Assignment-2

K-Means clustering

Varshith Vootla (Varshith@buffalo.edu)

UBID: 50365537

# Abstract

Implementation of k-means algorithm to do multi-class classification on the CIFAR-10 dataset. CIFAR-10 dataset consists of images of size 32 x 32 with 3 channels which makes the image size as 32 x 32 x 3. To make 10 clusters we assume initial 10 points from the given data randomly as centroids and then make iterations to find the distance between all the data set features and find the minimum points.

# Dataset

CIFAR-10 dataset has a total of 60,000 samples of which 50,000 are in the training set and 10,000 are in the test set and each image is of 1024 pixels. Each pixel values lies between 0-255. We are using only the test data to do clustering.

# Preprocessing & Normalization

As the images are RGB we first convert them into gray scale to make it of size 32 x 32 x 1 and then reshape them into 1024 sized array. All the pixel values lie between 0 and 255 and we will not have any outliers so we don’t need to convert to normalize the pixel values.

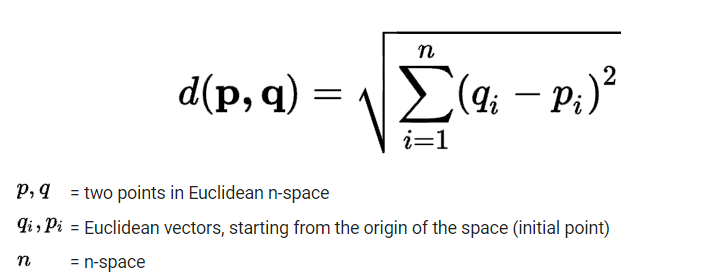
# Implementation

The idea of k-means clustering is to find the initial points from the given data set. Here in our data we take 10 points as we know there are 10 clusters in the dataset. From the initial 10 points we calculate the Euclidean distance for all the 10,000 set of data.

1. Take 10 random points and pick it from the data set which are to be used as initial centroids.
2. Calculate Euclidean distances of all the data points from the initial centroids.
3. Make each data point assign to a cluster based on distances obtained from step 2.
4. Now we take these initial assignments and run the same process as above with 50 iterations to make the clustering optimizable.
5. We will stop iterations when centroid doesn’t change.

## Calculation of distances:

Back implementation of Euclidean distance calculation is

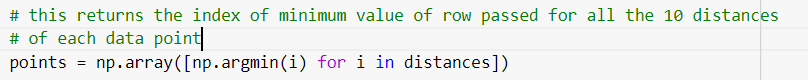


This is implemented from the cdist function imported from ‘scipy.spatial.distance’ library.



It returns as shape of (10000, 10). Each data point of test data set is combined with each of the centroids to return a distance. So, we have 10 distances for each data point of test dataset.

Now for each datapoint we need to take the minimum valued distance index (in range 0 to 9). For all the 10000 data points we take these indices which makes the data point belongs to that cluster.



We obtain distances between the test dataset used and initial assumed centroids

This is to be repeated for 50 iterations with initial assumed clusters and we find the best possible clusters

# Model Validation:

Silhouette score and Dunn index are used to validate the data clustering

## Silhouette score:

## It is a metric with range [-1, 1] with highest value reasons to very good clustering and lowest value -1 leads to bad clustering.

For this assignment we are expecting the silhouette score to be greater than 0.054

## Dunn index:

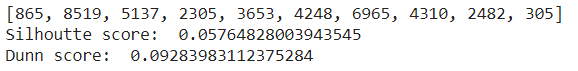
It is metric to identify the sets of clusters that are compact and having less variance between the clusters

For this assignment we are expecting the Dunn index to be greater than 0.089

Libraries used to implement above indices

from validclust import dunn

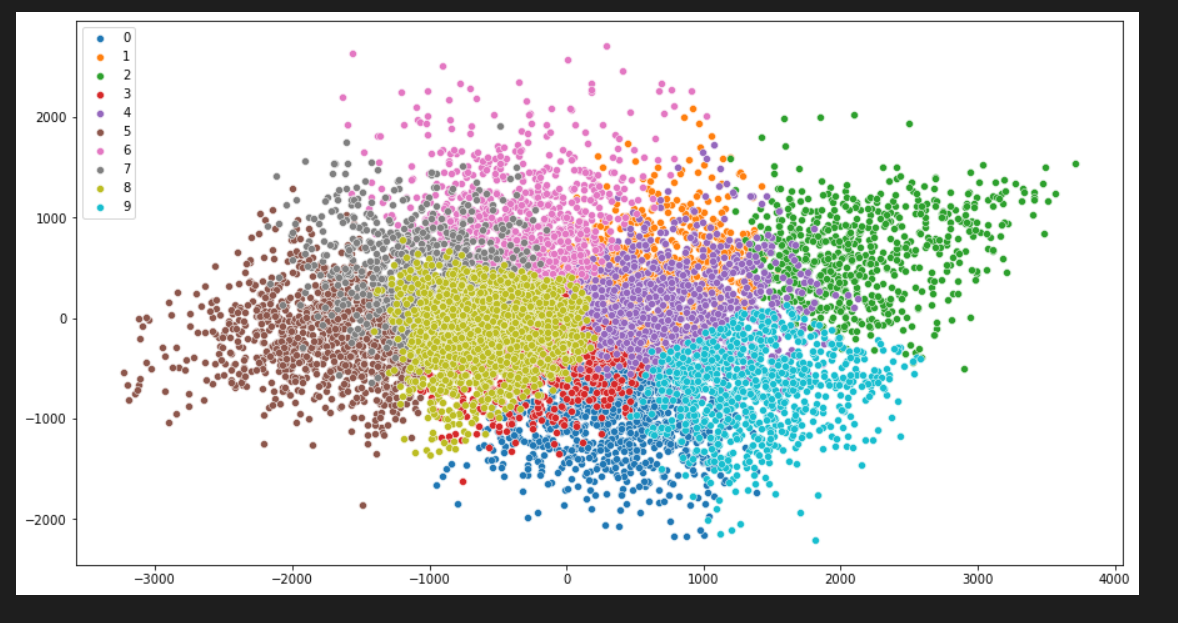
from sklearn.metrics import silhouette\_samples, silhouette\_score



For these particular initial random points, we are getting

**Silhouette score: 0.057**

**Dunn score: 0.092**

****

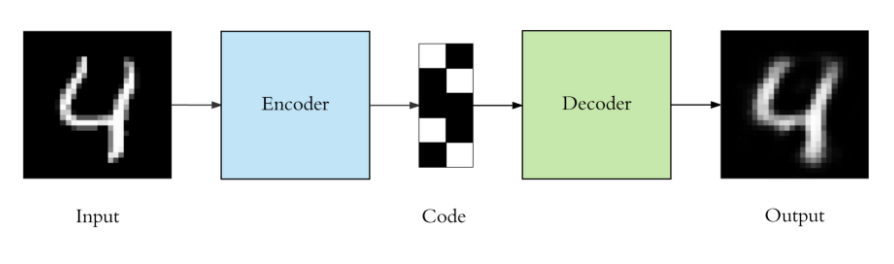
# Conclusion:

For the cifar dataset test data after k means clustering implementation we got the **Silhoutte score – 0.057 and Dunn index – 0.092**

# Auto Encoder Implementation:

Auto encoders are used to reconstruct input images by first encoding the given image with less no. of layers and then decoded to produce output of original size.

Encoded images are sparse representations which reduces the dimensionality of input



A simple auto encoder consists of an encoder layer and a decoded layer which is output layer.

For this project we are building a ‘Deep Auto Encoder’ which has 2 encoding layers and 2 decoding layers. We will generate sparse representations with this auto encoder.

Sparse representations are the outputs generated by Bottle neck. A bottle neck is the last layer of encoding i.e, the lowest possible dimensions of input image. These sparse representations are then fed into decoder layers to generate the output.

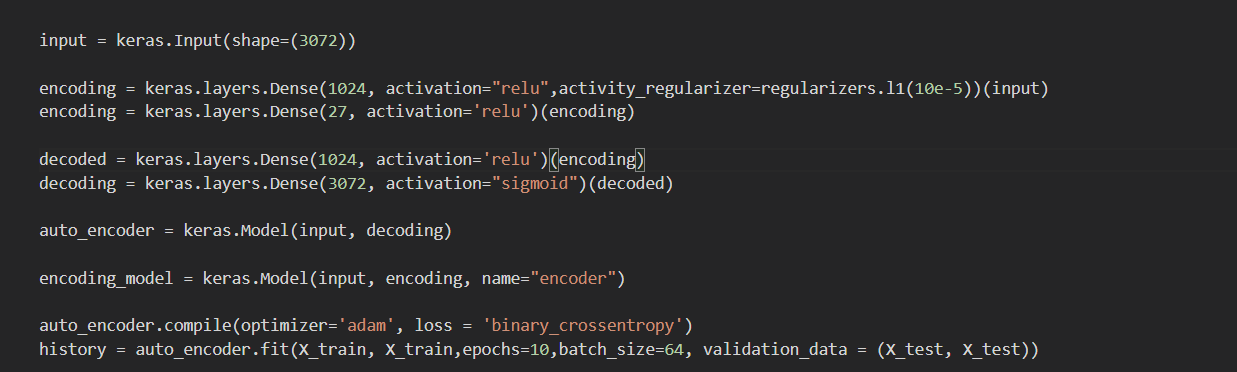
**Dataset:**

We are using the train data of cifar dataset. It consists of 50000 sample images of size 32 x 32 x 3. As we are using the auto encoder we are normalizing the pixel values with (data/255) to make all the pixel values lie in between 0 and 1

The image size is then reshaped into 32 x 32 x 3 = 3072

**Implementation:**

For the encoder we are passing the input with size 3072 which is created with keras.Input



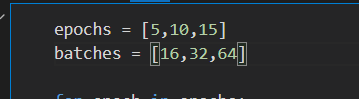
The input size is 3072 as mentioned earlier

1. We are encoding the image of size 1024 in the first encoding layer
2. Adding the regularizer l1 to have the sparsity constraint
3. Then we are moving into next layer with size 27. This is the bottleneck and lowest possible dimension of image in this model.
4. Images generated in the bottleneck are the sparse representation which are used to compare with original and decoded inputs
5. Then we move on to decoder to again increase the size to 1024.
6. In the last step make the size to the original input which is 3072

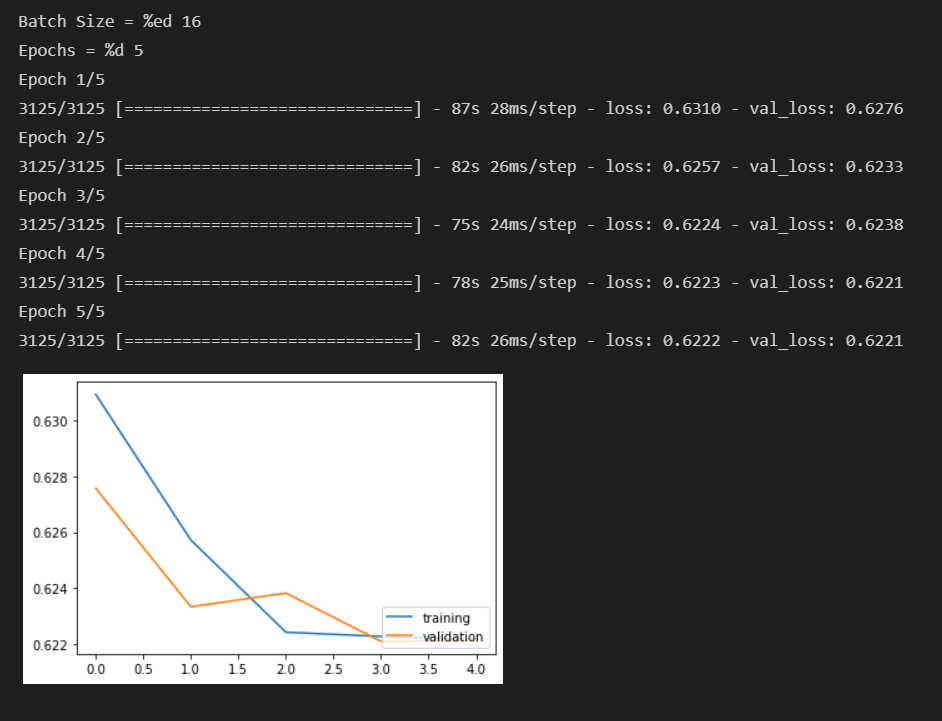
The decoder should always have the size of original input.

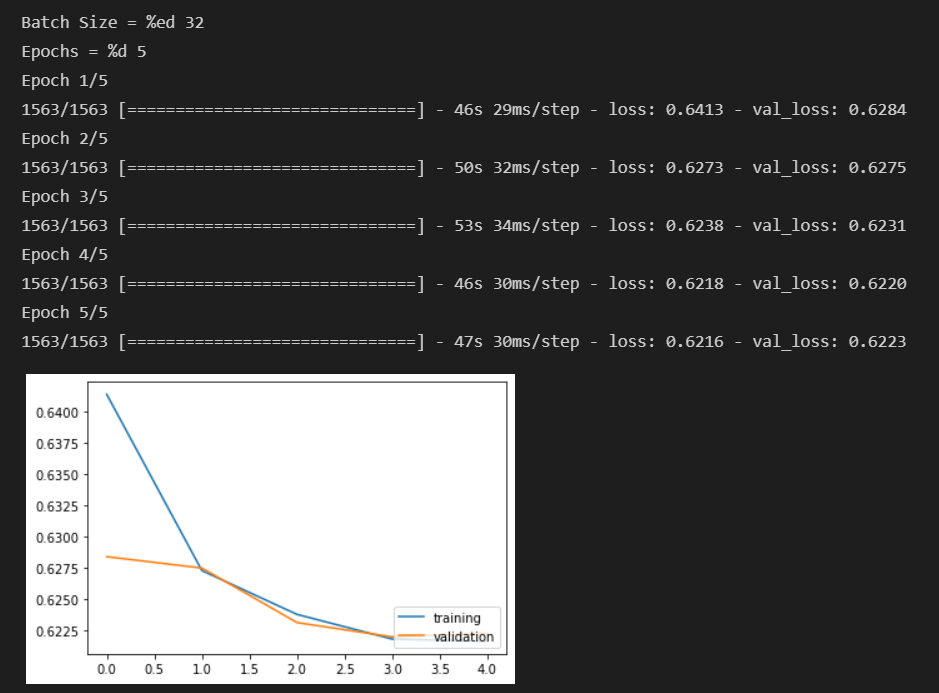
We can build an auto encoder with these encoder and decoder

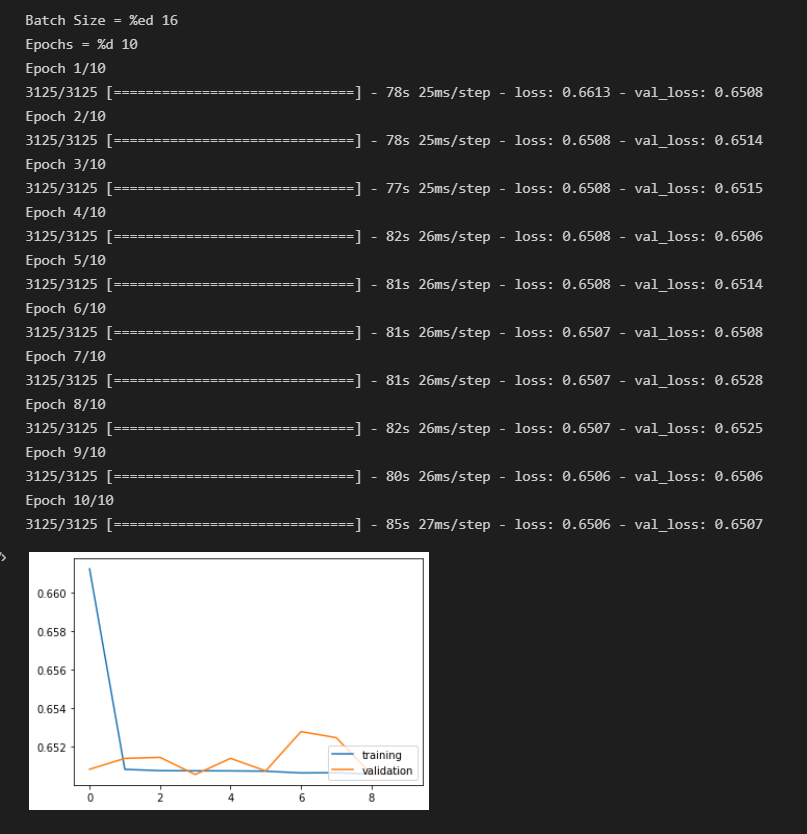
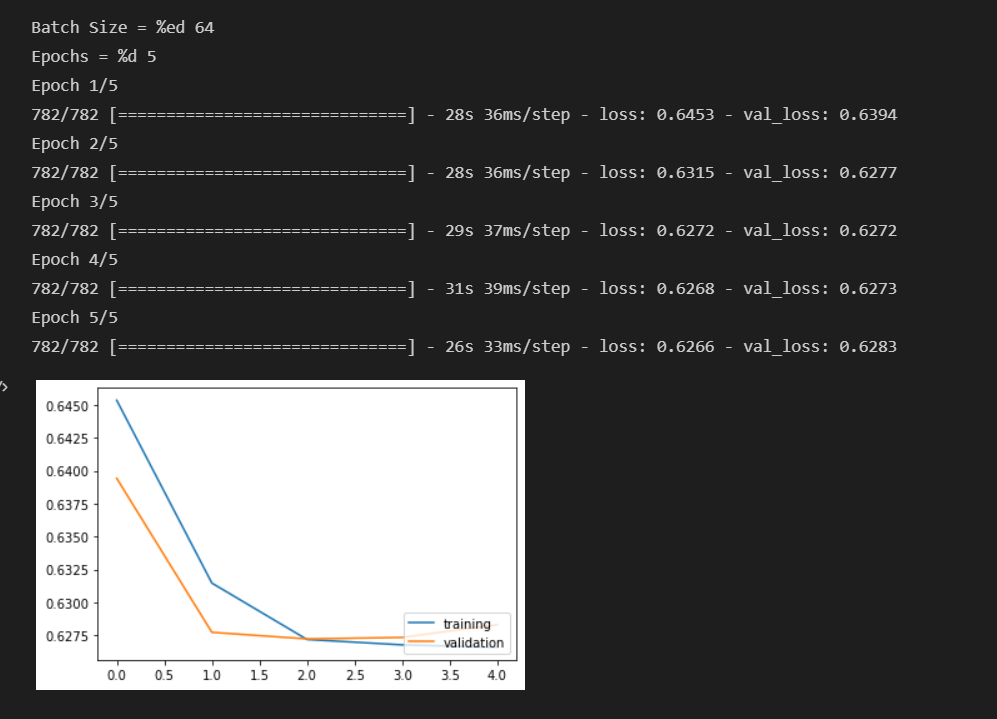
To find the optimal batch size and epochs we run a loop with different combinations of increasing values for both of them.

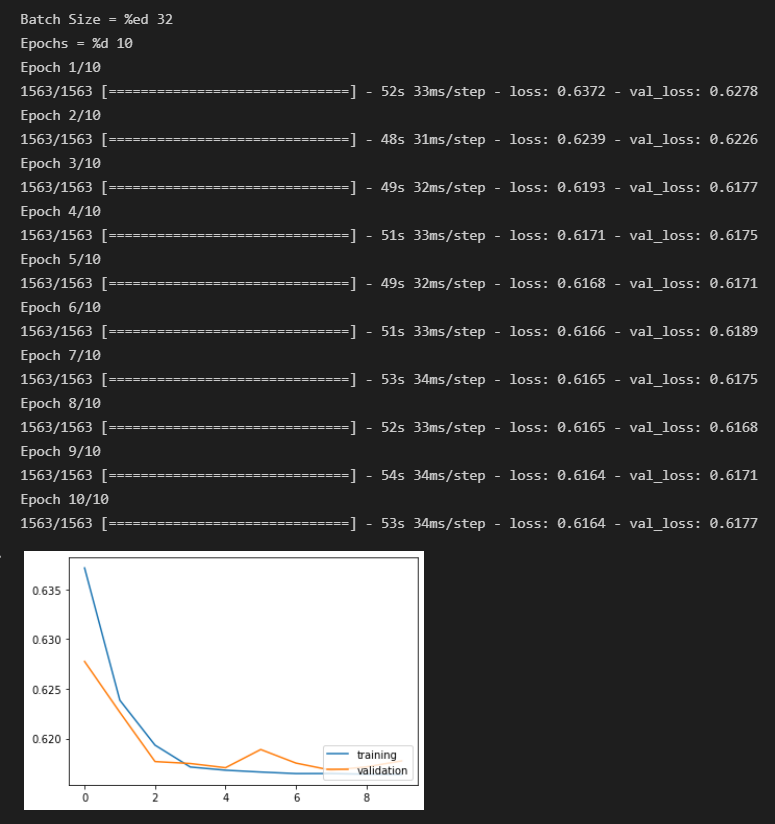


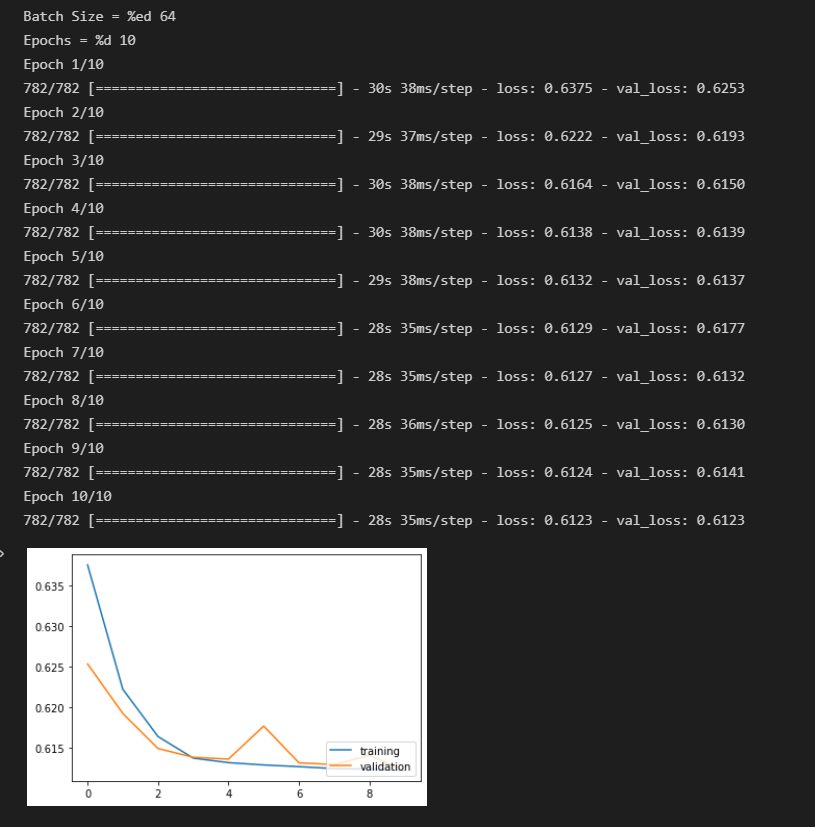
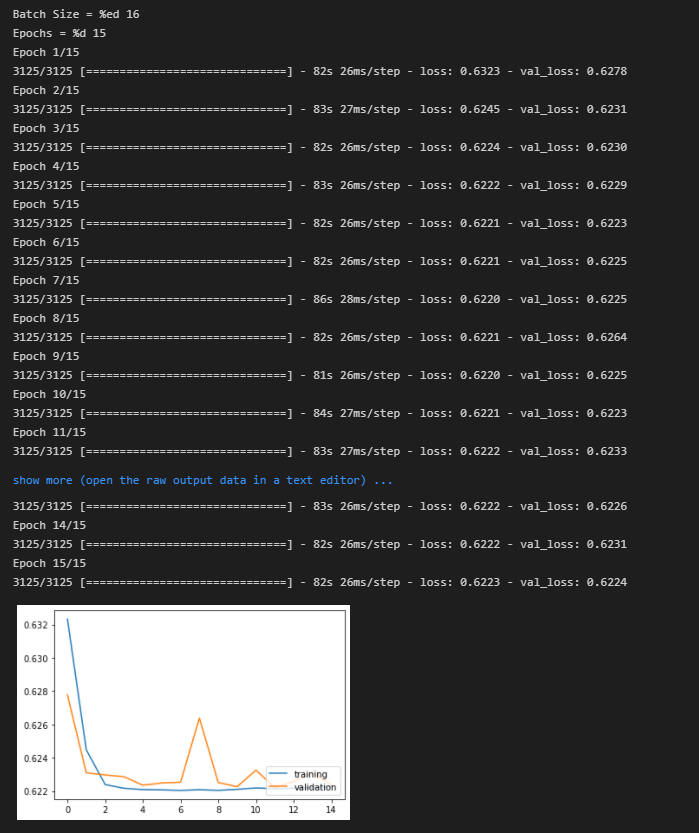
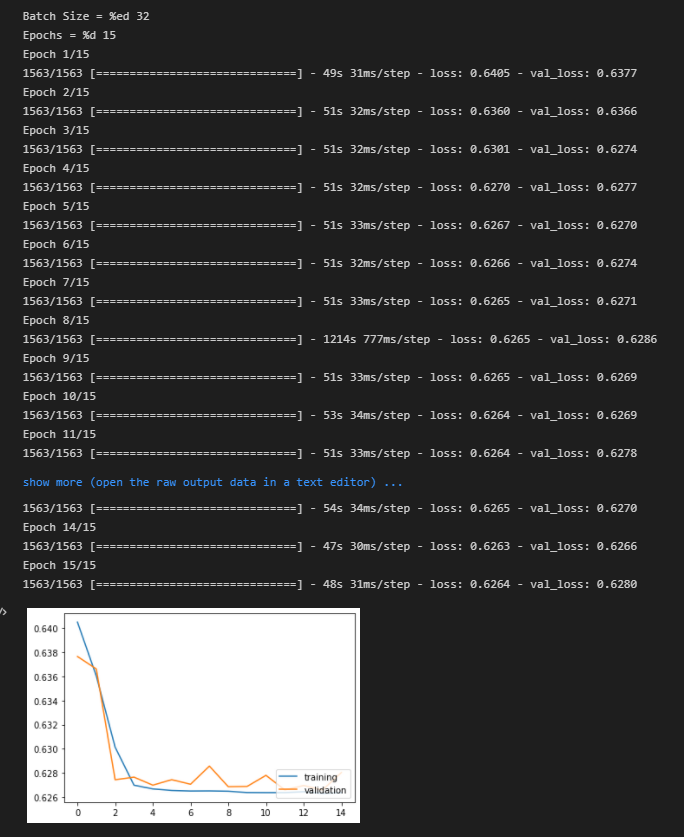
With these values, the following are results

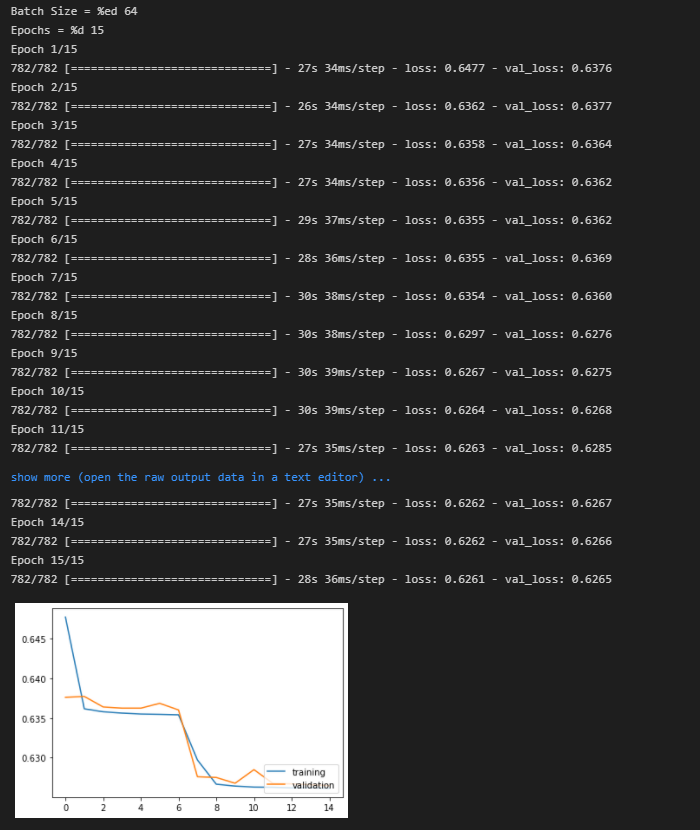








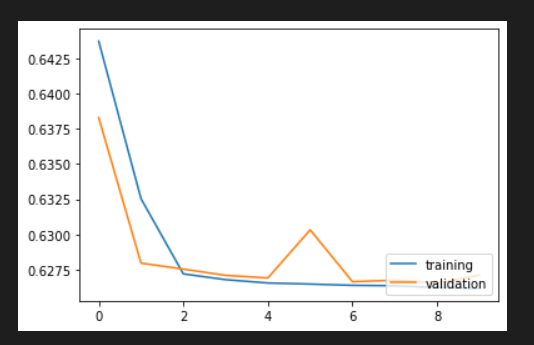
  

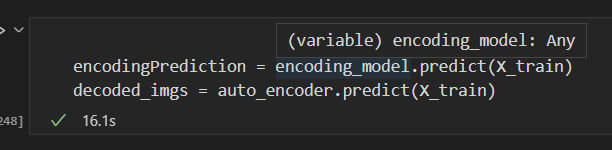


After running the loop, batch size = 64 with epochs = 10 we got the lowest loss value for the auto encoder.

Now we can run the auto encoder with the above values

Training loss vs Validation loss:

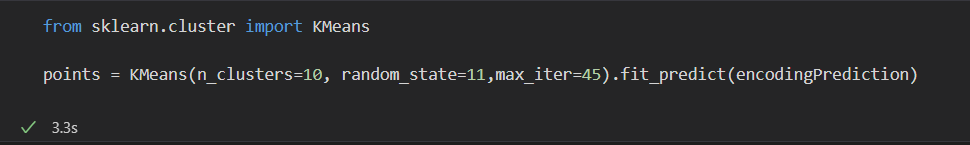




We are getting the sparse representation with running training data on encoding model and decoded images are always the output of auto encoder

# K means with auto encoder:

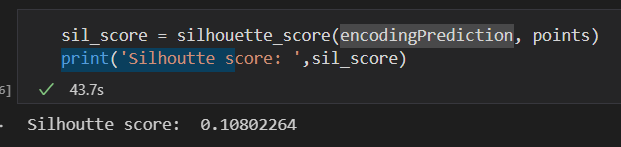
We are implementing the K means with data generated from encoder i.e, sparse representations of the input data is then used as input for K-means to generate clusters.



Using the sklearn Kmeans library to implement the model.

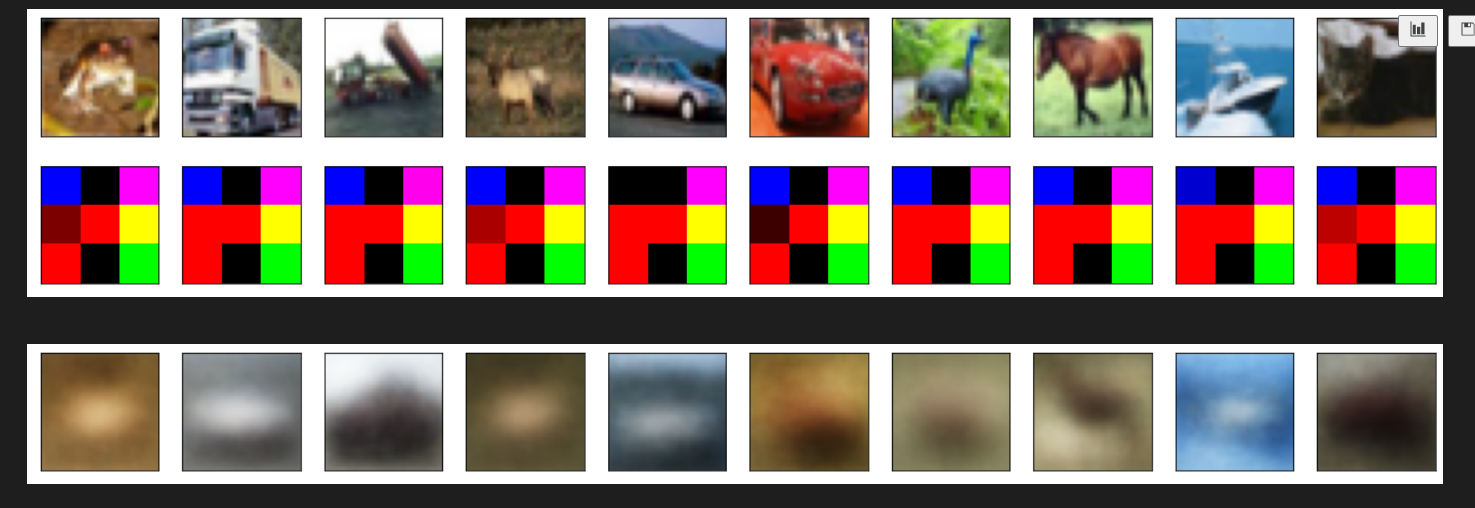
After we generate clusters and get the points or y or target data we then run the silhouette score on the sparse representations.

**Silhouette score is: 0.108**

****

# Image Representation:

After running the auto encoder, we try to plot the encoded images and then decoded images



First row are the original inputs

Second row is the sparse representation of inputs

Third row is the decoded images

References:

1. <https://numpy.org/doc/stable/reference/generated/numpy.argmin.html>
2. <https://www.google.com/search?q=euclidean+distance+formula&rlz=1C1CHZN_enUS968US968&oq=euclidean+di&aqs=chrome.1.69i57j69i59j0i433i512l2j0i512j0i20i263i512j69i60l2.4708j0j7&sourceid=chrome&ie=UTF-8>
3. <https://en.wikipedia.org/wiki/Dunn_index>
4. <https://matplotlib.org/stable/gallery/lines_bars_and_markers/scatter_with_legend.html>
5. https://blog.keras.io/building-autoencoders-in-keras.html