Assignment 2

Sanyam Gupta (50359154)

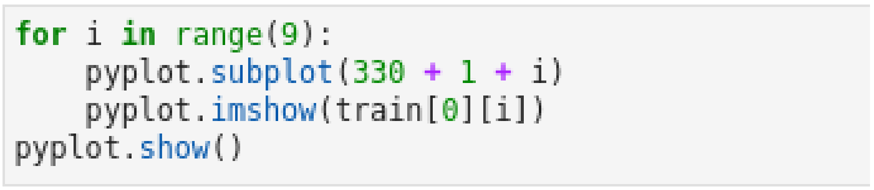
Nov 14 2021

1. Overview

The goal here is to train an unsupervised machine learning model for 10 class classification problem. The efficiency of the K-means clustering algorithm has been demonstrated on the Cifar 10 dataset and on data generated by autoencoder.

2. Dataset

The cIfar 10 Database has 60,000 samples. 50,000 of these are used for training the model and rest are used for testing the model efficacy.



```
for i in range(9):
    pyplot.subplot(330 + 1 + i)
    pyplot.imshow(train[0][i])
pyplot.show()
```

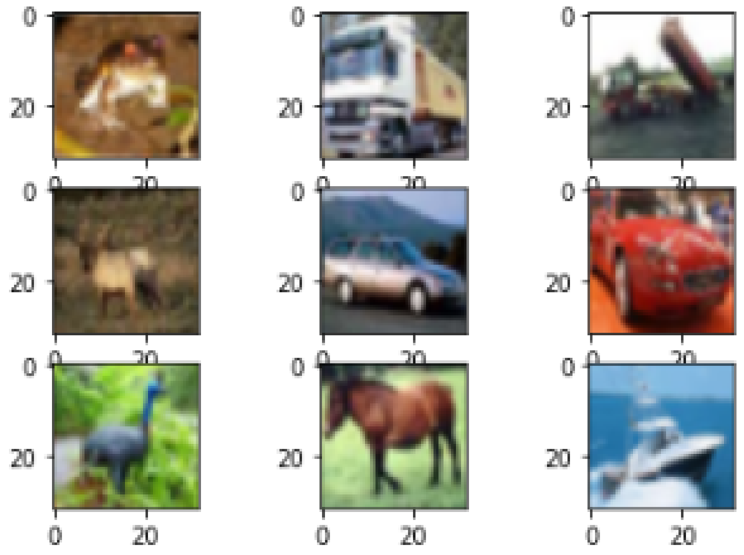
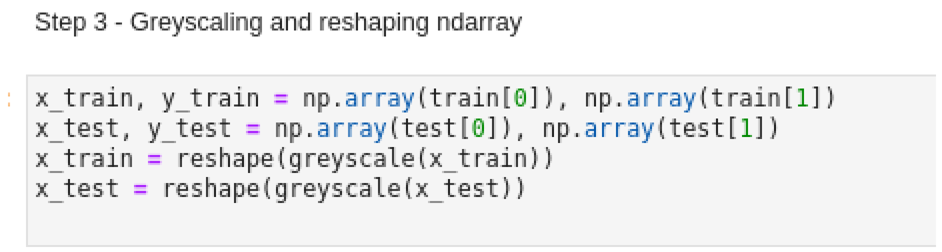


Fig 1 - Sample images in dataset

3. Data Preprocessing

An image data is made up of 3 channels, in order to better train the model all the images have been transformed to grayscale. Necessary reshaping has also been done. This resulting data is used for training and testing.



Step 3 - Greyscaling and reshaping ndarray

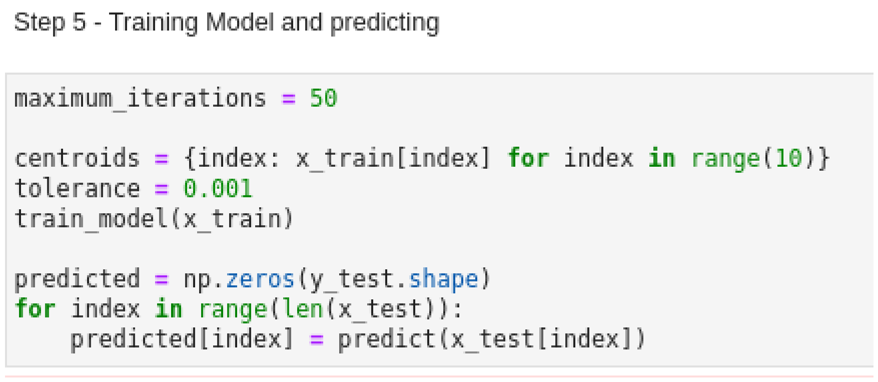
```
x_train, y_train = np.array(train[0]), np.array(train[1])
x_test, y_test = np.array(test[0]), np.array(test[1])
x_train = reshape(greyscale(x_train))
x_test = reshape(greyscale(x_test))
```

Fig 2 - Data preprocessing

4. K Means

4.1 Training the model

Algorithm is trained for either defined maximum iterations or until a change in cluster centroids falls below a tolerance value, whichever happens first. The obtained cluster centroids are used to make predictions on the testing dataset.



Step 5 - Training Model and predicting

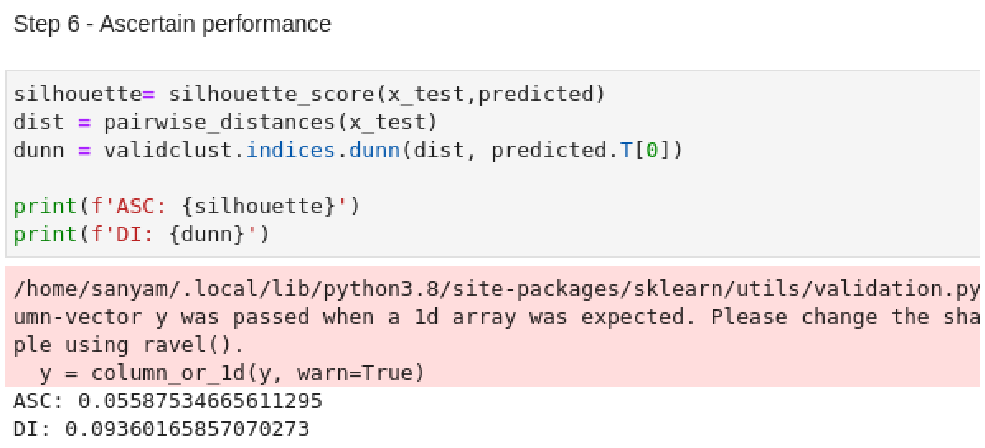
```
maximum_iterations = 50

centroids = {index: x_train[index] for index in range(10)}
tolerance = 0.001
train_model(x_train)

predicted = np.zeros(y_test.shape)
for index in range(len(x_test)):
    predicted[index] = predict(x_test[index])
```

Fig 3 - Training and predicting

4.2. Outcome

The following are evaluation metrics ASC (Average Silhouette Coefficient) and DI (Dunn's Index), and the resulting values:-

- ASC: 0.056
- DI: 0.093



Step 6 - Ascertain performance

```
silhouette= silhouette_score(x_test,predicted)
dist = pairwise_distances(x_test)
dunn = validclust.indices.dunn(dist, predicted.T[0])

print(f'ASC: {silhouette}')
print(f'DI: {dunn}')
```

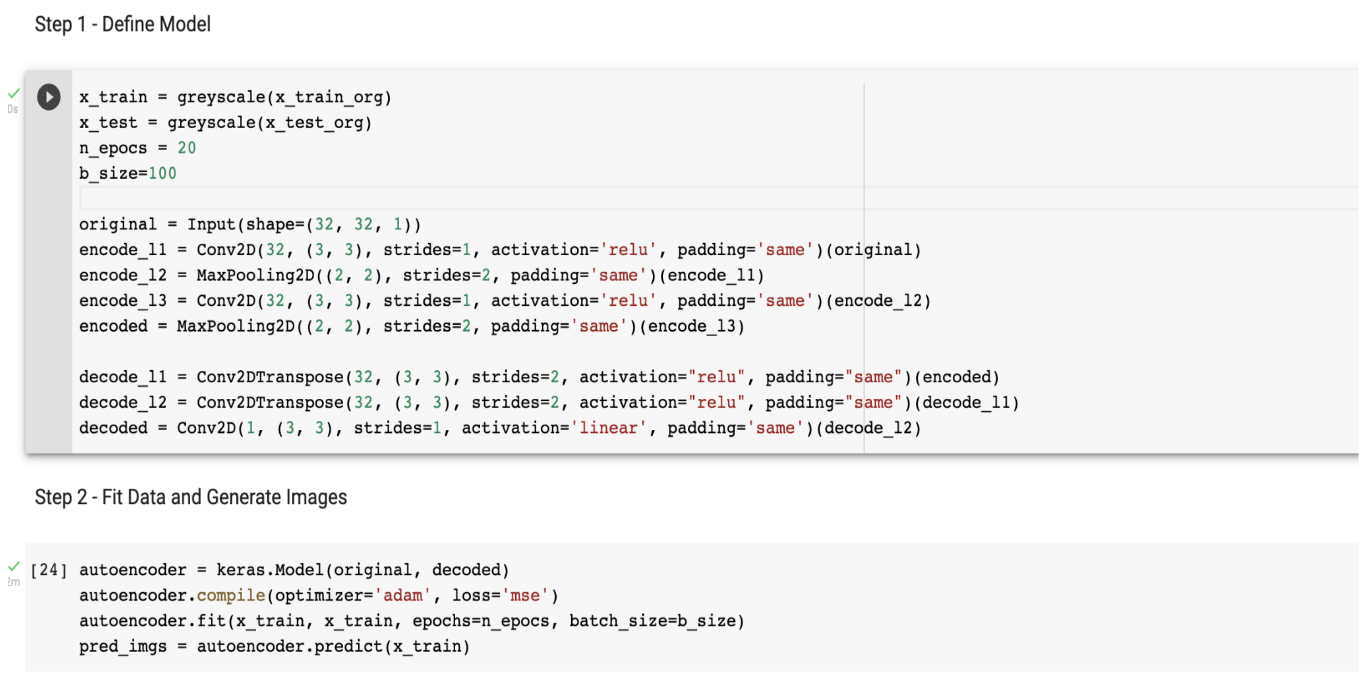
```
/home/sanyam/.local/lib/python3.8/site-packages/sklearn/utils/validation.py
umn-vector y was passed when a 1d array was expected. Please change the sha
ple using ravel().
  y = column_or_1d(y, warn=True)
ASC: 0.05587534665611295
DI: 0.09360165857070273
```

Fig 4 - Model Performance

5. Autoencoder

5.1 Training the Model

Multiple layers are used on both the Encoder and Decoder side. The resulting autoencoder has been trained on training data.


Step 1 - Define Model

```
x_train = greyscale(x_train_org)
x_test = greyscale(x_test_org)
n_epocs = 20
b_size=100

original = Input(shape=(32, 32, 1))
encode_l1 = Conv2D(32, (3, 3), strides=1, activation='relu', padding='same')(original)
encode_l2 = MaxPooling2D((2, 2), strides=2, padding='same')(encode_l1)
encode_l3 = Conv2D(32, (3, 3), strides=1, activation='relu', padding='same')(encode_l2)
encoded = MaxPooling2D((2, 2), strides=2, padding='same')(encode_l3)

decode_l1 = Conv2DTranspose(32, (3, 3), strides=2, activation="relu", padding="same")(encoded)
decode_l2 = Conv2DTranspose(32, (3, 3), strides=2, activation="relu", padding="same")(decode_l1)
decoded = Conv2D(1, (3, 3), strides=1, activation='linear', padding='same')(decode_l2)
```

Step 2 - Fit Data and Generate Images

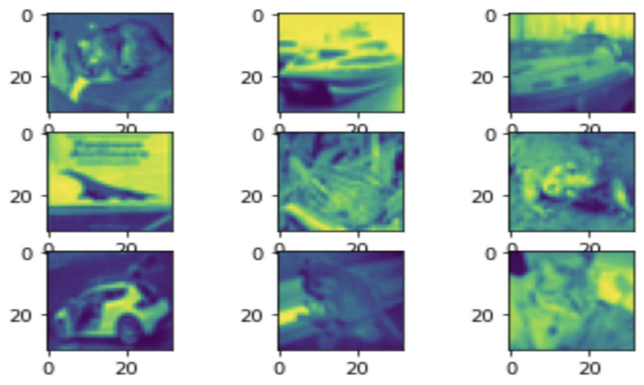
```
[24] autoencoder = keras.Model(original, decoded)
     autoencoder.compile(optimizer='adam', loss='mse')
     autoencoder.fit(x_train, x_train, epochs=n_epocs, batch_size=b_size)
     pred_imgs = autoencoder.predict(x_train)
```

Fig 5 - Autoencoder layer and Training

Post training, training data is used to get generated images. These generated images have been observed to be similar to original training data.

Original Images

```
[ ]  for i in range(9):
         pyplot.subplot(330 + 1 + i)
         pyplot.imshow(x_test[i].reshape(32, 32))
     pyplot.show()
```



Encoded images

```
for i in range(9):
    pyplot.subplot(330 + 1 + i)
    pyplot.imshow(pred_imgs[i].reshape(32,32))
pyplot.show()
```

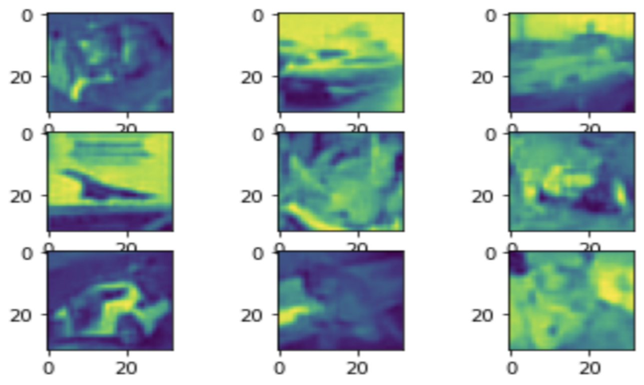
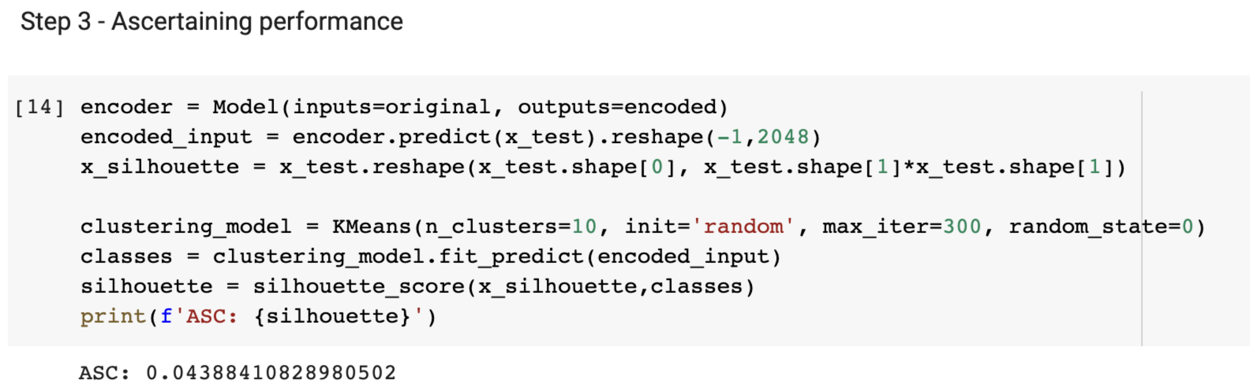


Fig 6 - Comparing Autoencoder output with Original Images

5.2 Ascertaining Performance

ASC (Average Silhouette Coefficient) on the encoder output has been calculated as `0.0438.`



Step 3 - Ascertaining performance

```
[14] encoder = Model(inputs=original, outputs=encoded)
     encoded_input = encoder.predict(x_test).reshape(-1,2048)
     x_silhouette = x_test.reshape(x_test.shape[0], x_test.shape[1]*x_test.shape[1])

     clustering_model = KMeans(n_clusters=10, init='random', max_iter=300, random_state=0)
     classes = clustering_model.fit_predict(encoded_input)
     silhouette = silhouette_score(x_silhouette,classes)
     print(f'ASC: {silhouette}')

     ASC: 0.04388410828980502
```

Fig 7 - Model Performance

References

https://blog.keras.io/building-autoencoders-in-keras.html
https://matplotlib.org/stable/tutorials/introductory/pyplot.html#sphx-glr-tutorials-introductory-pyplot-py