CSE 574 Introduction to Machine Learning

Project 2

Rahul Reddy Karumuri (50416790)

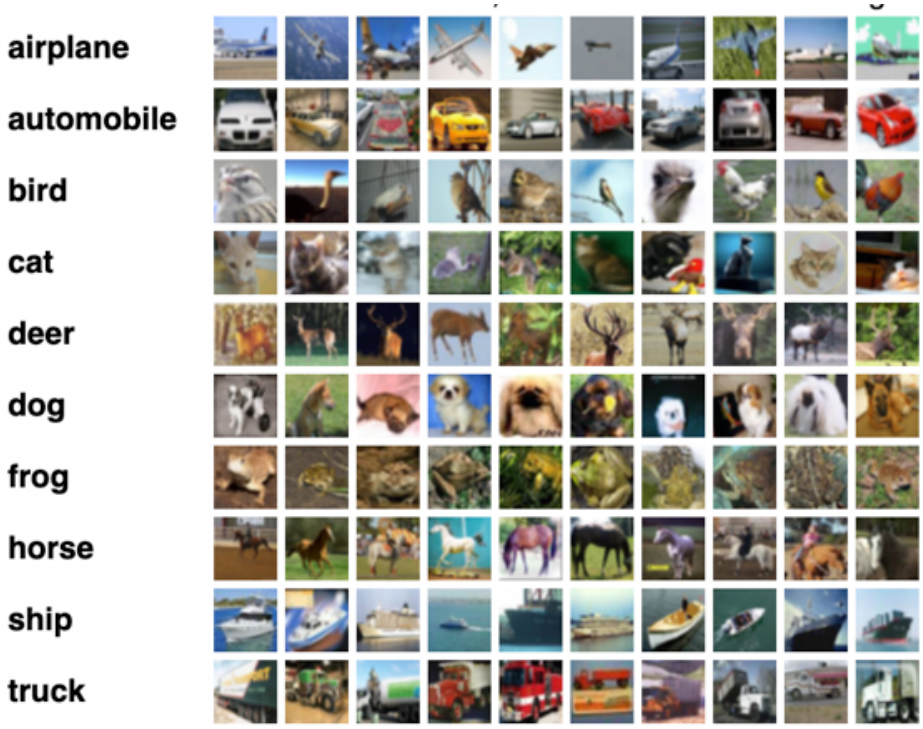
1. Overview:

The goal of this project is to implement k means clustering on a CIFAR-10 dataset and to perform optimal clustering. In the 1st part of this project, I have implemented K Means Clustering on image data from scratch, and calculated the quality of clusters using ASC (Average Silhouette Coefficient) and DI (Dunn's Index), while in the 2nd part, I have implemented Autoencoders using Convolutional Auto Encoders

2. Dataset:

Dataset that we need to consider is CIFAR-10 dataset. This dataset contains the record of 60,000 32x32 color images of 10 classes, with 6000 images per class. Out of 50000 are training images and 10000 are for testing. This dataset is divided into five training batches and one test batch, with 10000 images each.

Here is the sample dataset, with 10 random images from each cluster:



3. Environment:

I have used Jupiter Notebook on Google Collab for implementation of this project.

4. Implementing K-Means Clustering:

4.1 Processing:

1. Firstly, I converted the cifar10 dataset into grayscale using cv2

2. Secondly, I normalize the given data for the better results

3. Lastly, I reshaped the images from (10000,32,32) to (10000,1024)

4.2 Algorithm for K-means Clustering:

1. Randomly, choose k examples as initial clusters
2. while True:
2.1 create k clusters by assigning each example to the closest centroid
2.2 compute k new centroids by averaging examples in each cluster
2.3 if difference between previous centroid and current centroid is 0:
2.4 break

4.3 Common terms for this part:

K Means: K means Clustering is an unsupervised machine learning algorithm that usually divides the dataset into K clusters (each cluster has a centroid) such that each datapoint is assigning to the closet centroid. This algorithm works by first initializing the k centroids of clusters, and at each step, it classifies the points based on which cluster it is closest to, and then updates the centroids of the clusters by calculating their average.

Euclidean Distance: Euclidean Distance is the distance between two pints in an n dimensional space, most commonly the length of the line segment between two points.
Formula to calculate Euclidean Distance:

$$d(\mathbf{p}, \mathbf{q}) = \sqrt{\sum_{i=1}^{n} (q_i - p_i)^2}$$

p, q = two points in Euclidean n-space
q_i, p_i = Euclidean vectors, starting from the origin of the space (initial point)
n = n-space

Silhouette: Silhouette refers to a method of interpretation and validation of consistency within clusters of data. The technique provides a representation of how well each object has been classified. The silhouette value is a measure of how similar an object is to its own cluster compared to other clusters

Dunn's Index: The Dunn Index is the ratio of the smallest distance between observations not in the same cluster to the largest intra-cluster distance.

In this project we have calculated our evaluation metrics for our algorithm by using the Silhouette and Dunn's index score

4.4 Some sample code blocks, that I used in this project:

Here, is the function that I used to normalize our given data which will take train and test data as parameters

```
def normalize(XTrain, XTest):
    trainNorm = XTrain.astype('float32')
    testNorm = XTest.astype('float32')
    trainNorm /= 255.0
    testNorm /= 255.0
    return trainNorm, testNorm
```

Here is the function I used to calculate the initial random centroids which will take data and data size as inputs and return K centroids.

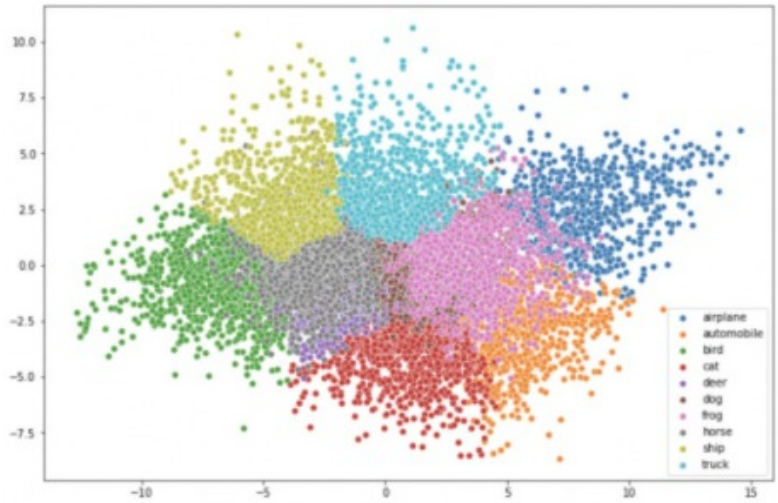
```
def initializeRandomCentroids(data, numFeatures):
    #Average of the given data
    mean = np.mean(data, axis = 0)
    #Standard Deviation of the given data
    standardDeviation = np.std(data, axis = 0)
    return np.random.randn(K,numFeatures)*standardDeviation + mean
```

4.5 Results:

For calculating the silhouette score I have used silhouette_score function from sklearn_library And the result I got is ***0.056311768***

For calculating the Dunn's Index, I have used the Dunn function from the validclust library. And the result I got is ***0.09360166***

Let's have a look at our clusters after plotting them



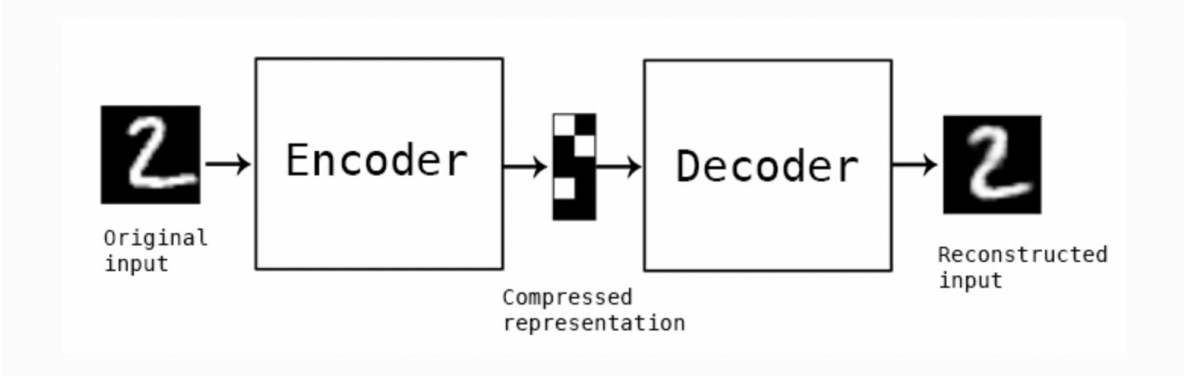
5. Implementing Auto Encoder:

Autoencoding is an unsupervised artificial neural network algorithm that learns how to efficiently compress and encode the data, then learns how to reconstruct the data back from the reduced encoded representation to a representation that is as close to the original input as possible. where the compression and decompression functions are

Data-specific: It means it can only be able to compress data like what they have been trained on, for example an autoencoder trained on pictures of faces would do a rather poor job of compressing pictures of trees, because the features it would learn would be face-specific.

Lossy: It means that the decompressed outputs will be degraded compared to the original inputs

learned automatically: Autoencoders are learned automatically from data examples, which is a useful property, it means that it is easy to train specialized instances of the algorithm that will perform well on a specific type of input. Below figure is an example of Auto Encoder



5.1 Autoencoder Components:

Encoder: In this part, model learns how to reduce the input dimensions and converts the input data to encoded representation

Bottleneck: This layer contains the compressed representation of input data

Decoder: In this part the model learns how to reconstruct the image from the encoded representation

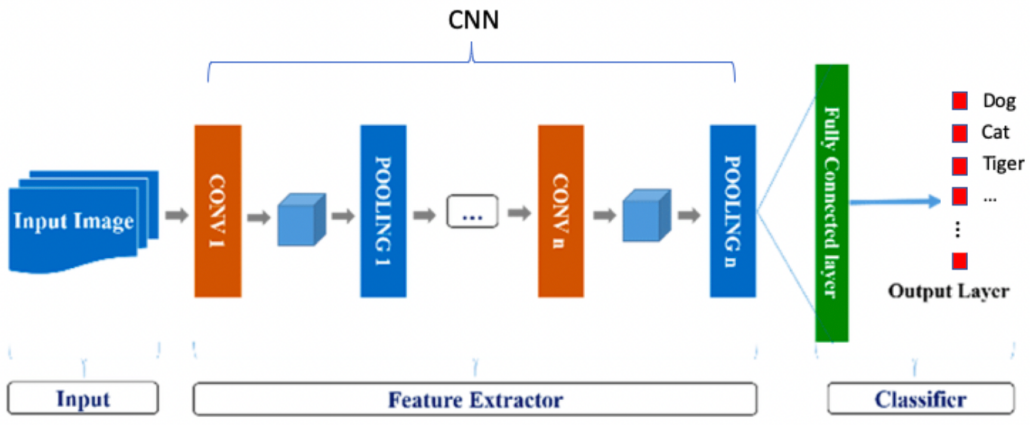
Reconstruction Loss: This is a method to check how well the decoder is performing and how close the input is to the original image

5.2 Types of Autoencoders:

There are many types of Autoencoders:

- Denoising Autoencoder
- Sparse Autoencoder
- Deep Autoencoder
- Contractive Autoencoder
- Undercomplete Autoencoder
- Convolutional Autoencoder
- Variational Autoencoder

In this project I have used convolutional Autoencoder,



5.3 Building the Autoencoder Model (Convolutional Autoencoder):

In this part I have used many modules from keras.layers and keras.models libraries and I applied kmeans on the encoded images and calculate the Evaluation metrics(Silhoutte, Dunn's Index) for the encoded model. For this part, I have used K means function from the sklearn library. And finally, I plotted the images after decoding the images and compared it with the actual images.

5.3.1. Functions I used for building the model:

Maxpooling: Max pooling selects the brighter pixels from the image. It is useful when the background of the image is dark and we are interested in only the lighter pixels of the image.

Batch Normalization: Batch normalization is a technique for training deep neural networks that standardizes the inputs to a layer for each minibatch.

Upscaling: This is used to convert the low-resolution image to a higher dimension image

For compilation of the model: I have used adam optimizer and for loss function I have used mean squared error since these are giving better results than the others.

5.4 Results:

In this part, after building the autoencoder model, I had applied k-means(sklearn library) to the encoded images

And I calculate the Silhoutte score and Dunn's index same as the above function and obtained the results as

```
Average Silhouette Score: 0.019351285
```

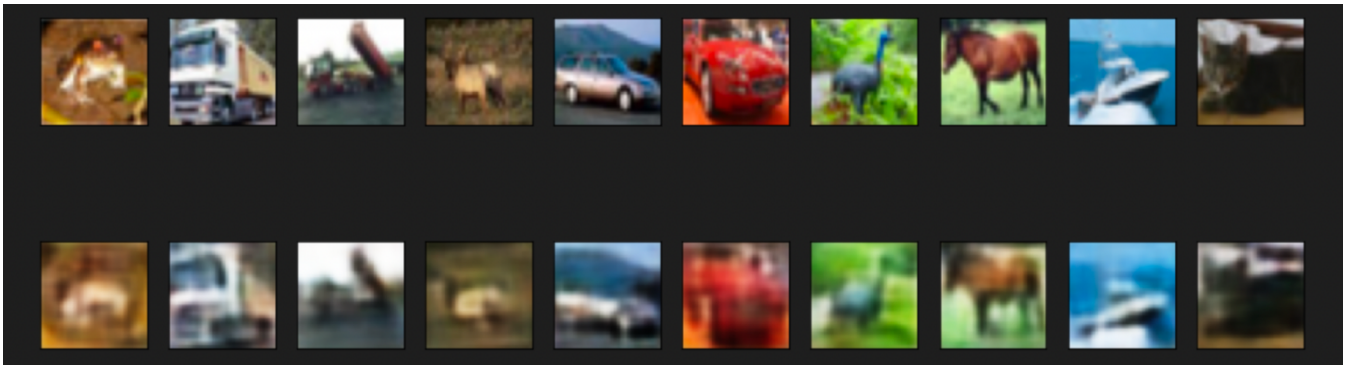
```
Dunn Index: 0.3701047
```

And, while building this model, I got the Accuracy and validation loss as

```
accuracy: 0.7636
```

```
loss: 0.0061
```

Finally, I had plotted the images between the decoded images and actual images, and this looks like below:



7.References

1. K Means Clustering Algorithm | K Means Example in Python | Machine Learning Algorithms | Edureka (https://www.youtube.com/watch?v=1XqG0kaJVHY&t=1270s)

2. https://blog.keras.io/building-autoencoders-in-keras.html

3. https://towardsdatascience.com/auto-encoder-what-is-it-and-what-is-it-used-for-part-1-3e5c6f017726

4. https://iq.opengenus.org/types-of-autoencoder/