
PROJECT - 2

Sai Teja Cherukuri (50418484)

cheruku3@buffalo.edu

14th November 2021

---

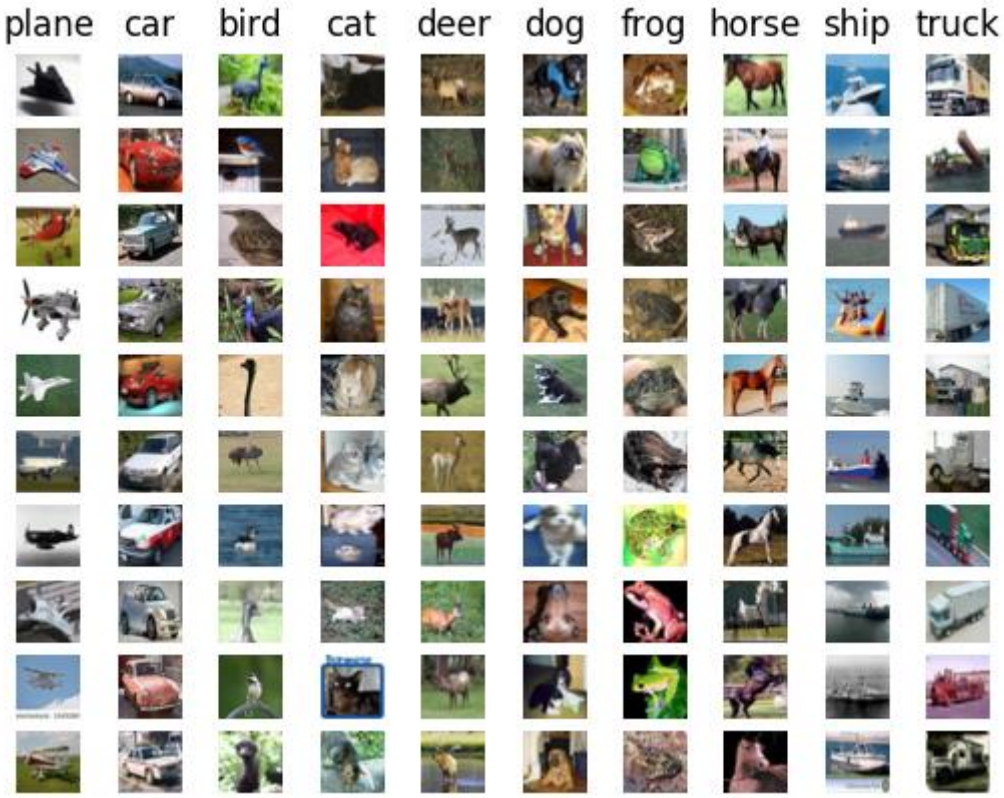
1. Project Overview

The goal of the project is to perform unsupervised learning on Cifar-10 dataset. In the first part of the project, I have performed K-means clustering on the raw data from scratch. Then, I have used Silhouette analysis and Dunn index are used to evaluate the model. In the second part, I have performed k-means clustering on the representations generated by Auto-Encoder.

2. Dataset

Cifar-10 dataset has a training set of 50,000 examples, and a test set of 10,000 examples. Each example is a 32x32 image, associated with a label from 10 classes. Each image is 32 pixels in height and 32 pixels in width, for a total of 1024 pixels in total.
The dataset is divided into 5 training batches and 1 testing batch, each with 10000 images. The test batch contains exactly 1000 random selected images from each class and training batches contain exactly 5000 images from each class. There is no overlap in the classes.

Below is the representation of some images from Cifar-10 dataset:



3. Python Editor

I have used Jupiter Notebook on Google Collab for implementation and shared.

3.1 Data Pre-Processing

We gray scale the images, hence making the size of the initial array (10,000 * (32,32,3)) to 10,000*(32,32)

4. Model Building

Part -1 : Implementing K-Means Clustering

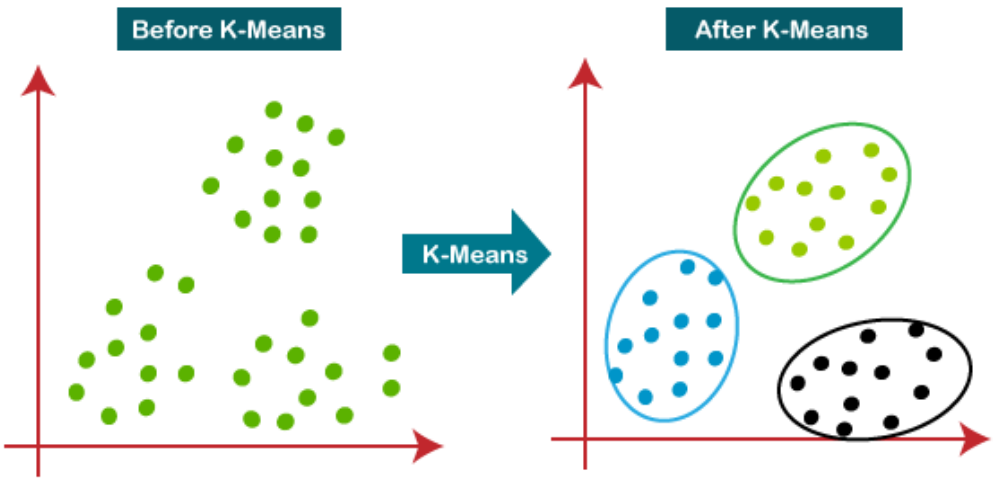
- Load train and test data from Cifar-10 dataset.

```
(x_train, y_train), (x_test, y_test) = cifar10.load_data()
```

- Convert the images to a grey scale, hence converting the size of array from (10,000 * (32,32,3)) to 10000*(32,32).
- Now, import the pairwise_distances from sklearn and calculate the distance between every pair of samples.

K-Means function:

1. Select K samples at random from the dataset to be the initial clusters. (Let K be 10).
2. Calculate the Euclidean distance between every object and the centroid.
3. Now assign the objects to the nearest cluster.
4. For each of the k clusters, update the cluster centroids by calculating the mean of all the data points in that cluster.
5. Iteratively repeat step 3 & 4 until we reach terminate condition i.e., the error is less than tolerance (0.0001) or the iterations reach the assigned max_iterations.



- Next, we define the number of clusters as 10 and the above K-Means function is iteratively called for a given number of iterations.

- We retrieve the centroids, distortion measure and the number of iterations the model took to converge.

```
print("Model took ",iteration_number," iterations to converge.")
print("Model's distortion measure value: ",distoration_measure)
```

```
Model took  29  iterations to converge.
Model's distortion measure value:  224173816.7808051
```

Evaluation Metrics (Silhouette index and Dunn index):

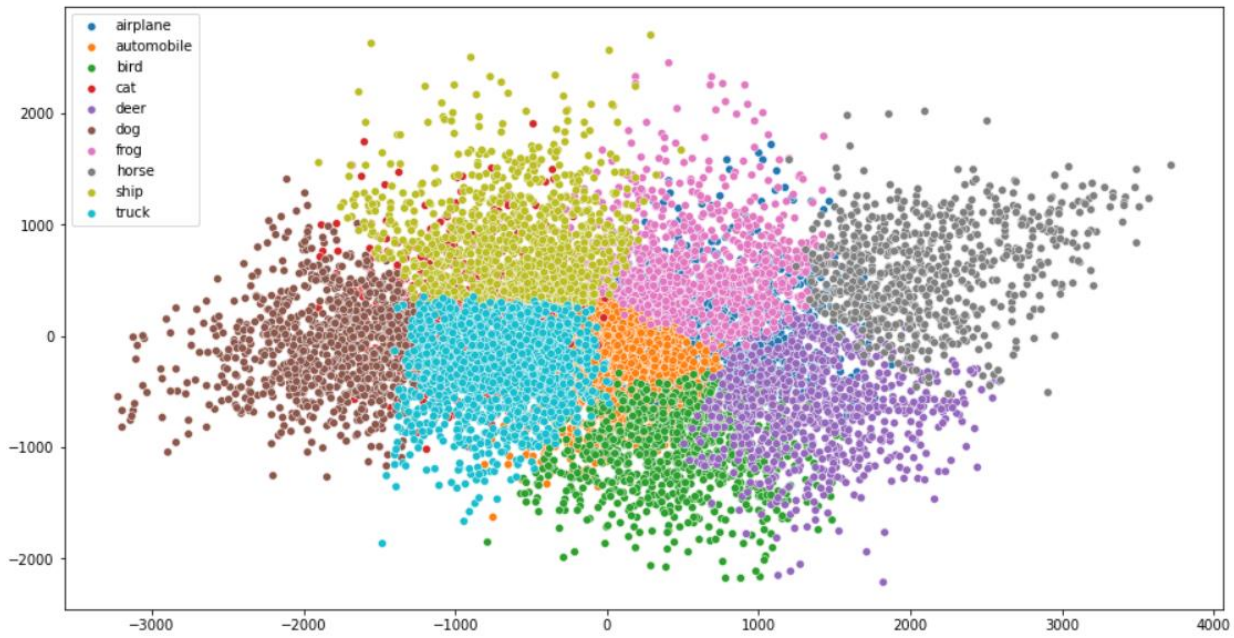
- We import the Silhouette_score function and Dunn Index function from sklearn and evaluate these metrics with the train data, distance and the sample points.

```
print('silhouette_score: ',silhouette_score(x_train,points))
print('Dunn index: ',dunn(distance1,points))
```

```
silhouette_score:  0.05978191335608655
Dunn index:  0.09121709729952622
```

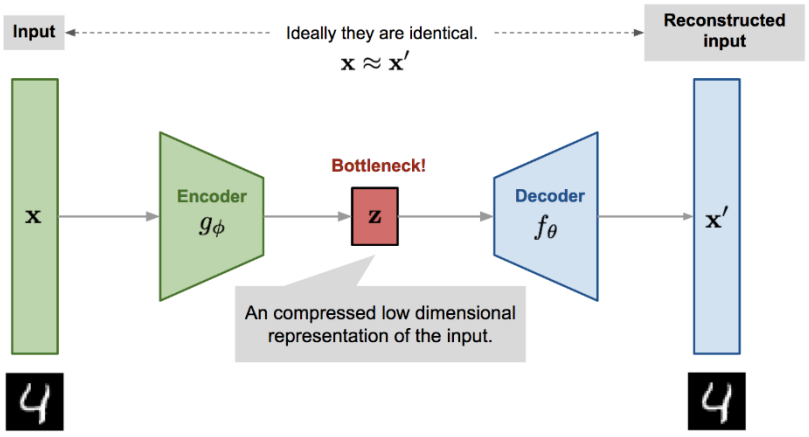
Visualizing the final clusters:

- We assign different colours to 10 clusters and run Principal Component Analysis (PCA) which reduces the dimensions of the dataset to 1024.
- Now, we use a scatter plot to visualize the final clusters as below.



Part – 2: Implementing Convolutional Auto-Encoder

- As a first step, we import tensorflow and necessary libraries from Keras.
- "Autoencoding" is a data compression algorithm where the compression and decompression functions are 1) data-specific, 2) lossy, and 3) *learned automatically from examples* rather than engineered by a human. Additionally, in almost all contexts where the term "autoencoder" is used, the compression and decompression functions are implemented with neural networks.



Data Pre-processing:

- Divide the pixel values of train data with 255 (maximum value), inorder to rescale them into range [0,1]

```
# normalize data
X_train = X_train.astype('float32')
X_test = X_test.astype('float32')
X_train /= 255
X_test /= 255
```

Model building:

- Now we define our Convolutional Auto-Encoder model,
  - The convolutional Auto-Encoder is a set of Encoder (consists of convolutional, maxpooling and batch normalization layers) and Decoder (consists of convolutional, up sampling and batch normalization layers)
  - The goal of our convolutional autoencoder is to extract features from image, with the measurement of mean squared error between input and output image.
  - For the Encoder, I have used 3 hidden layers:
    - First, we have a convolutional layer with 32 neurons and the activation function ReLu (Rectified Linear Unit). This is a typical number of neurons, just as the activation function is also widely used in neural networks. The size of the kernel filter is 3x3 and was chosen according to the size of the input image. Then, the output of this layer is normalized by a Batch Normalization layer. Further then, a Max Pooling operation is applied, which reduces the data dimensions to 16x16.
    - The next layer repeats the pattern of the first, with the same number of neurons, the same size of the kernel filter and a layer of batch normalization in the output along with the Max Pooling operation, making the data dimensions reduce to 16 x 16.

- Third layer also follows the same pattern as above two, and hence has the data dimensions as 16 x 16.
- The encoded layers is combined as a Encoder Model.

```
# Encoder
x = Conv2D(32, (3, 3), padding='same', activation='relu')(input_image)
x = BatchNormalization()(x)
x = MaxPooling2D((2, 2), padding='same')(x)
x = Conv2D(32, (3, 3), padding='same', activation='relu')(x)
x = BatchNormalization()(x)
x = MaxPooling2D((2, 2), padding='same')(x)
x = Conv2D(32, (3, 3), padding='same', activation='relu')(x)
x = BatchNormalization()(x)
encoded = MaxPooling2D((2, 2), padding='same')(x)
encodeModel = keras.Model(input_image, encoded, name="Encoder")
```

- For the Decoder, I have used 3 hidden layers:
  - First, we have a convolutional layer with 32 neurons and the activation function ReLu (Rectified Linear Unit). The size of the kernel filter is 3x3 and was chosen according to the size of the input image. Then, the output of this layer is normalized by a Batch Normalization layer. Further then, a Up Sampling operation is applied, which increases the data dimensions to 64x64.
  - Second and Third layers are similar to the first layer, which have a Batch Normalization layer as well as Up sampling operation.

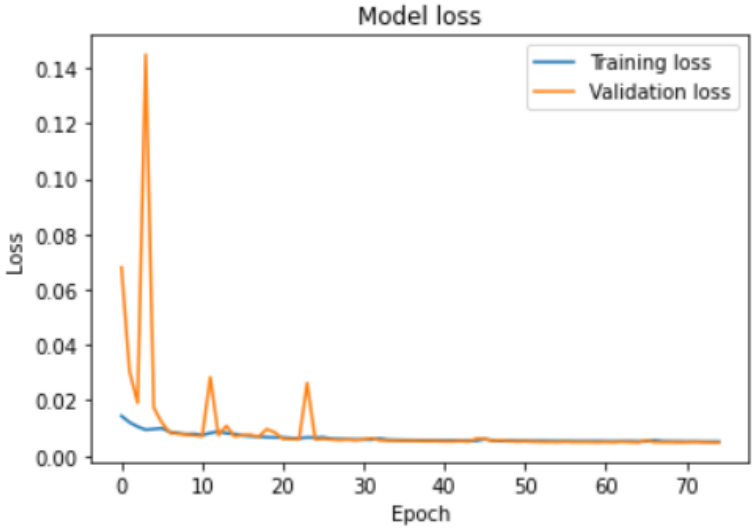
```
# Decoder
x = Conv2D(32, (3, 3), padding='same', activation='relu')(encoded)
x = BatchNormalization()(x)
x = UpSampling2D((2, 2))(x)
x = Conv2D(32, (3, 3), padding='same', activation='relu')(x)
x = BatchNormalization()(x)
x = UpSampling2D((2, 2))(x)
x = Conv2D(32, (3, 3), padding='same', activation='relu')(x)
x = BatchNormalization()(x)
x = UpSampling2D((2, 2))(x)
x = Conv2D(3, (3, 3), padding='same', activation='sigmoid')(x)
decoded = BatchNormalization()(x)
```

- Now we compile our Auto-Encoder model with the above layers and by using optimizer as ‘adam’. We have ‘mean squared error’ loss function as it works better than categorical or binary crossentropy.

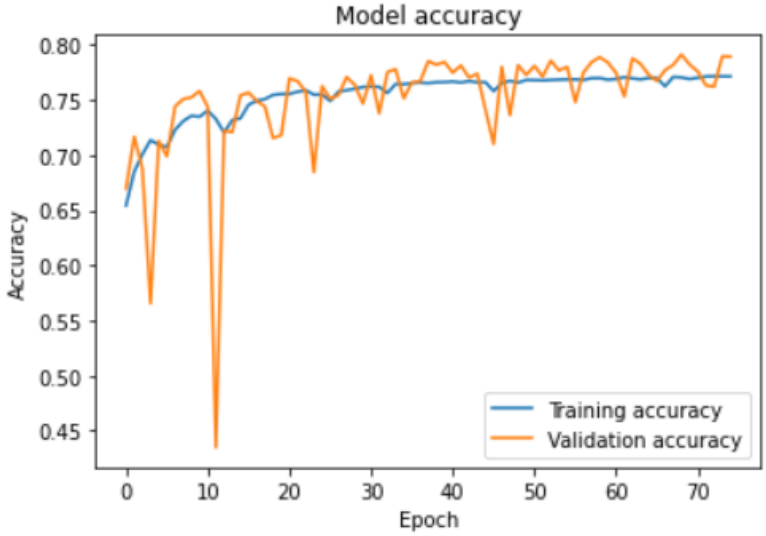
Training:

- We train the model for 75 epochs with batch size of 64.
- Once the model is trained, we evaluate the model on test data,
the accuracy achieved is **79.09%** and loss is reduced to **0.0047**
- We can now visualize the Model loss vs Epoch and Model accuracy vs Epoch

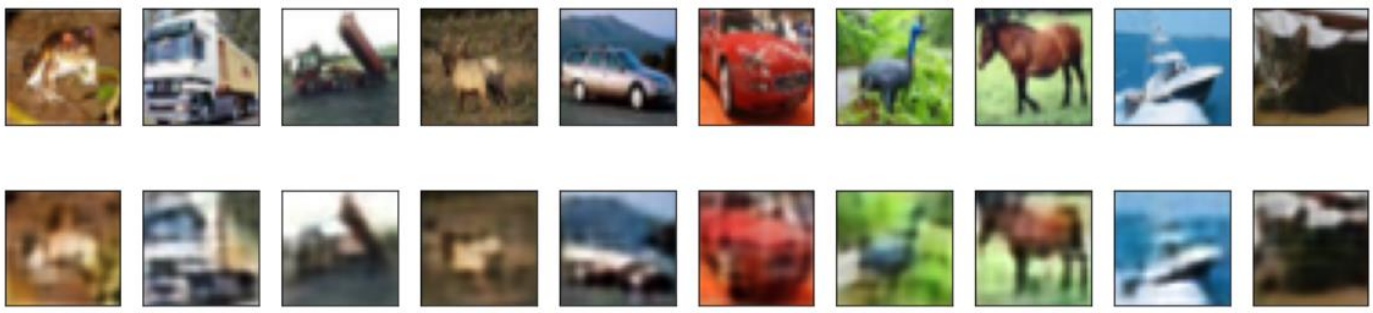
Below is the graph for Model loss vs Epoch.



Below is the graph for Model accuracy vs Epoch



➔ Now we visualize the Original image vs Re-constructed image from the Auto-Encoder model.



➔ **Generating clusters using K-Means from the sparse representations generated by the Auto-Encoder.**

- We pass the encoded images generated from Convolutional AutoEncoder to KMeans function to generate clusters.
- Using Pairwise_distance function from sklearn, we calculate the distance between every pair of samples
- Finally, we print the Silhouette score and Dunn's index.

```
print('silhouette_score: ',silhouette_score(encodedImages,points_))
print('Dunn index: ',dunn(distance2,points_))
```

```
silhouette_score:  0.03913329
Dunn index:  0.184344
```

References:

1. Demo of KMeans Clustering on Hand Written digits data (https://scikit-learn.org/stable/auto_examples/cluster/plot_kmeans_digits.html)
2. Convolutional Auto Encoders (https://keras.io/examples/vision/autoencoder/)
3. Implementing Convolutional Auto Encoders (https://www.youtube.com/watch?v=PIkhRbUWanw)