Part 1: Implement K-Means Clustering

Index

List of Figures

1] Overview:

The goal of the project is to use K-means clustering on supervised learning using the Cifar-10 data set and to use various evaluation metrics to identify the validation of the predicted result.

2] Dataset Description & Analysis:

CIFAR-10 is an established computer-vision dataset used for object recognition. It is a subset of the 80 million tiny images dataset and consists of 60,000 32x32 color images containing one of 10 object classes, with 6000 images per class.

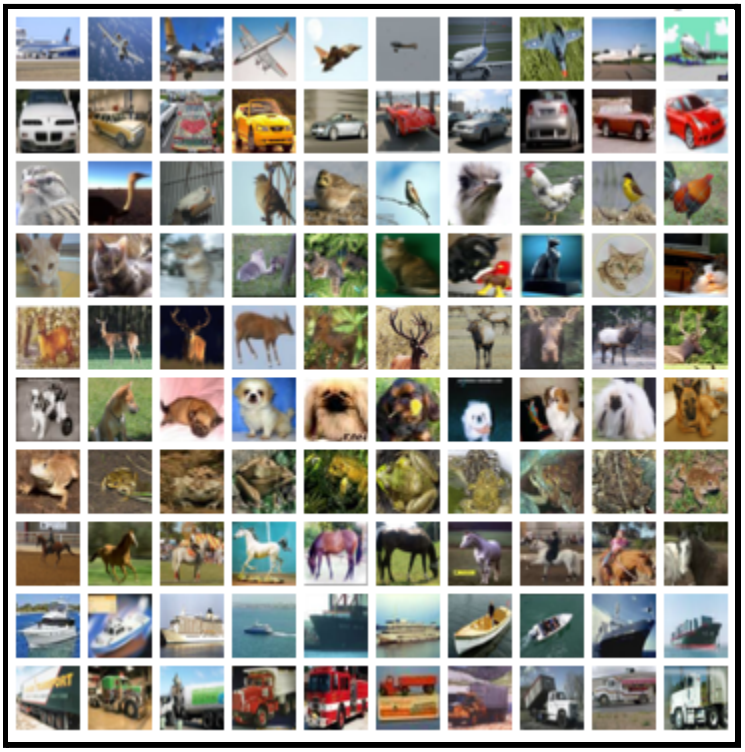

Fig 1: Dataset

Classes in Cifar-10 include: airplanes, cars, birds, cats, deer, dogs, frogs, horses, ships, and trucks

3] Treating Outliers and Non-Normality:

- I imported the dataset from Keras datasets.
- I then converted the test set into black & white images from RGB, this was done to reduce the further computational part of the dataset
- I tried to normalize the dataset, that is I tried to normalize the pixel values of the images. But after training the K-means model found out that the model gave better results without normalizing.

- Hence I went ahead with the normal dataset.
- Since the shape of the testing dataset was [10000,32,32,3] it would be tedious to work with it so I flattened it to [10000,1024] where 1024 is 32*32*3.

```
# Import Dataset
from keras.datasets import cifar10
(x_train, y_train), (x_test, y_test) = cifar10.load_data()

#converting to Grayscale
x_test = np.array([cv2.cvtColor(image, cv2.COLOR_BGR2GRAY) for
image in x_test])

# Flattening Dataset
x_test = x_test.reshape((-1, 1024))
```

Fig 2: Data importing & preprocessing

4] Designing K-Means Clustering:

- As per my research, K-Means Clustering is an unsupervised learning algorithm that aims to group the observations in a given dataset into clusters. The number of clusters is provided as an input. It forms the clusters by minimizing the sum of the distance of points from their respective cluster centroids.

- Clustering is a type of unsupervised learning which is used to split unlabeled data into different groups

- **Steps followed for K-Means Clustering:**

    1. **Initialize centroids:** I started by randomly choosing K no of points, these were points chosen from the dataset.

```
for i in range(k):
    random_img_centroid = random.randint(0, 9)
    centroids[i] =  x_test[random_img_centroid]
```

Fig 3: Initializing centroids

2. **Assign Clusters:** Then the clusters are assigned to each point in the dataset by calculating their distance from the centroid and assigning it to the centroid with minimum distance.
    i. To compute the distance between a point and every other point Euclidean distance was used.

```
def recalculate_clusters(X, centroids, k):
  """ Recalculates the clusters """
  clusters = {}
  for i in range(k):
    clusters[i] = []
  for data in X:
    euc_dist = []
    for j in range(k):
      euc_dist.append(np.linalg.norm(data - centroids[j]))
    clusters[euc_dist.index(min(euc_dist))].append(data)
  return clusters
```

Fig 4: Creating Clusters

3. **Re-calculate the centroids:** Updating the centroid by calculating the centroid of each cluster created.

```
def recalculate_centroids(centroids, clusters, k):
  for i in range(k):
    centroids[i] = np.mean(clusters[i], axis=0)
  return centroids
```

Fig 5: Re-calculating Centroids

4. **Repeating steps 2 & 3:** for 500 iterations to obtain final clusters and their centroids

5] Evaluation Metrics & Results:

- **Silhouette Score:** Silhouette refers to a method of interpretation and validation of consistency within clusters of data. The silhouette value is a measure of how similar an object is to its own cluster compared to other clusters. The best value is 1 and the worst value is -1.

```
Silhouette_score = 0.07897219597931708
```

Fig 6: Silhouette score

- **Dunn's Index:** The Dunn index is a metric for evaluating clustering algorithms. This is part of a group of validity indices including the Davies–Bouldin index or Silhouette index, in that it is an internal evaluation scheme, where the result is based on the clustered data itself.

```
Dunns_Index = 0.09491897994556555
```

Fig 7: Dunns Index

6] Additional Trial & Error:

- I tried to decompose the data and then rain the model
- I used Principal component analysis (PCA) with 2 components, & and whiten as True to the data.
- I got the Silhouette Score pretty high

```
Silhouette_score = 0.32784730678845336
```

- But the Dunn's Index did not give good results

```
Dunns_Index = 0.0009760651010421804
```

Part 2: Implement Auto-Encoder

Index

List of Figures

1] Data Normalizing:

- I normalized the dataset, that is I tried to normalize the pixel values of the images. By dividing it by 255, hence getting the pixels values in a normalized range.

2] Auto Encoders Architecture:

- Autoencoder is used as a data compression algorithm before the Cigar-10 images are fed to the means model

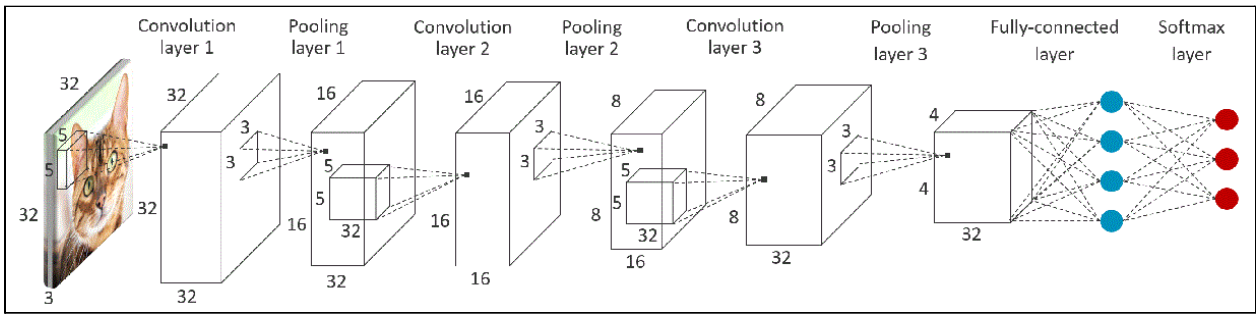

Fig 1: Convolutional Neural Network [ref]

- I have used *Convolutional Neural Network* to train the autoencoder.
- A Convolutional Neural Network takes in an input image, assigns importance (learnable weights and biases) to various aspects/objects in the image, and is able to differentiate one from the other.
- In the used CNN the size of the input image is 32*32*3.

Encoder Model:

```
Convolution Layer: 64 nodes - Relu Activation, padding=same
Batch Normalization
MaxPooling2D - 2*2 reduction, padding=same
Convolution Layer: 32 nodes - Relu Activation, padding=same
Batch Normalization
MaxPooling2D - 2*2 reduction, padding=same
Convolution Layer: 16 nodes - Relu Activation, padding=same
Batch Normalization
MaxPooling2D - 2*2 reduction, padding=same
```

Fig 2: Encoder Model Architecture

Decoder Model:

```
Convolution Layer: 16 nodes - Relu Activation, padding=same
Batch Normalization
UpSampling2D - 2*2 reduction, padding=same
Convolution Layer: 32 nodes - Relu Activation, padding=same
Batch Normalization
UpSampling2D - 2*2 reduction, padding=same
Convolution Layer: 64 nodes - Relu Activation, padding=same
Batch Normalization
UpSampling2D - 2*2 reduction, padding=same
```

Fig 3: Decoder Model Architecture

- **Batch Normalization:**
  - It is a method used to make artificial neural networks faster and more stable through normalization of the layers' inputs by re-centering and re-scaling.

- **Max Pooling:**
  - Operations downsamples the input along its spatial dimensions (height and width) by taking the maximum value over an input window for each channel of the input.

- **Activation Function:**
  - The Relu [Rectified Linear Unit] activation function has been used for all nodes except those in the output layer.
  - The output layer uses a linear activation function to maintain the dimensionality of the output image.

- **Loss or Cost Function:**
  - Mean Squared Error loss function has been used because the output contains either 1 for positive diagnosis or 0 for negative diagnosis. MSE measures the average of the squares of the errors, that is the average squared difference between the estimated values and the actual value.

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(Y_i - \hat{Y}_i)^2$$

MSE	=	mean squared error
n	=	number of data points
Yi	=	observed values
Y^i	=	predicted values

- **Optimizer:**
    - Adam has been used as the optimizer because it is an efficient way to apply gradient descent.

- **Metrics:**
    - Tensorflow accuracy metrics are used to evaluate the model

- **Early Stoping:**
    - Early stooping is applied during the training of the model, that is it keeps an eye on the loss of the model. If the model stops learning properly that is if the loss of the model does not reduce after 2 epochs [patience=2], then the model stops training. The best model is stored after each epoch that saves the weights of the model in order to be used in the next epoch.

- **Model:**
    - The model is trained for 20 epochs
    - With a batch size of 256 images
    - Test data is used as validation data for the model
    - Accuracy of the autoencoder model is 75.64%

2] Kmeans Architecture:

- Encoded images from the encoder of the autoencoder model are predicted.
- These encoded images are then reshaped to be fed to the Kmeans model for clustering.
- Sklearns Kmeans model is trained with the input of the encoded images.
- **Hyperparameters**
    - No. of Clusters = 10

- n_initint = 30, Number of times the k-means algorithm will be run with different centroid seeds. The final results will be the best output of n_init consecutive runs in terms of inertia.
- Max iteration = 1000

- **Evaluation Metrics & Results:**
  - **Silhouette Score:** Silhouette refers to a method of interpretation and validation of consistency within clusters of data. The silhouette value is a measure of how similar an object is to its own cluster compared to other clusters. The best value is 1 and the worst value is -1.

```
Silhouette_score = 0.02020698
```

Fig 4: Silhouette score

  - Below is the comparison results of the original training data with the decoded images from the autoencoder.

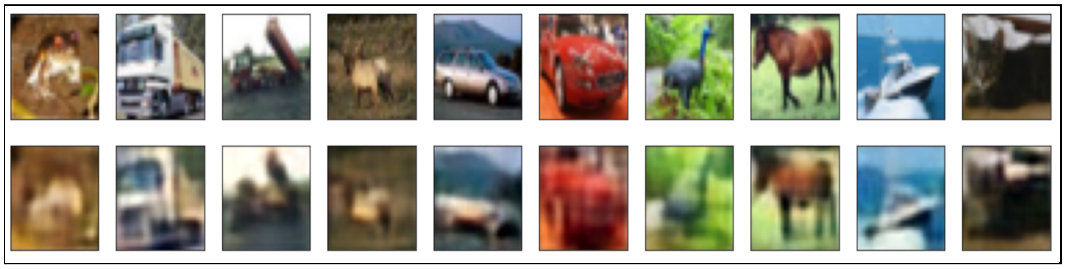

Fig 5: Original vs Decoded

6] Additional Trial & Error:

- I tried adding noise to the data input to autoencoder, but the kmeans did not act well noise data.

```
noise_factor = 0.5
x_train_noisy = x_train + noise_factor * np.random.normal(loc=0.0,
scale=1.0, size=x_train.shape)
x_test_noisy = x_test + noise_factor * np.random.normal(loc=0.0,
scale=1.0, size=x_test.shape)

x_train_noisy = np.clip(x_train_noisy, 0., 1.)
x_test_noisy = np.clip(x_test_noisy, 0., 1.)
```

- I also tried changing the number of nodes per layer for the autoencoder. But got the best results with the current architecture
- I also tried using the sigmoid activation function in the output layer but had to go with linear to maintain the dimensionality of the images