CLUSTER ANALYSIS ON FASHION MNIST DATASET USING UNSUPERVISED LEARNING

Project 3 - CSE 574 Intro to Machine learning

Krishna Naga Karthik BODAPATI
Department of Computer Science
(SUNY) University at Buffalo
Buffalo NY, 14221
kbodapat@buffalo.edu

Abstract

The Goal of this project is to classify Fashion MINST clothing images into 10 classes using Clustering.We used different types of clustering methods - K Means and Gaussian Mixture Model along with different types of Autoencoders

1 Introduction

1.1 Clustering:

Cluster analysis or clustering is the task of grouping a set of objects in such a way that objects in the same group (called a cluster) are more similar (in some sense) to each other than to those in other groups (clusters). It is a main task of exploratory data mining, and a common technique for statistical data analysis, used in many fields, including machine learning, pattern recognition, image analysis, information retrieval, bioinformatics, data compression, and computer graphics.

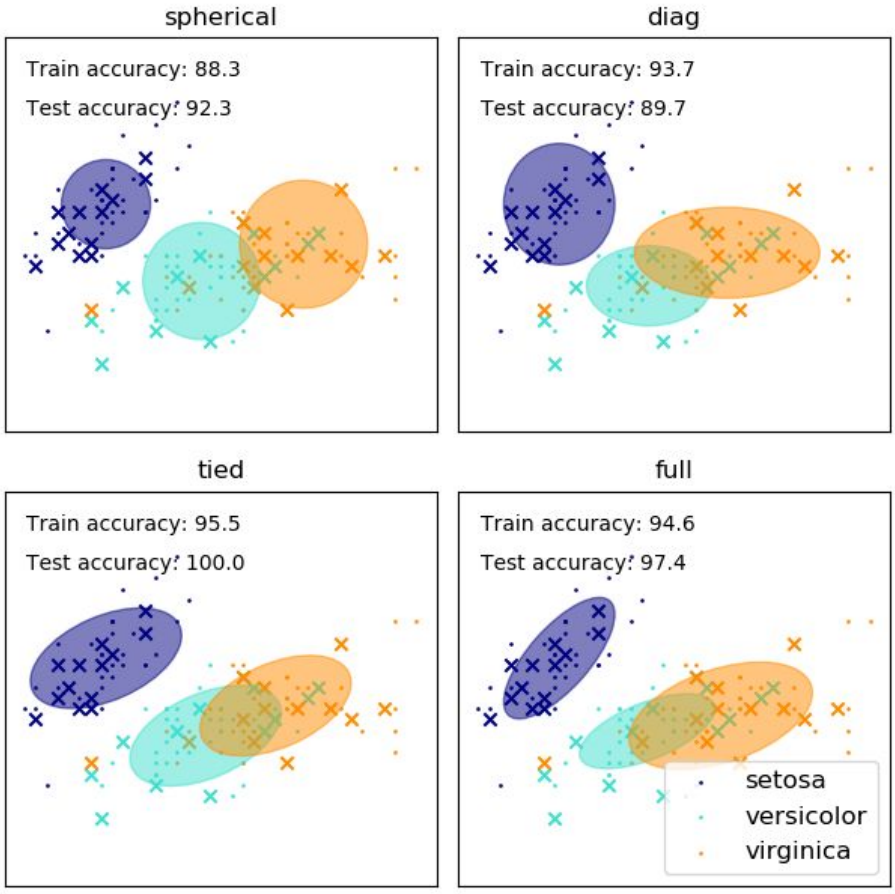
1.1.1 K means:

k-means clustering is a method of vector quantization, originally from signal processing, that is popular for cluster analysis in data mining. k-means clustering aims to partition n observations into k clusters in which each observation belongs to the cluster with the nearest mean, serving as a prototype of the cluster. This results in a partitioning of the data space into Voronoi cells. k-Means minimizes within-cluster variances (squared Euclidean distances), but not regular Euclidean distances, which would be the more difficult Weber problem: the mean optimizes squared errors, whereas only the geometric median minimizes Euclidean distances. Better Euclidean solutions can for example be found using k-medians and k-medoids.

1.1.2 Gaussian mixture model:

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 GaussianMixture.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.
The GaussianMixture comes with different options to constrain the covariance of the difference classes estimated: spherical, diagonal, tied or full covariance.



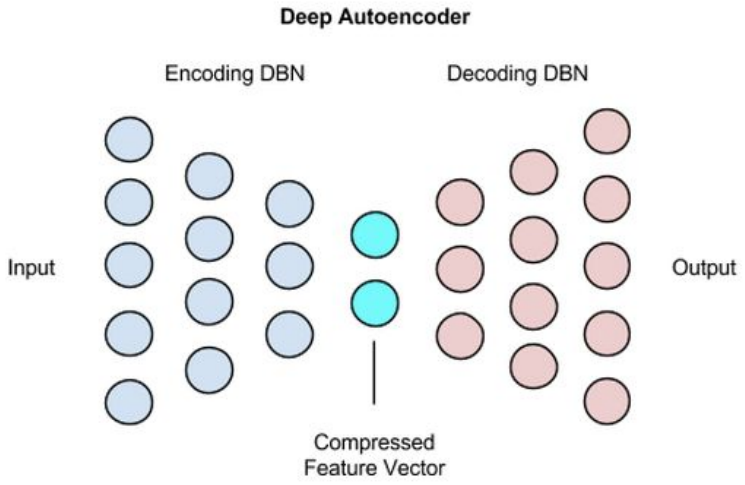
1.2 Auto Encoder:

An autoencoder is a type of artificial neural network used to learn efficient data codings in an unsupervised manner.[1] The aim of an autoencoder is to learn a representation (encoding) for a set of data, typically for dimensionality reduction, by training the network to ignore signal “noise”. Along with the reduction side, a reconstructing side is learnt, where the autoencoder tries to generate from the reduced encoding a representation as close as possible to its original input, hence its name. Several variants exist to the basic model, with the aim of forcing the learned representations of the input to assume useful properties. Examples are the regularized autoencoders (Sparse, Denoising and Contractive autoencoders), proven effective in learning representations for subsequent classification tasks, and Variational autoencoders, with their recent applications as generative models.Autoencoders are effectively used for solving many applied problems, from face recognition to acquiring the semantic meaning for the words.

1.2.1 Deep Auto Encoder:

A deep autoencoder is composed of two, symmetrical deep-belief networks that typically have four or five shallow layers representing the encoding half of the net, and the second set of four or five layers that make up the decoding half.

The layers are restricted Boltzmann machines, the building blocks of deep-belief networks, with several peculiarities that we'll discuss below. Here's a simplified schema of a deep autoencoder's structure,



1.2.2 Convolutional Autoencoder:

Convolutional AutoEncoders (CAEs) approach the filter definition task from a different perspective: instead of manually engineer convolutional filters we let the model learn the optimal filters that minimize the reconstruction error. These filters can then be used in any other computer vision task.CAEs are the state-of-art tools for unsupervised learning of convolutional filters. Once these filters have been learned, they can be applied to any input in order to extract features. These features, then, can be used to do any task that requires a compact representation of the input, like classification.CAEs are a type of Convolutional Neural Networks (CNNs): the main difference between the common interpretation of CNN and CAE is that the former are trained end-to-end to learn filters and combine features with the aim of classifying their input. In fact, CNNs are usually referred as supervised learning algorithms. The latter, instead, are trained only to learn filters able to extract features that can be used to reconstruct the input.

2 Dataset Definition

The Zalando's 'Fashion-MNIST' dataset comprises of 60000 gray-scale images each of size 28 x 28 (which is equivalent to 784 features per image) for training and another 10000 such images for testing the model. Intensity of each pixel in an image can vary between the integer values of 0 to 255. Each image belongs to one of the following ten classes listed below. A sample set of images is also present next to the list.

Class	Label
0	T-shirt/top
1	Trouser
2	Pullover
3	Dress
4	Coat

5	Sandal
6	Shirt
7	Sneaker
8	Bag
9	Ankle Boot

3 Pre-Processing

Since all the 784 features in an image are integer values with in 0 to 255, the data is somewhat evenly distributed. Nevertheless, bringing the data to a zero mean distribution and with magnitude less than 1 is computationally advantageous. The data is already partitioned into Training and Test sets and we are required to pre-process both the sets

As stated before a Neural Network needs each input to be in the form of a vector while the Convolutional Neural Network can handle input in 2D form. Hence for the One-layer and Multi-Layer NNs we are reshaping the input to a vector of 784 values.

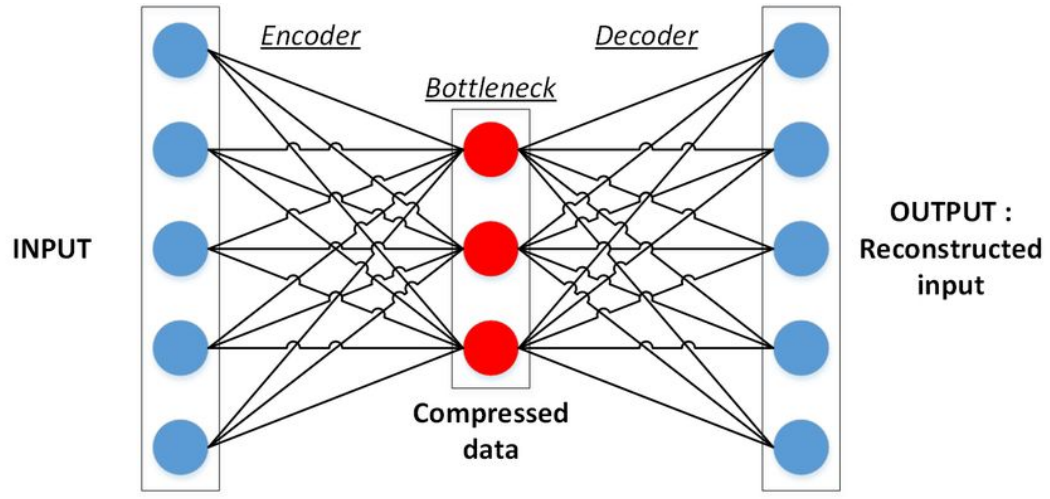
4 Model Architecture

To implement clustering using Autoencoder , we are considering three different architectures.

a) Single-hidden-layer autoencoder
b) Multi-hidden-layer autoencoder
c) Convolutional autoencoder

Applying this to both K means and GMM clustering, and training and testing data is same for all these models

4.1 Single Hidden layer Autoencoder



Basically, It is a single layer neural network but the output layer is same as input layer so during training instead of using (x_train, y_train) we use (x_train, x_train) here the size of bottleneck determines the dimension to which image is reduced in our model we used 128 nodes for bottleneck. And sigmoid function is used as activation

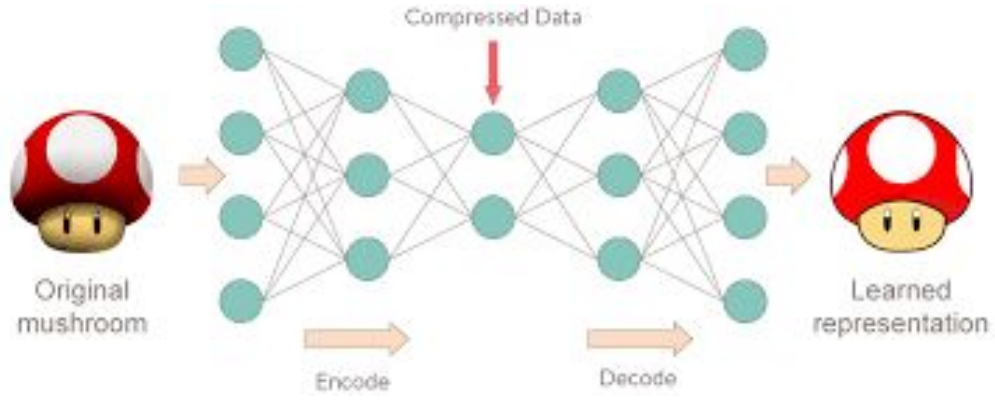
between input-bottleneck and bottleneck-output

```
Model: "model_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         (None, 784)               0
_________________________________________________________________
dense_1 (Dense)              (None, 128)               100480
_________________________________________________________________
dense_2 (Dense)              (None, 784)               101136
=================================================================
Total params: 201,616
Trainable params: 201,616
Non-trainable params: 0
_________________________________________________________________
```

4.2 Multi-Hidden-LayerAutoencoder:

This is similar to multi layer neural network, but like in single layer Autoencoder output layer is same as input layer, but for bottleneck we use multiple layers to learn complex features from image instead of single layer

In this example we used 6 layers in bottleneck (3 for encoding and 3 for decoding)



784 nodes -> 128 nodes -> 64 nodes -> 32 nodes -> 64 nodes -> 128 nodes -> 784 nodes

MODEL SUMMARY

```
Model: "model_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         (None, 784)               0
_________________________________________________________________
dense_1 (Dense)              (None, 128)               100480
_________________________________________________________________
dense_2 (Dense)              (None, 64)                8256
_________________________________________________________________
dense_3 (Dense)              (None, 32)                2080
_________________________________________________________________
dense_4 (Dense)              (None, 32)                1056
_________________________________________________________________
dense_5 (Dense)              (None, 64)                2112
_________________________________________________________________
dense_6 (Dense)              (None, 128)               8320
_________________________________________________________________
dense_7 (Dense)              (None, 784)               101136
=================================================================
Total params: 223,440
Trainable params: 223,440
Non-trainable params: 0
```

4.3 Convolutional Autoencoder:

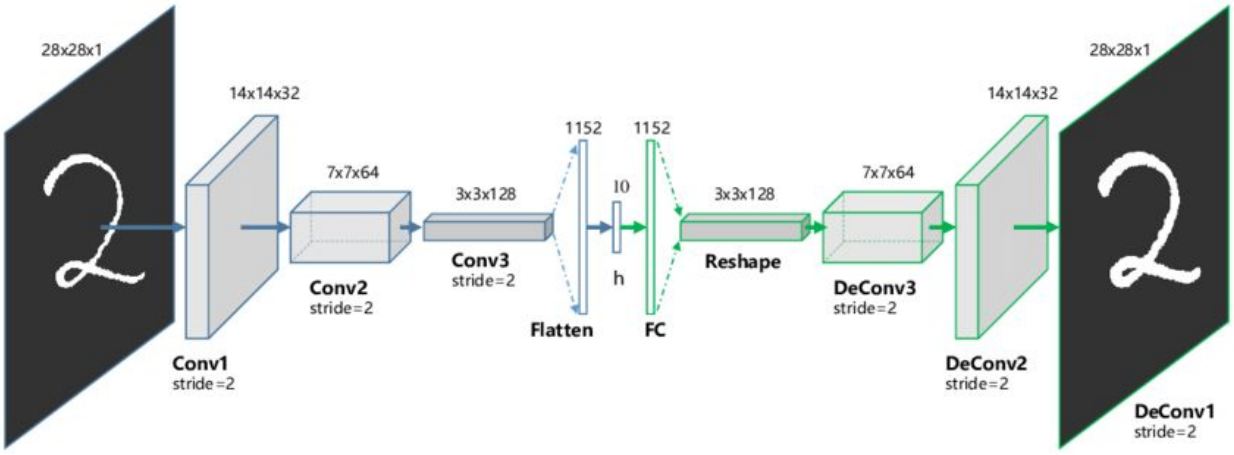
CNNs take 2D input image and pass through cycles of Convolution and Pooling kernels or filters. At the end the 2D data is flattened out and

4.3.1 Convolution operation is similar to dot product of two matrices. Convolutional kernel of a small size, say 2 x 2 matrix is used to compute dot products with patches in the input image. Convolution operation highlights the patterns in an image patch such as edges, outlines etc. Using multiple kernels in a layer helps to detect such elements in an image patch with more sharply. One can use number of kernels in the first layer (usually in the order of powers of 2 such as 32, 64, 128 etc. corresponding to dimensions of digital images in general). The output size of the convolution layer diminishes by a factor of the kernel size.

4.3.2 Activation Function: After every convolution an activation function is incorporated to introduce the necessary nonlinearity in the network. Without activation, the convolution becomes simple linear multiplications. Sigmoid, ReLU, Hyperbolic Tangent (tanh) etc. activation functions can be used. Our implementation makes use of sigmoid to have a similarity with the one and multi-layer neural networks.

4.3.3 Pooling layer: Output of convolution layer is passed on to a Pooling layer. Pool operation operates on a neighborhood of pixels reducing them to one pixel. Example operations can be max, min, average, sum etc. For example if we are using a max or min pooling filter the contrast between adjacent pixels is more compared to an

average pooling filter which tries to smoothen such contrasts. Pooling reduces the number of computational parameters and also helps to reduce overfitting.



The values that we use in each of the kernels essentially become the weights that we train. Once again we make use of Deep Learning frameworks such as Tensorflow and Keras to implement the CNN architecture. The number of kernels, size of each kernel, the pooling function, the loss function etc are hyper parameters which we choose as per the necessity.

Also, Keras provides a compile() method to choose the type of gradient descent and the hyperparameters that we can make use of for a particular type of descent algorithm.

Eg: sdg - stochastic gradient descent, RMSprop, adam etc.

Adam is an optimized variant of stochastic gradient with adaptive learning rate. It is computationally less expensive and yet has good performance than a normal stochastic gradient descent.

CONVOLUTIONAL AUTOENCODER MODEL SUMMARY

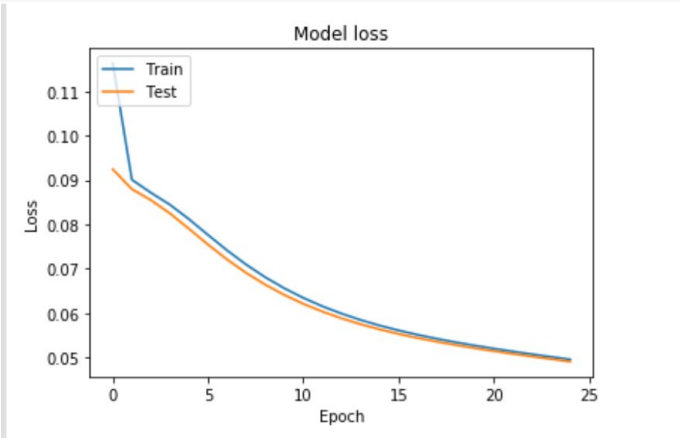
```
Model: "model_3"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_3 (InputLayer)         (None, 28, 28, 1)         0
_________________________________________________________________
conv2d_8 (Conv2D)            (None, 28, 28, 128)       1280
_________________________________________________________________
max_pooling2d_4 (MaxPooling2 (None, 14, 14, 128)       0
_________________________________________________________________
conv2d_9 (Conv2D)            (None, 14, 14, 64)        73792
_________________________________________________________________
max_pooling2d_5 (MaxPooling2 (None, 7, 7, 64)          0
_________________________________________________________________
```

```
conv2d_10 (Conv2D)           (None, 7, 7, 32)          18464
_________________________________________________________________
max_pooling2d_6 (MaxPooling2 (None, 4, 4, 32)          0
_________________________________________________________________
conv2d_11 (Conv2D)           (None, 4, 4, 32)          9248
_________________________________________________________________
up_sampling2d_4 (UpSampling2 (None, 8, 8, 32)          0
_________________________________________________________________
conv2d_12 (Conv2D)           (None, 8, 8, 64)          18496
_________________________________________________________________
up_sampling2d_5 (UpSampling2 (None, 16, 16, 64)        0
_________________________________________________________________
conv2d_13 (Conv2D)           (None, 14, 14, 128)       73856
_________________________________________________________________
up_sampling2d_6 (UpSampling2 (None, 28, 28, 128)       0
_________________________________________________________________
conv2d_14 (Conv2D)           (None, 28, 28, 1)         1153
=================================================================
Total params: 196,289
Trainable params: 196,289
Non-trainable params: 0
_________________________________________________________________
```

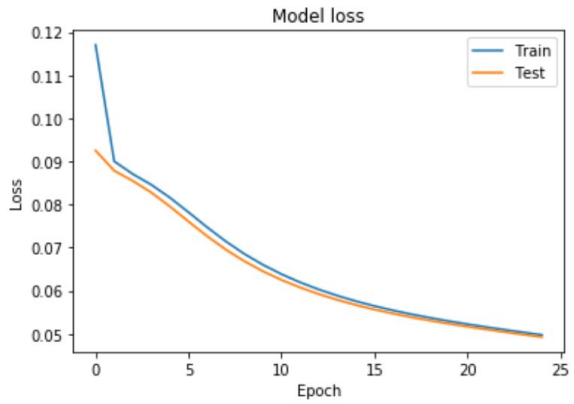
5 Results

Plots:

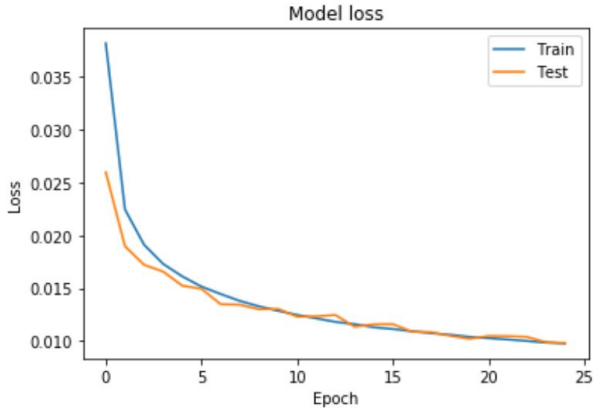
Single Layer Auto Encoder



Multi Layer Auto Encoder



Convolutional Auto Encoder



Task 1 - K Means

Confusion Matrix for Test Data (After Mapping labels):

```
array([[587,  29,   0,   5,  34,  93, 245,   1,   6,   0],
       [ 50, 890,   0,   0,   9,  22,  29,   0,   0,   0],
       [ 19,   4,   0,   4, 566,  61, 342,   0,   4,   0],
       [277, 503,   0,   3,  10,  92, 113,   0,   2,   0],
       [136,  27,   0,   5, 627,  42, 159,   0,   4,   0],
       [ 44,   0,   0,   0,   0, 651,   6, 228,   0,  71],
       [189,  12,   0,   0, 311, 115, 358,   0,  15,   0],
       [  2,   0,   0,   0,   0,  62,   0, 789,   0, 147],
       [  4,   6,   0, 408,  62,  84,  36,  41, 351,   8],
       [412,   0,   0,   2,   0,  29,   4,  25,   0, 528]])
```

Test Accuracy - 0.4784

Train Accuracy - 0.46875

Task 2 - K Means with Auto Encoder

Single Layer Auto Encoder:

Confusion Matrix for Test Data (After Mapping labels):

```
array([[659,  41,  66, 186,   0,   2,  36,   1,   9,   0],
       [ 37, 919,  13,  25,   0,   0,   5,   0,   1,   0],
       [ 27,   4, 674,  19,   0,   2, 254,   0,  19,   1],
       [221, 590,  16, 149,   0,   1,  21,   0,   2,   0],
       [131,  42, 685,  29,   0,   2,  95,   0,  16,   0],
       [  0,   0,   0,   6, 134, 316,   5, 449,   3,  87],
       [232,  18, 405,  77,   0,  14, 222,   0,  32,   0],
       [  0,   0,   0,   0, 205,   6,   0, 782,   0,   7],
       [  3,   1,  49,  59, 112,  28,  99,  10, 474, 165],
       [  1,   0,   0,   2, 340,  38,  14,  19,  38, 548]])
```

Test Accuracy - 0.4743

Train Accuracy - 0.47965

Multi Layer Auto Encoder:

Confusion Matrix for Test Data (After Mapping labels):

```
array([[706,  37,  31,   0,  58,   2, 155,   1,  10,   0],
       [ 53, 906,   7,   0,  11,   0,  22,   0,   1,   0],
       [ 42,   4, 255,   1, 661,   1,  21,   0,  15,   0],
       [276, 569,  18,   1,  14,   0, 122,   0,   0,   0],
       [166,  34, 106,   0, 653,   1,  31,   0,   9,   0],
       [  0,   0,   4, 134,   0, 482,   7, 280,   9,  84],
       [252,  16, 227,   3, 387,   5,  84,   0,  26,   0],
       [  0,   0,   0,   3,   0,  44,   0, 819,   0, 134],
       [  3,   3,  24, 186,  72,  11,  86,  13, 483, 119],
       [  1,   0,   8, 385,   0,   6,   4,  39,  52, 505]])
```

Test Accuracy - 0.4894

Train Accuracy - 0.48258333333333

Convolutional Auto Encoder:

Confusion Matrix for Test Data (After Mapping labels):

```
array([[481,  24, 183,   0,  33,   2, 270,   1,   6,   0],
       [ 33, 819,  23,   0,   6,   0, 119,   0,   0,   0],
       [ 11,   3, 325,   0, 433,   1, 223,   0,   4,   0],
       [278, 428,  59,   0,   8,   0, 226,   0,   1,   0],
       [103,  29, 265,   0, 492,   0, 103,   0,   8,   0],
       [  0,   0,   1,   0,   0, 418,   6, 499,   1,  75],
       [138,  14, 287,   0, 226,   1, 320,   0,  14,   0],
       [  0,   0,   0,   0,   0,  19,   1, 740,   0, 240],
       [  5,   6, 171,   0,  44,  45, 110,  38, 573,   8],
       [  0,   1,   3,   0,   0,  89,   0,   8,   1, 898]])
```

Test Accuracy - 0.5066

Train Accuracy - 0.50455

Task 3 - GMM with Auto Encoder

Single Layer Auto Encoder:

Confusion Matrix for Test Data (After Mapping labels):

```
array([[465,   0,  11,  51, 169,   4, 296,   0,   4,   0],
       [  7, 644,   2, 269,  49,   0,  29,   0,   0,   0],
       [ 29,   0, 511,   1, 233,   0, 217,   0,   9,   0],
       [169,  65,   2, 470, 181,   1, 112,   0,   0,   0],
       [138,   0, 520,  20, 137,   1, 177,   0,   7,   0],
       [  0,   0,   0,   0,   1, 582,   0, 402,   4,  11],
       [136,   2, 272,  14, 234,   4, 326,   0,  12,   0],
       [  0,   0,   0,   0,   0,  51,   0, 718,   2, 229],
       [  2,   0,  13,   0, 344, 112,   3,   3, 522,   1],
       [  0,   0,   0,   0,   3, 165,   0,   9, 274, 549]])
```

Test Accuracy - 0.4924

Train Accuracy - 0.49695

Multi Layer Auto Encoder:

Confusion Matrix for Test Data (After Mapping labels):

```
array([[666,  64,  23,   7,  61,   0, 176,   0,   3,   0],
       [ 32, 924,   8,   1,   6,   0,  28,   0,   1,   0],
       [ 53,   4, 692,   3, 116,   0, 130,   0,   2,   0],
       [202, 602,   8,   1,  15,   0, 172,   0,   0,   0],
       [165,  35, 639,   1,  49,   0, 109,   0,   2,   0],
       [  0,   0,   0,  65,   0, 391,   2, 512,   5,  25],
       [241,  24, 409,  11, 102,   0, 206,   0,   7,   0],
       [  0,   0,   0,   2,   0,  11,   0, 950,   0,  37],
       [  3,   0,  24,  35, 301,   2,  80,  13, 541,   1],
       [  0,   0,   0, 100,   1,  27,   0, 124,   0, 748]])
```

Test Accuracy - 0.5168

Train Accuracy - 0.5188666666666666667

Convolutional Auto Encoder:

Confusion Matrix for Test Data (After Mapping labels):

```
array([[627,  36, 124,  19,   4, 111,  33,   0,  46,   0],
       [ 22, 792,   4, 141,   3,  11,  24,   0,   3,   0],
       [  7,   7, 275,   0, 509,  51,  14,   0, 137,   0],
       [128, 603,  48, 114,   8,  58,  21,   0,  20,   0],
       [  9,  79, 143,   5, 578,  38,  31,   0, 117,   0],
       [  0,   0,   0,   0,   0, 440,  40, 514,   0,   6],
       [214,  14, 248,   9, 312,  59,  53,   0,  91,   0],
       [  0,   0,   0,   0,   0,   5,  43, 906,   0,  46],
       [  2,   0,   0, 165,   1,  85, 134,   3, 610,   0],
       [  0,   0,   0,   0,   0,  44,  69,  87,   0, 800]])
```

Test Accuracy - 0.5195

Train Accuracy - 0.51345

6 Conclusion

We tried different clustering methods (K means and Gaussian Mixture model) using different types of autoencoders (Single layer, Deep Autoencoder, Convolutional Autoencoder). We got the best accuracy for both training data and testing data by clustering using Gaussian Mixture Model with Convolutional Autoencoder

7 References (for writing Report)

1. Wikipedia - Clustering, K means, GMM , autoencoders - Definitions
2. SKlearn - Gaussian Mixture Models
3. https://skymind.ai/wiki/deep-autoencoder - Deep autoencoder theory
4. https://pgaleone.eu/neural-networks/2016/11/24/convolutional-autoencoders/ - Convolution autoencoder
5. https://www.researchgate.net/figure/Basic-architecture-of-a-single-layer-autoencoder-made-of-an-encoder-going-from-the-input_fig3_333038461 - Single layer autoencoder Image
6. https://towardsdatascience.com/deep-autoencoders-using-tensorflow-c68f075fd1a3 - Deep Autoencoder Image
7. https://www.researchgate.net/figure/The-structure-of-proposed-Convolutional-AutoEncoders-CAE-for-MNIST-In-the-middle-there_fig1_320658590 - Convolutional autoencoder Image

8 References (for Code)

1. https://www.datacamp.com/community/tutorials/autoencoder-keras-tutorial
2. https://blog.keras.io/building-autoencoders-in-keras.html