Clustering using K-Means and Gaussian Mixture Model

Chan Shu You
Department of Computer Science
University at Buffalo
chanshuy@buffalo.edu

Abstract

When a target is given for predicting an input, it can easily be predicted using a supervised learning method. But what if a target is not given? This is when an unsupervised learning method comes to play. One of the most favorable methods is clustering. When a set of data without labels are given, we predict the data by grouping them to different sets of clusters, which is one of the hyper-parameters for training. For this project, we use K-Means and Gaussian Mixture Model to cluster the data.

1. Introduction

K-Means and Gaussian Mixture are two popular unsupervised learning methods in machine learning. It comes handy when the target is not given to input training data. Both K-Means and Gaussian Mixture Model consist two big steps – Expectation (E) step and Maximization (M) step; it is also called EM algorithm. During E step, we calculate the mean square error of each data from the initialized k points, and assign those points to the closest k value. We then move to M step, where the k centers are updated based on the mean of each clusters. Gaussian Mixture Model, although similar to K-Means, has a great advantage over K-Means. While K-Means can only luster data in circle, Gaussian Mixture Model can cluster the data in different shape.

2. Dataset

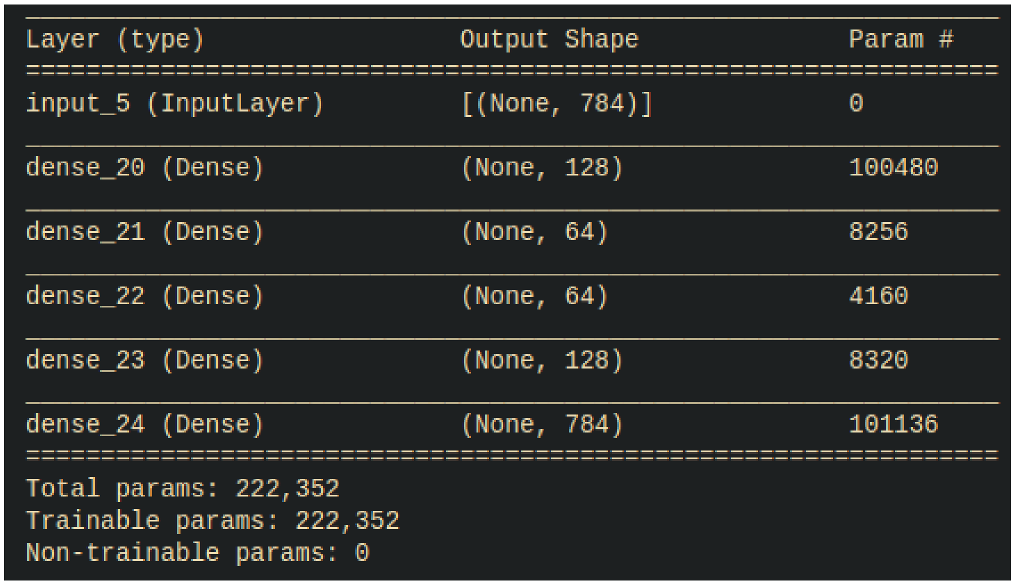
This project uses MNIST fashion images for datasets. There are two groups of datasets – Training and Testing. Training dataset contains 60,000 images, and the testing dataset contains 10,000 datasets. The dataset consist of 10 different classes of images.

3. Preprocessing

Before the dataset are used for inputs, we need to either convert each image to a 784 array of 28x28 matrix, depends on the training model. Since the dataset is image, the input dataset needs to be normalized in a range of 0 to 1. For the second model, since the first layer starts with Convolutional layer, it is necessary to convert the data to 4 dimensions, which is (batchx28x28x1).

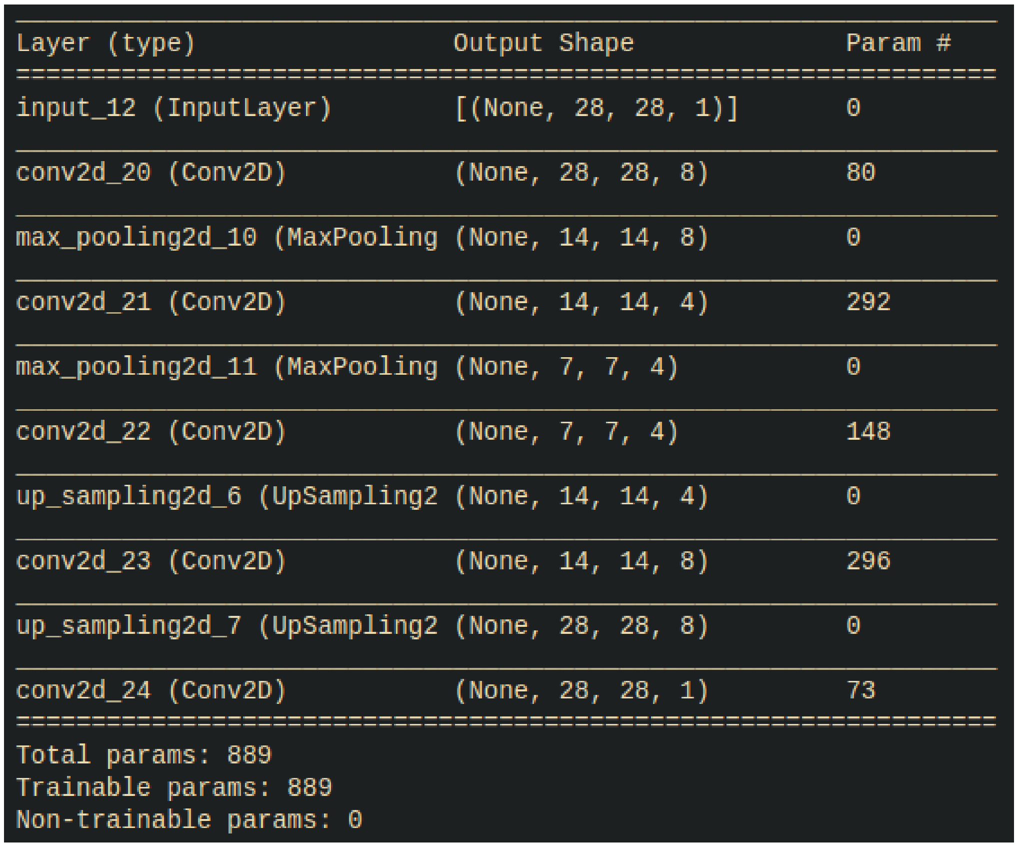
4. Architecture

This project uses two different auto-encoder models.
The following is the architecture of the first model, which is named “Flat-Model” in this project:



```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_5 (InputLayer)         [(None, 784)]             0
_________________________________________________________________
dense_20 (Dense)             (None, 128)               100480
_________________________________________________________________
dense_21 (Dense)             (None, 64)                8256
_________________________________________________________________
dense_22 (Dense)             (None, 64)                4160
_________________________________________________________________
dense_23 (Dense)             (None, 128)               8320
_________________________________________________________________
dense_24 (Dense)             (None, 784)               101136
=================================================================
Total params: 222,352
Trainable params: 222,352
Non-trainable params: 0
```

The following is the architecture of the second model, which is named “Conv-Model” in this project:



```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_12 (InputLayer)        [(None, 28, 28, 1)]       0
_________________________________________________________________
conv2d_20 (Conv2D)           (None, 28, 28, 8)         80
_________________________________________________________________
max_pooling2d_10 (MaxPooling (None, 14, 14, 8)         0
_________________________________________________________________
conv2d_21 (Conv2D)           (None, 14, 14, 4)         292
_________________________________________________________________
max_pooling2d_11 (MaxPooling (None, 7, 7, 4)           0
_________________________________________________________________
conv2d_22 (Conv2D)           (None, 7, 7, 4)           148
_________________________________________________________________
up_sampling2d_6 (UpSampling2 (None, 14, 14, 4)         0
_________________________________________________________________
conv2d_23 (Conv2D)           (None, 14, 14, 8)         296
_________________________________________________________________
up_sampling2d_7 (UpSampling2 (None, 28, 28, 8)         0
_________________________________________________________________
conv2d_24 (Conv2D)           (None, 28, 28, 1)         73
=================================================================
Total params: 889
Trainable params: 889
Non-trainable params: 0
```

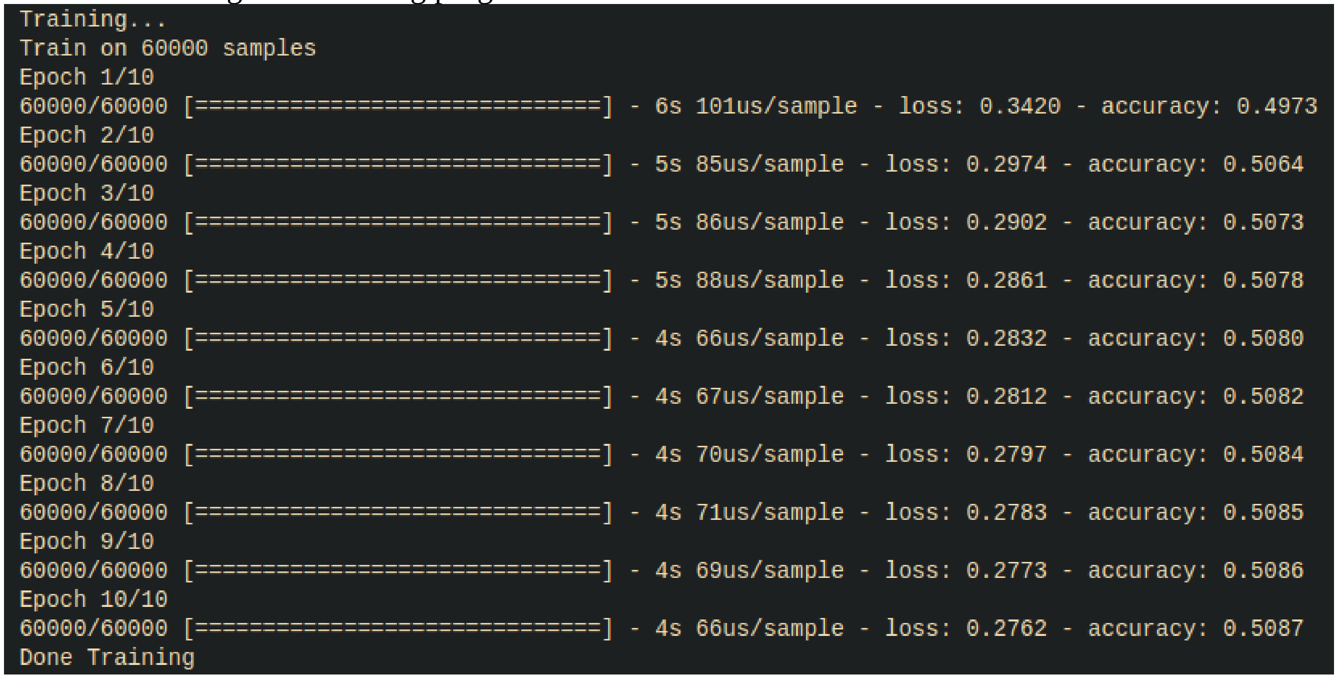
To compare and contrast the two models, the Flat model consist of only dense layers, while the Conv model only consist Convolutional layers.

5. Results

5.1 Training:

Flat Model:

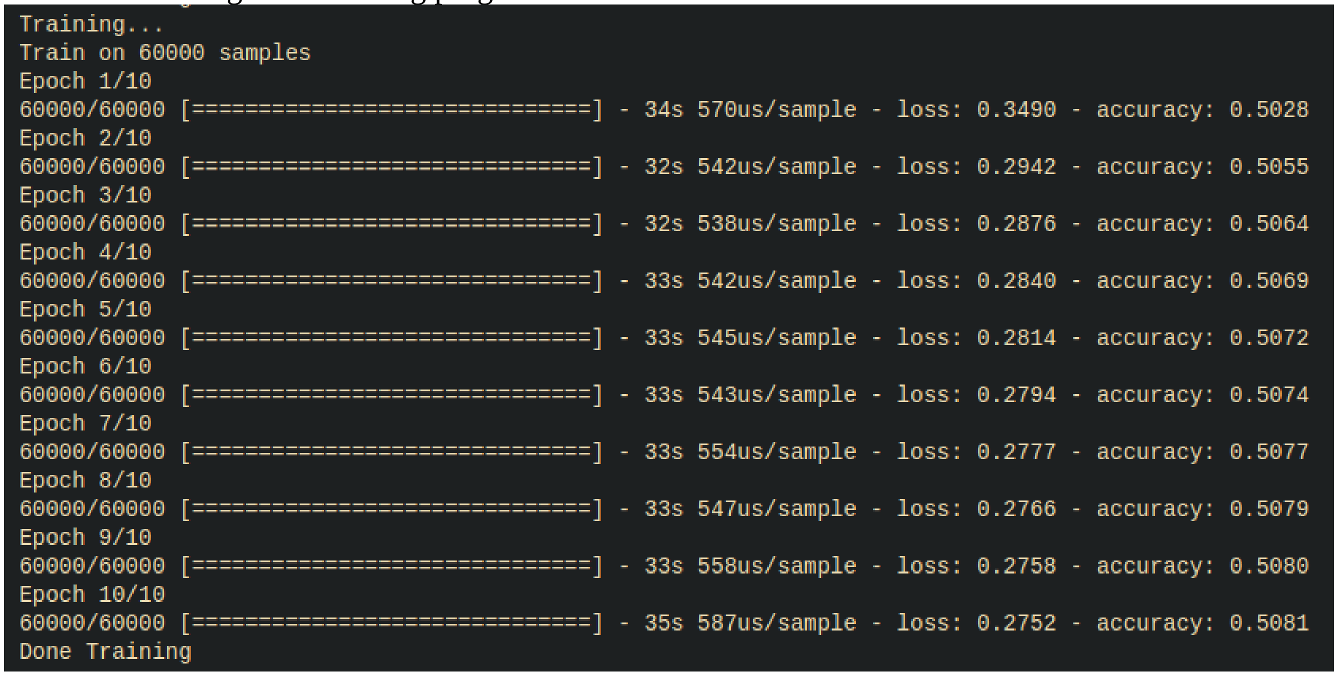
The following is the training progress of Flat Model:



```
Training...
Train on 60000 samples
Epoch 1/10
60000/60000 [==============================] - 6s 101us/sample - loss: 0.3420 - accuracy: 0.4973
Epoch 2/10
60000/60000 [==============================] - 5s 85us/sample - loss: 0.2974 - accuracy: 0.5064
Epoch 3/10
60000/60000 [==============================] - 5s 86us/sample - loss: 0.2902 - accuracy: 0.5073
Epoch 4/10
60000/60000 [==============================] - 5s 88us/sample - loss: 0.2861 - accuracy: 0.5078
Epoch 5/10
60000/60000 [==============================] - 4s 66us/sample - loss: 0.2832 - accuracy: 0.5080
Epoch 6/10
60000/60000 [==============================] - 4s 67us/sample - loss: 0.2812 - accuracy: 0.5082
Epoch 7/10
60000/60000 [==============================] - 4s 70us/sample - loss: 0.2797 - accuracy: 0.5084
Epoch 8/10
60000/60000 [==============================] - 4s 71us/sample - loss: 0.2783 - accuracy: 0.5085
Epoch 9/10
60000/60000 [==============================] - 4s 69us/sample - loss: 0.2773 - accuracy: 0.5086
Epoch 10/10
60000/60000 [==============================] - 4s 66us/sample - loss: 0.2762 - accuracy: 0.5087
Done Training
```

Conv Model:

The following is the training progress for the model:



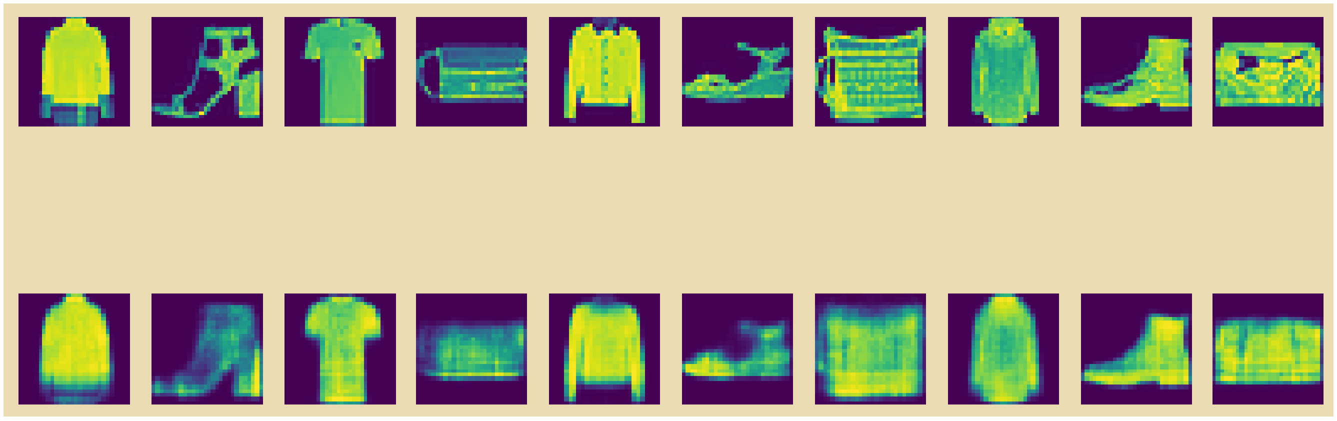
```
Training...
Train on 60000 samples
Epoch 1/10
60000/60000 [==============================] - 34s 570us/sample - loss: 0.3490 - accuracy: 0.5028
Epoch 2/10
60000/60000 [==============================] - 32s 542us/sample - loss: 0.2942 - accuracy: 0.5055
Epoch 3/10
60000/60000 [==============================] - 32s 538us/sample - loss: 0.2876 - accuracy: 0.5064
Epoch 4/10
60000/60000 [==============================] - 33s 542us/sample - loss: 0.2840 - accuracy: 0.5069
Epoch 5/10
60000/60000 [==============================] - 33s 545us/sample - loss: 0.2814 - accuracy: 0.5072
Epoch 6/10
60000/60000 [==============================] - 33s 543us/sample - loss: 0.2794 - accuracy: 0.5074
Epoch 7/10
60000/60000 [==============================] - 33s 554us/sample - loss: 0.2777 - accuracy: 0.5077
Epoch 8/10
60000/60000 [==============================] - 33s 547us/sample - loss: 0.2766 - accuracy: 0.5079
Epoch 9/10
60000/60000 [==============================] - 33s 558us/sample - loss: 0.2758 - accuracy: 0.5080
Epoch 10/10
60000/60000 [==============================] - 35s 587us/sample - loss: 0.2752 - accuracy: 0.5081
Done Training
```

As we can see, the accuracy for both model barely passed over 50, and stayed mostly constant throughout the training.

5.2 Decoder

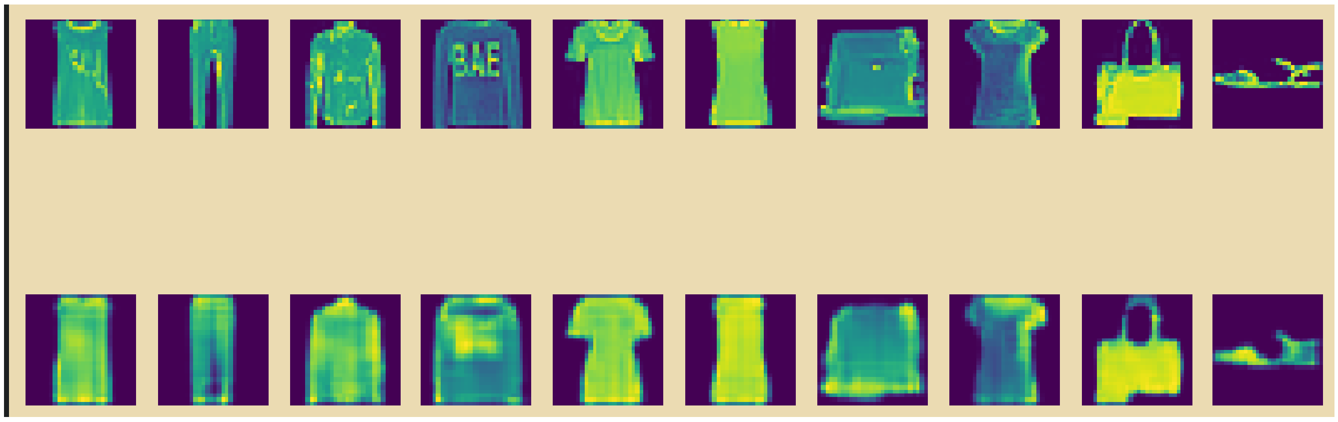
Flat Model:

The following is a comparison between 10 original random data and 10 decoded data after the model is trained:



Conv Model:

The following is a comparison between 10 original random data and 10 decoded data after the model is trained:



As we can see from the results, Flat Model is slightly less accurate than Conv Model, but both models can perfectly decode the data back to a very similar format to the original image.
Thus, for K-Means and Gaussian Mixture Model, the Flat Model will be used.

5.3 Encoder

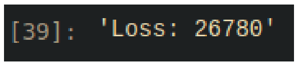
If we go back to the architecture parts, we can see from the architecture that flat model encodes the data into an array size of 64, and conv model converts the input image from size of 28x28 to 7x7 with 4 kernel layers.

5.3.1 K-Mean Clustering without Encoding

When K-Mean is implemented with training data, K-Means does not know which label belongs to which images. Thus, it is mandatory to plot out images by labels and manually figure out which label belongs to which. For this project, 20 random images are plotted from random range of data. After several plotting, it is decided that the following index from the predicted label belongs to the following actual label:

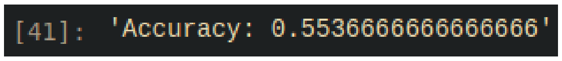
Predicted Label	Actual Label
0	0 - T-shirt
1	5 - Sandal
2	8 - Bag
3	1 - Trouser
4	7 - Sneaker
5	9 - boot
6	9 - boot
7	4 - Coat
8	6 - Shirt
9	8 - Bag

The array is thus converted based on the matching label. K-Mean is fit with training dataset, and after comparing the training data target and the prediction, the following number of labels did not match:



```
[39]:  'Loss: 26780'
```

which means that the accuracy is:



```
[41]:  'Accuracy: 0.5536666666666666'
```

5.3.2 K-Means Clustering with Encoding

This sections test both model on how good each of them can cluster using both K-Mean and Gaussian Mixture Model. The procedure will consist of:

1. Match label – Since both K-means and Gaussian Mixture Model do not know which label belongs to which class, it is necessary to manually plot the data and categorize the label

2. Calculate the loss and accuracy – the number of mismatch data / total data will be the loss, and the accuracy will be the opposite

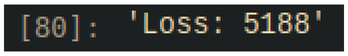
5.3.2.1 Flat Model

1. K-Means

Predicted Label	Actual Label
0	4- Coat
1	0 - T-shirt
2	5 - Sandal
3	7 - Sneaker
4	1 - Trouser
5	9 - Boots
6	6 - Shirt
7	8 - Bag
8	0 - T-shirt
9	9 - Boots

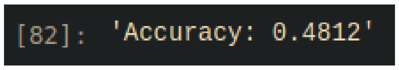
The following is the result of implementing K-Means clustering with encoded data:

Loss:



```
[80]: 'Loss: 5188'
```

Accuracy:



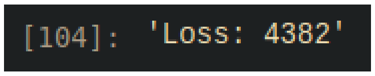
```
[82]: 'Accuracy: 0.4812'
```

2. Gaussian Mixture Model

Predicted Label	Actual Label
0	8 - Bag
1	0 - T-shirt
2	5 - Sandal
3	4 - Coat
4	7 - Sneaker
5	9 - Boots
6	3 - Dress
7	9 - Boots
8	8 - Bag
9	1 - Trouser

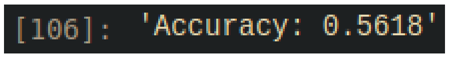
The following is the result of implementing Gaussian Mixture clustering with encoded data:

Loss:



```
[104]:  'Loss: 4382'
```

Accuracy:



```
[106]:  'Accuracy: 0.5618'
```

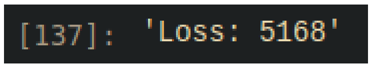
5.3.2.2 Conv Model

1. K-Means

Predicted Label	Actual Label
0	0 - T-shirt
1	9 - Boots
2	8 - Bags
3	1 - Trouser
4	9 - Boots
5	5 - Sandals
6	6 - Shirts
7	4 - Coat
8	0 - T-shirts
9	3 - Dress

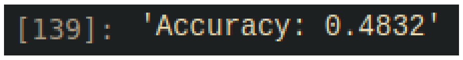
The following is the result of implementing K-Means clustering with encoded data:

Loss:



```
[137]:  'Loss: 5168'
```

Accuracy:



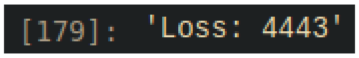
```
[139]:  'Accuracy: 0.4832'
```

2. Gaussian Mixture Model

Predicted Label	Actual Label
0	8 - Bag
1	4 - Coat
2	9 - Boots
3	3 - Dress
4	5 - Sandal
5	7 - Sneakers
6	0 - T-shirt
7	8 - Bag
8	9 - Boots
9	1 - Trouser

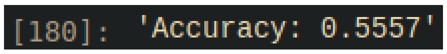
The following is the result of implementing Gaussian Mixture clustering with encoded data:

Loss:



```
[179]:  'Loss: 4443'
```

Accuracy:



```
[180]:  'Accuracy: 0.5557'
```

Conclusion

As we can see from the loss and accuracy values, the highest accuracy was 56.18% using the Flat Model to implement Gaussian Mixture Model.