplest and vastly popular unsupervised machine learning algorithms. Typically, unsupervised algorithms make inferences from Datasets using only input vectors without referring to known, or labeled, outcomes. In this project we have utilized the K-means clustering algorithm along with implementing an Auto-Encoder based K-means clustering model and an Auto-Encoder based Gaussian Mixture Model clustering model.

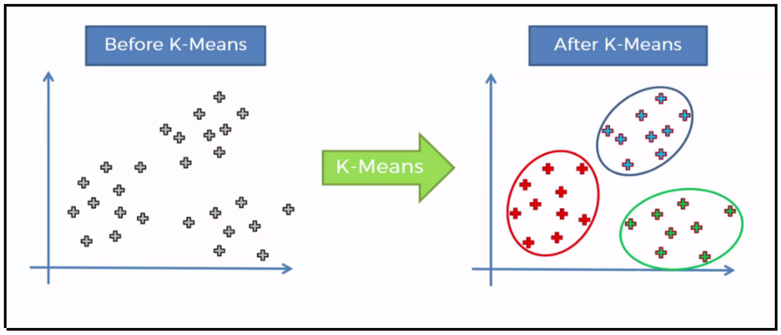


Fig1 – Basics of K-Means Clustering

Components:

- **K-means algorithm** - The K-Means algorithm clusters data by trying to separate samples in n groups of equal variance, minimizing a criterion known as the inertia or within-cluster sum-of-squares (see below). This algorithm requires the number of clusters to be specified. It scales large number of samples and has been used across a large range of application areas in many different fields. The k-means algorithm divides a set of samples into disjoint clusters, each described by the mean of the samples in the cluster. The means are commonly called the cluster centroids.

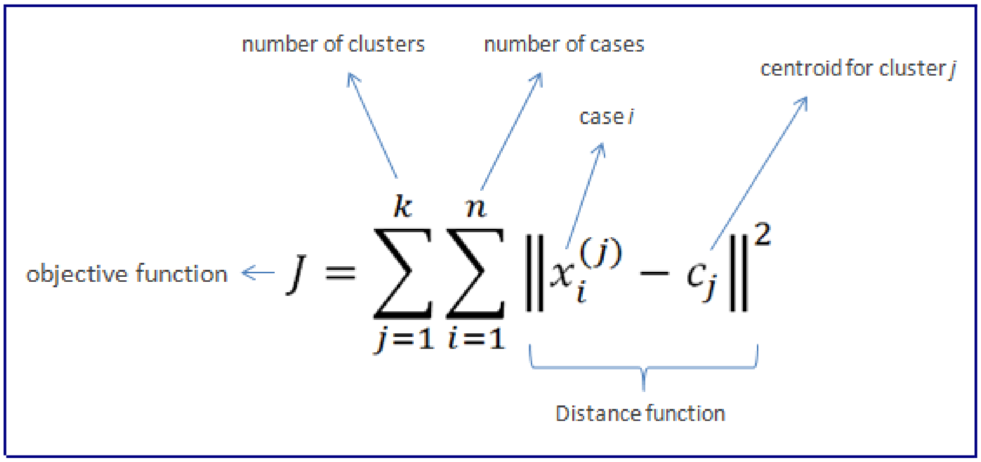


Fig2 – The K-Means Equation explained

- **Auto Encoder** - ”Auto-encoding” is a data compression algorithm where the compression and decompression functions are data-specific, lossy, and can learn automatically from examples rather than being engineered by a human. Auto-encoder uses compression and decompression functions which are implemented with neural networks. To build an auto-encoder, you need three things: an encoding function, a decoding function, and a distance function between the amount of information loss between the compressed representation of your data and the decompressed representation (i.e. a ”loss” function). The encoder and decoder will be chosen to be parametric functions (typically neural networks), and to be differential with respect to the distance function, so the parameters of the encoding/decoding functions can be optimize to minimize the reconstruction loss, using Stochastic Gradient Descent.

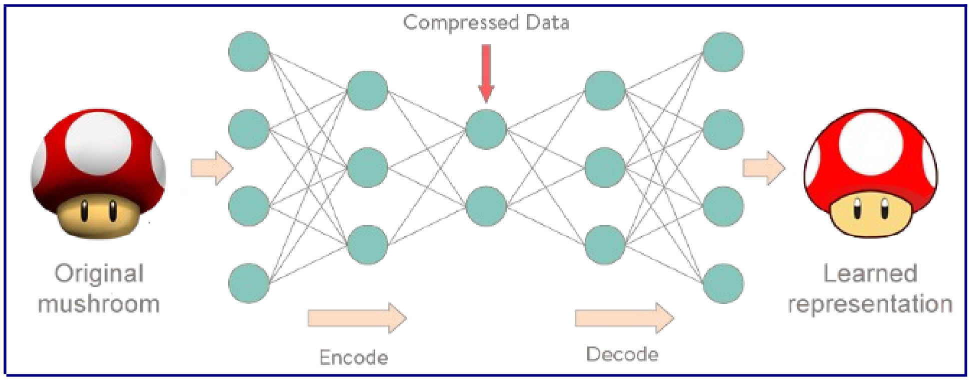


Fig3 – Understanding what an Auto-encoder does

- **Auto-Encoder with K-Means Clustering** - The K-means algorithm aims to choose centroids that minimize the inertia, or within-cluster sum-of-squares criterion. Inertia can be recognized as a measure of how internally coherent clusters are. Inertia is not a normalized metric: we just know that lower values are better and zero is optimal. But in very high dimensional spaces, Euclidean distances tend to become inflated (this is an instance of the so-called curse of dimensionality). Running a dimensionality reduction algorithm such as Principal component analysis (PCA) or Auto-encoder prior to k-means clustering can alleviate this problem and speed up the computations.

- **Auto-Encoder with GMM Clustering** - A Gaussian mixture model is a probabilistic model that assumes all the data points are generated from a mixture of a finite number of Gaussian distributions with unknown parameters. One can think of mixture models as generalizing k-means clustering to incorporate information about the covariance structure of the data as well as the centers of the latent Gaussians. The GaussianMixture 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. A Gaussian Mixture.fit method is provided that learns a Gaussian Mixture Model from train data. Given test data, it can assign to each sample the Gaussian it mostly probably belong to using the GaussianMixture.predict method.

2. Dataset definition:

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 28×28 grayscale image, associated with a label from 10 classes. Fashion-MNIST is intended to serve as a direct drop-in replacement of the original MNIST dataset for bench marking machine learning algorithms.

Dataset Partitioning:

- Training Data: Used for learning and hence making predictions about the testing data. We take 60,000 training examples in this project.
- Testing Data: Used for depicting how accurate our model is and how well it performs. Predictions made about the Testing Data show how well our model has been trained. We take 10,000 testing examples in this project.

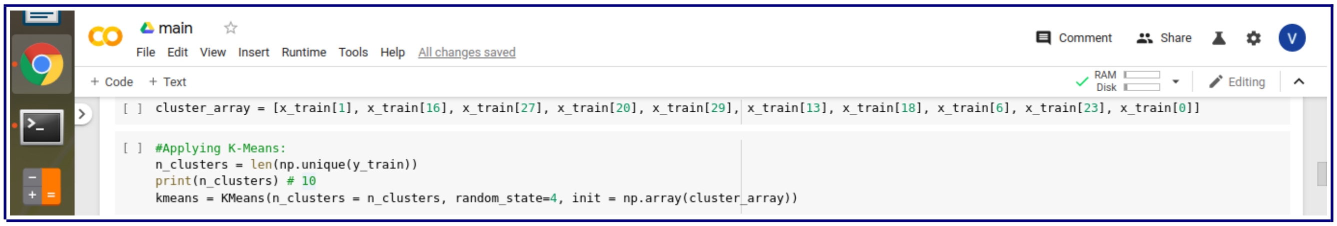
3. Architecture and Work Flow:

3.1 – Extracting feature values:

Fashion MNIST dataset is downloaded and processed into a Numpy array that contains the feature vectors and a Numpy array that contains the labels. Normalization and necessary reshaping were performed as a part of pre-processing.

3.2 – K-Means Clustering:

Using Sklearns library to cluster Fashion MNIST dataset into 10 clusters. Determined the clustering accuracy.



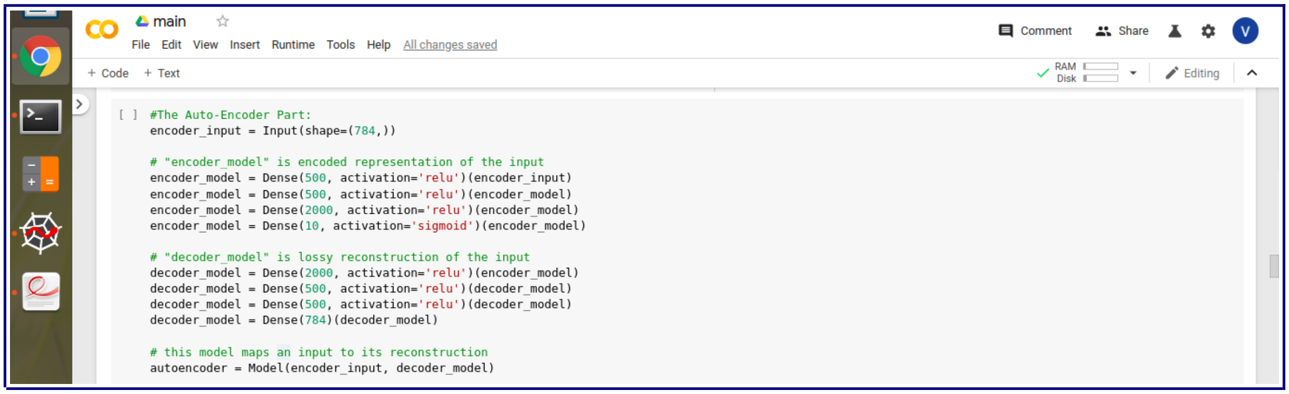
```
[ ] cluster_array = [x_train[1], x_train[16], x_train[27], x_train[20], x_train[29], x_train[13], x_train[18], x_train[6], x_train[23], x_train[0]]
```

```
[ ] #Applying K-Means:
    n_clusters = len(np.unique(y_train))
    print(n_clusters) # 10
    kmeans = KMeans(n_clusters = n_clusters, random_state=4, init = np.array(cluster_array))
```

Fig4 – K Means using Sklearns library

3.3 – Training using Auto-Encoder Network:

Used Keras library to build a Auto-Encoder Network.



```
[ ] #The Auto-Encoder Part:
    encoder_input = Input(shape=(784,))

    # "encoder_model" is encoded representation of the input
    encoder_model = Dense(500, activation='relu')(encoder_input)
    encoder_model = Dense(500, activation='relu')(encoder_model)
    encoder_model = Dense(2000, activation='relu')(encoder_model)
    encoder_model = Dense(10, activation='sigmoid')(encoder_model)

    # "decoder_model" is lossy reconstruction of the input
    decoder_model = Dense(2000, activation='relu')(encoder_model)
    decoder_model = Dense(500, activation='relu')(decoder_model)
    decoder_model = Dense(500, activation='relu')(decoder_model)
    decoder_model = Dense(784)(decoder_model)

    # this model maps an input to its reconstruction
    autoencoder = Model(encoder_input, decoder_model)
```

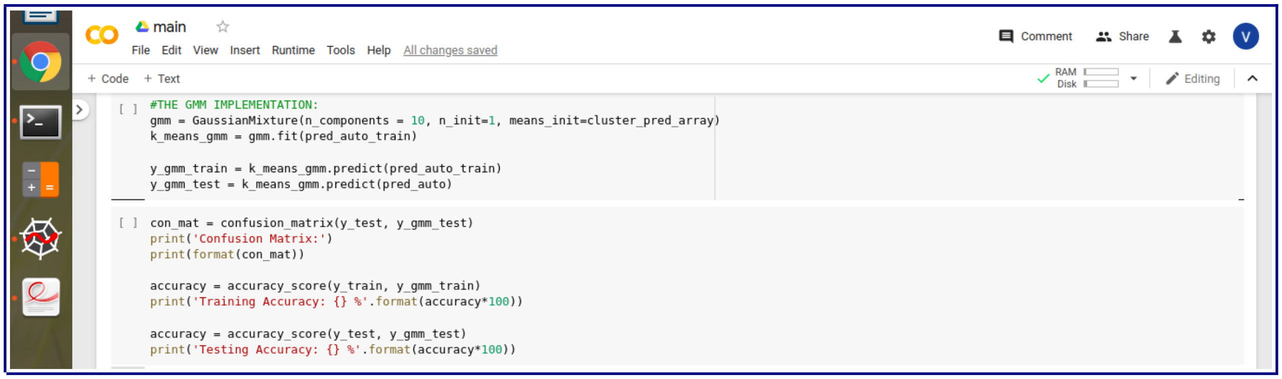
Fig5 – Auto-Encoder Logic’s Code snippet

3.4 – Creating K-Means clustering layer:

By training the auto-encoder, the encoder “learned” to compress each image into latent floating point values. I have used K-Means to generate the cluster centroids (I.e the 10 cluster centers in this case) in the latent feature space. I have then cluster-compressed latent codes produced by Auto-Encoder using K-Means.

3.5 – Creating GMM clustering layer:

Similar to Step 4, I have used Gaussian Mixture Model to generate the cluster centroids, which are the 10 cluster centers in the latent feature space (Latent space provided by pre-trained encoder network)

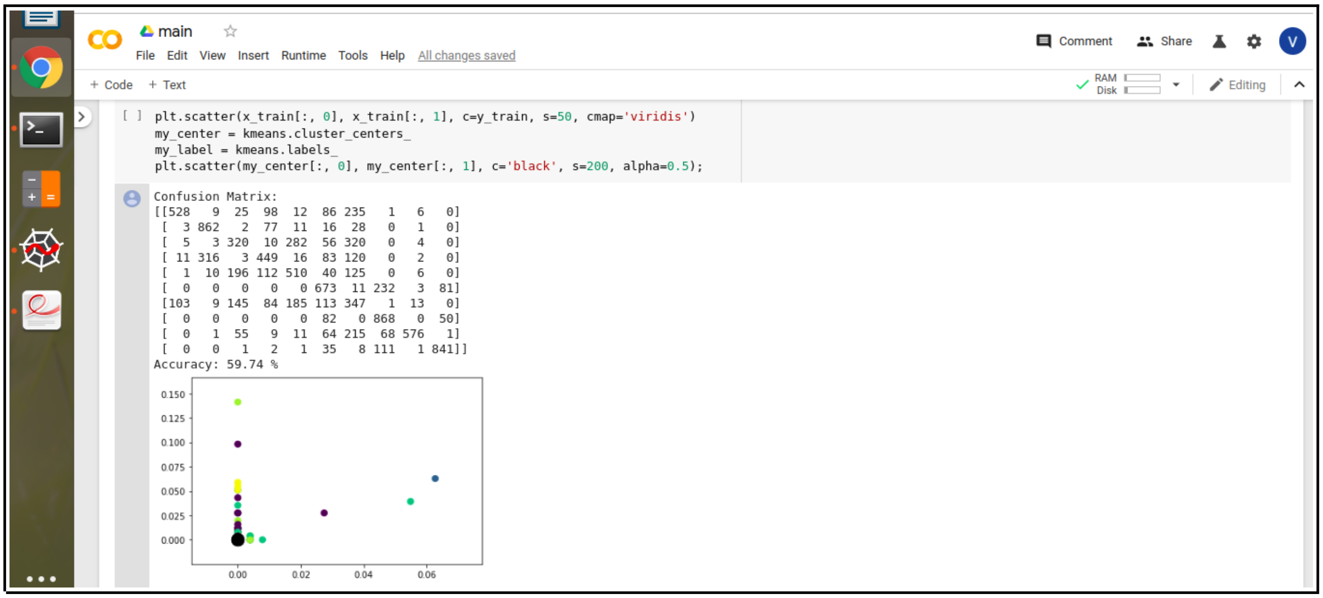


```
[ ] #THE GMM IMPLEMENTATION:
    gmm = GaussianMixture(n_components = 10, n_init=1, means_init=cluster_pred_array)
    k_means_gmm = gmm.fit(pred_auto_train)

    y_gmm_train = k_means_gmm.predict(pred_auto_train)
    y_gmm_test = k_means_gmm.predict(pred_auto)

[ ] con_mat = confusion_matrix(y_test, y_gmm_test)
    print('Confusion Matrix:')
    print(format(con_mat))

    accuracy = accuracy_score(y_train, y_gmm_train)
    print('Training Accuracy: {} %'.format(accuracy*100))

    accuracy = accuracy_score(y_test, y_gmm_test)
    print('Testing Accuracy: {} %'.format(accuracy*100))
```

Fig6 – GMM Implementation Code snippet

3.6 – Tested my machine learning scheme:

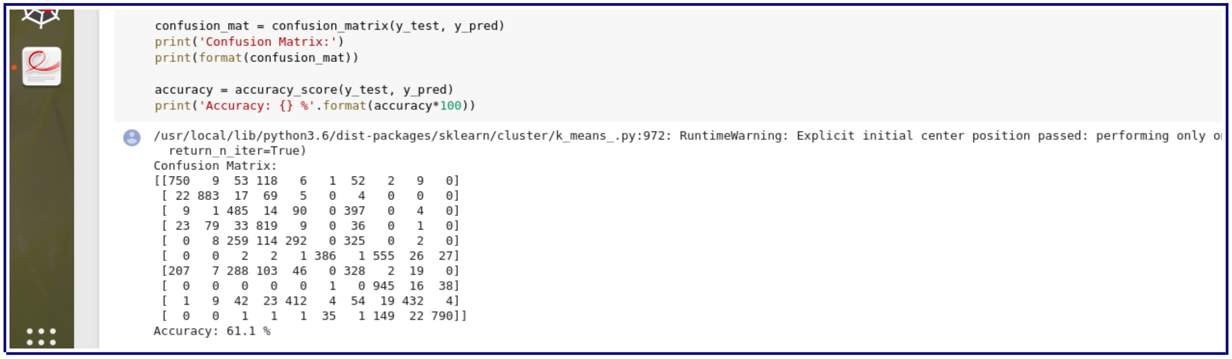
Created a confusion matrix and determined the clustering accuracy by matching the clustering assignment for step 3.2, 3.4 and 3.5.



```
plt.scatter(x_train[:, 0], x_train[:, 1], c=y_train, s=50, cmap='viridis')
my_center = kmeans.cluster_centers_
my_label = kmeans.labels_
plt.scatter(my_center[:, 0], my_center[:, 1], c='black', s=200, alpha=0.5);
```

```
Confusion Matrix:
[[528   9  25  98  12  86 235   1   6   0]
 [  3 862   2  77  11  16  28   0   1   0]
 [  5   3 320  10 282  56 320   0   4   0]
 [ 11 316   3 449  16  83 120   0   2   0]
 [  1  10 196 112 510  40 125   0   6   0]
 [  0   0   0   0   0 673  11 232   3  81]
 [103   9 145  84 185 113 347   1  13   0]
 [  0   0   0   0   0  82   0 868   0  50]
 [  0   1  55   9  11  64 215  68 576   1]
 [  0   0   1   2   1  35   8 111   1 841]]
Accuracy: 59.74 %
```

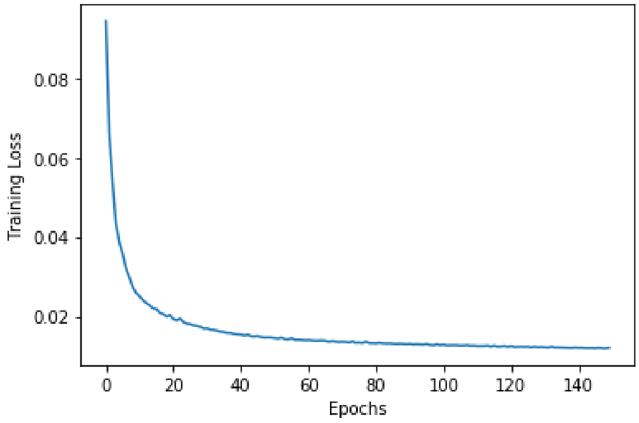
Fig7 – Confusion Matrix and Accuracy(s) for Part 1



```
confusion_mat = confusion_matrix(y_test, y_pred)
print('Confusion Matrix:')
print(format(confusion_mat))

accuracy = accuracy_score(y_test, y_pred)
print('Accuracy: {} %'.format(accuracy*100))
```

```
/usr/local/lib/python3.6/dist-packages/sklearn/cluster/k_means_.py:972: RuntimeWarning: Explicit initial center position passed: performing only o
  return_n_iter=True)
Confusion Matrix:
[[750   9  53 118   6   1  52   2   9   0]
 [ 22 883  17  69   5   0   4   0   0   0]
 [  9   1 485  14  90   0 397   0   4   0]
 [ 23  79  33 819   9   0  36   0   1   0]
 [  0   8 259 114 292   0 325   0   2   0]
 [  0   0   2   2   1 386   1 555  26  27]
 [207   7 288 103  46   0 328   2  19   0]
 [  0   0   0   0   0   1   0 945  16  38]
 [  1   9  42  23 412   4  54  19 432   4]
 [  0   0   1   1   1  35   1 149  22 790]]
Accuracy: 61.1 %
```

Fig8 – Confusion Matrix and Accuracy(s) for Part 2



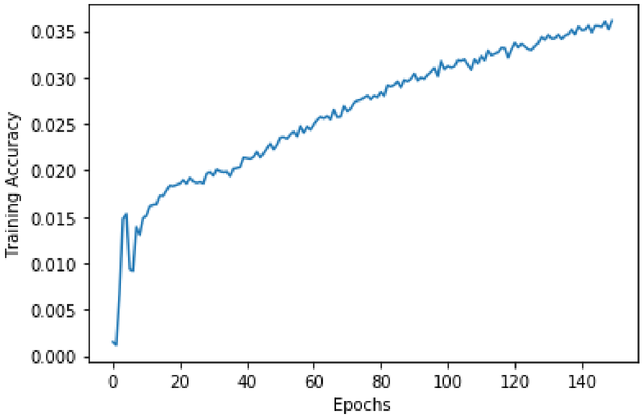
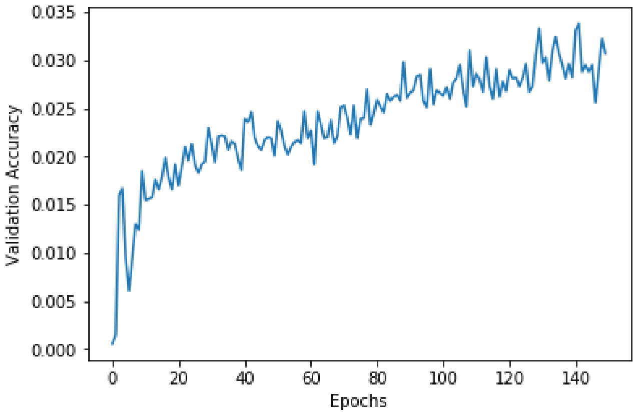
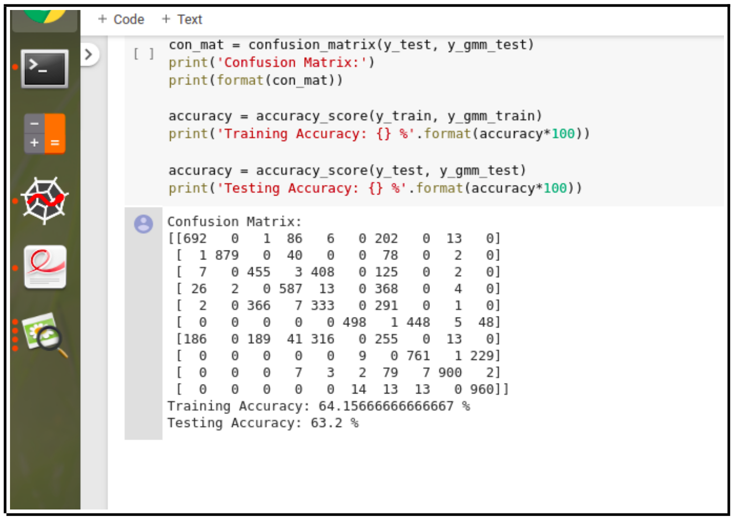


Fig9 – Part 2 training loss

Fig10 – Part 2 training accuracy

Fig11 – Part 2 Validation Accuracy





```
con_mat = confusion_matrix(y_test, y_gmm_test)
print('Confusion Matrix:')
print(format(con_mat))

accuracy = accuracy_score(y_train, y_gmm_train)
print('Training Accuracy: {} %'.format(accuracy*100))

accuracy = accuracy_score(y_test, y_gmm_test)
print('Testing Accuracy: {} %'.format(accuracy*100))
```

```
Confusion Matrix:
[[692   0   1  86   6   0 202   0  13   0]
 [  1 879   0  40   0   0  78   0   2   0]
 [  7   0 455   3 408   0 125   0   2   0]
 [ 26   2   0 587  13   0 368   0   4   0]
 [  2   0 366   7 333   0 291   0   1   0]
 [  0   0   0   0   0 498   1 448   5  48]
 [186   0 189  41 316   0 255   0  13   0]
 [  0   0   0   0   0   9   0 761   1 229]
 [  0   0   0   7   3   2  79   7 900   2]
 [  0   0   0   0   0  14  13  13   0 960]]
Training Accuracy: 64.15666666666667 %
Testing Accuracy: 63.2 %
```

Fig12 – Confusion Matrix and Accuracy(s) for Part 3

6. Conclusion:

The accuracy for all three parts of the project3 are as follows:

Part-1 – K-Means Algorithm: **Accuracy = 59%** (approx)

Part-2 – Auto-Encoder based K-Means clustering model: **Accuracy = 61%** (approx)

Part-3 – Auto-Encoder based Gaussian Mixture Model clustering model: **Training Accuracy = 64%** (approx) || **Testing Accuracy = 63%** (approx)

7. References:

(1) Class notes

(2) Project Description (iml_project3.pdf)

(3) https://en.wikipedia.org/wiki/K-means_clustering

(4) https://towardsdatascience.com/k-means-clustering-identifying-f-r-i-e-n-d-s-in-the-world-of-strangers-695537505d

(5) https://www.dlology.com/blog/how-to-do-unsupervised-clustering-with-keras/

(6) https://medium.com/datadriveninvestor/k-means-clustering-for-imagery-analysis-56c9976f16b6

(7) https://towardsdatascience.com/understanding-k-means-clustering-in-machine-learning-6a6e67336aa1

(8) https://ai-mrkogao.github.io/reinforcement%20learning/clusteringkeras/

(9) https://jakevdp.github.io/PythonDataScienceHandbook/05.12-gaussian-mixtures.html

(10) https://www.analyticsvidhya.com/blog/2018/05/essentials-of-deep-learning-trudging-into-unsupervised-deep-learning/

(11) https://towardsdatascience.com/gaussian-mixture-modelling-gmm-833c88587c7f

(12) https://www.kaggle.com/s00100624/digit-image-clustering-via-autoencoder-kmeans

(13) https://github.com/xoraus/K-Means-Clustering-for-Imagery-Analysis/blob/master/KMeans%20Clustering%20for%20Imagery%20Analysis%20(Jupyter%20Notebook).ipynb

(14) https://www.kaggle.com/residentmario/dimensionality-reduction-and-pca-for-fashion-mnist/notebook

(15) https://www.analyticsvidhya.com/blog/2018/05/essentials-of-deep-learning-trudging-into-unsupervised-deep-learning

(16) https://towardsdatascience.com/the-5-clustering-algorithms-data-scientists-need-to-know-a36d136ef68

(17) https://towardsdatascience.com/understanding-k-means-clustering-in-machine-learning-6a6e67336aa1https://towardsdatascience.com/gaussian-mixture-models-d13a5e915c8e