SHRI VILEPARLE KELAVANI MANDAL'S
DWARKADAS J. SANGHVI COLLEGE OF ENGINEERING
(Autonomous College Affiliated to the University of Mumbai)
NAAC ACCREDITED with "A" GRADE (CGPA : 3.18)
DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING (DATA SCIENCE)

Bhuvi Ghosh
60009210191

COURSE CODE: DJ19DSC501

COURSE NAME: Machine Learning - II

LAB EXPERIMENT NO. 8

AIM :

Compare the performance of PCA and Autoencoders on a given dataset

THEORY:

What is Dimensionality Reduction?

In machine learning classification problems, there are often too many factors on the basis of which the final classification is done. These factors are basically variables called features. The higher the number of features, the harder it gets to visualize the training set and then work on it. Sometimes, most of these features are correlated, and hence redundant. This is where dimensionality reduction algorithms come into play. Dimensionality reduction is the process of reducing the number of random variables under consideration, by obtaining a set of principal variables. It can be divided into feature selection and feature extraction.

Principle Component Analysis

Principle Component Analysis is an unsupervised technique where the original data is projected to the direction of high variance. These directions of high variance are orthogonal to each other resulting in very low or almost close to 0 correlation in the projected data. These features transformation is linear and the methodology to do it is:

Step 1: Calculate the Correlation matrix data consisting of n dimensions. The Correlation matrix will be of shape n*n.

Step 2: Calculate the Eigenvectors and Eigenvalues of this matrix.

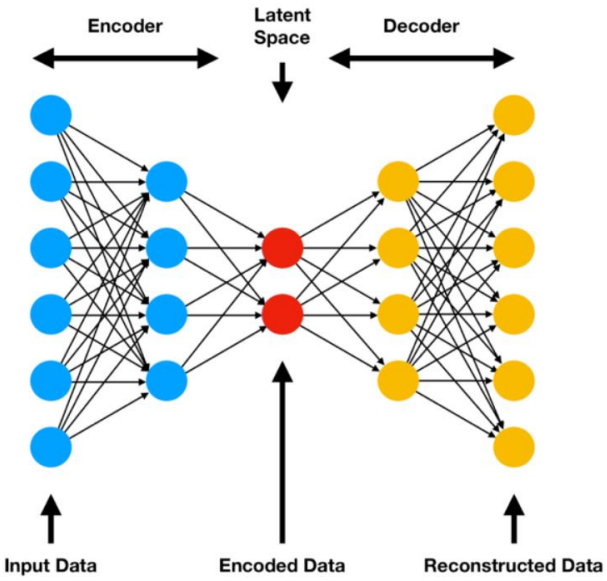
Step 3: Take the first k-eigenvectors with the highest eigenvalues.

Step 4: Project the original dataset into these k eigenvectors resulting in k dimensions where $k \leq n$.

Autoencoders

SHRI VILEPARLE KELAVANI MANDAL'S
DWARKADAS J. SANGHVI COLLEGE OF ENGINEERING
(Autonomous College Affiliated to the University of Mumbai)
NAAC ACCREDITED with "A" GRADE (CGPA : 3.18)

Autoencoder is an unsupervised artificial neural network that compresses the data to lower dimension and then reconstructs the input back. Autoencoder finds the representation of the data in a lower dimension by focusing more on the important features getting rid of noise and redundancy. It's based on Encoder-Decoder architecture, where encoder encodes the high-dimensional data to lower-dimension and decoder takes the lower-dimensional data and tries to reconstruct the original high-dimensional data.



Tasks to be performed:

1. **Use the Iris Dataset present in the scikit-learn library**
2. **Create an Auto Encoder and fit it with our data using 3 neurons in the dense layer**
3. **Use encoded layer to encode the training input**
4. **Plot loss for different encoders [PCA, Linear Autoencoder, Sigmoid based Non-Linear Autoencoder, ReLU based Non-Linear Auto encoder]**

```
import keras
class Histories(keras.callbacks.Callback):
    def on_train_begin(self, logs={}):
        self.weights = []

    def on_train_end(self, logs={}):
        return

    def on_epoch_begin(self, epoch, logs={}):
        return

    def on_epoch_end(self, epoch, logs={}):
        nx = []
        x = [x.reshape(-1) for x in self.model.get_weights()]
        for xi in x :
            nx += list(xi)
        self.weights.append(nx)
        return

    def on_batch_begin(self, batch, logs={}):
        return

    def on_batch_end(self, batch, logs={}):
        return
```

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt
%matplotlib inline
```

Importing Data

```
import seaborn as sns
```

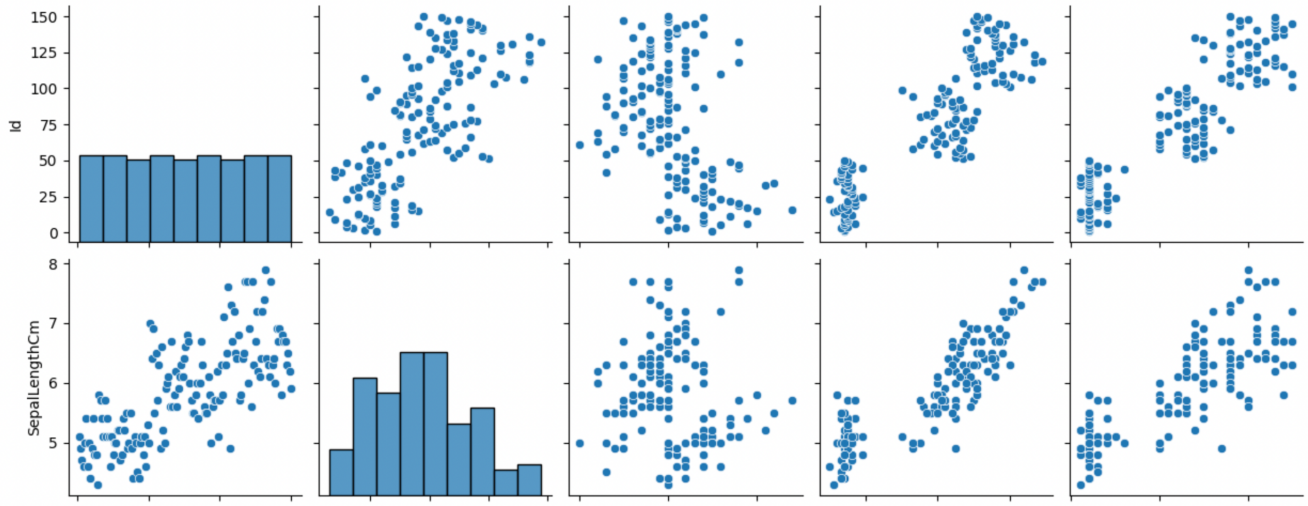
```
data = pd.read_csv("/content/Iris.xls")
x_train, x_test, y_train, y_test = train_test_split(data[['SepalLengthCm', 'SepalWidthCm',
                                                          'PetalLengthCm', 'PetalWidthCm']],
                                                    data['Species'],test_size=0.1, random_state=1)
```

```
x_train.head()
```

	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm
42	4.4	3.2	1.3	0.2
92	5.8	2.6	4.0	1.2
66	5.6	3.0	4.5	1.5
31	5.4	3.4	1.5	0.4
35	5.0	3.2	1.2	0.2

```
sns.pairplot(data)
```

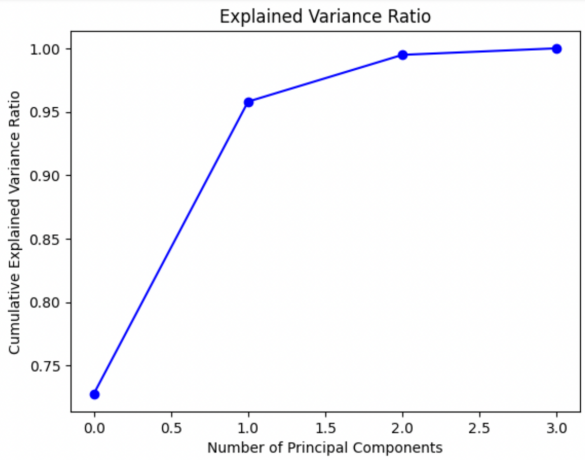
```
<seaborn.axisgrid.PairGrid at 0x7c9ea13e9690>
```



PCA

```
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler
import matplotlib.pyplot as plt

X = data[['SepalLengthCm', 'SepalWidthCm', 'PetalLengthCm', 'PetalWidthCm']]

scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
pca = PCA()

X_pca = pca.fit_transform(X_scaled)

explained_variance_ratio = pca.explained_variance_ratio_
cumulative_variance_ratio = explained_variance_ratio.cumsum()

print("Explained Variance Ratios:")
print(explained_variance_ratio)
print("\nCumulative Explained Variance Ratios:")
print(cumulative_variance_ratio)

plt.plot(cumulative_variance_ratio, marker='o', linestyle='-', color='b')
plt.title('Explained Variance Ratio')
plt.xlabel('Number of Principal Components')
plt.ylabel('Cumulative Explained Variance Ratio')
plt.show()
```

```
Explained Variance Ratios:
[0.72770452 0.23030523 0.03683832 0.00515193]

Cumulative Explained Variance Ratios:
[0.72770452 0.95800975 0.99484807 1.        ]
```




```
[ ] X = data[['SepalLengthCm', 'SepalWidthCm', 'PetalLengthCm', 'PetalWidthCm']]
    scaler = StandardScaler()
    X_scaled = scaler.fit_transform(X)
    pca = PCA(n_components=3)
    X_pca = pca.fit_transform(X_scaled)
    columns_pca = ['PC1', 'PC2', 'PC3']
    data_pca = pd.DataFrame(X_pca, columns=columns_pca)
    print("Data with 3 Principal Components:")
    print(data_pca.head())
```

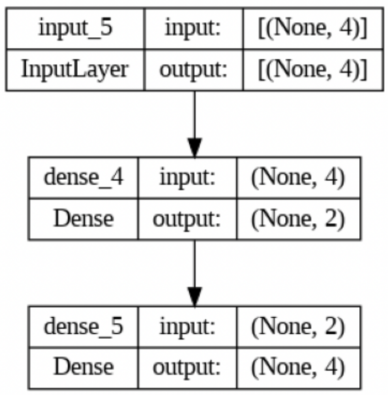
```
Data with 3 Principal Components:
        PC1       PC2       PC3
0 -2.264542  0.505704 -0.121943
1 -2.086426 -0.655405 -0.227251
2 -2.367950 -0.318477  0.051480
3 -2.304197 -0.575368  0.098860
4 -2.388777  0.674767  0.021428
```

```
from tensorflow.keras.utils import plot_model
```

Encoder and Decoder

```
from keras.layers import Input, Dense
from keras.models import Model
from keras.callbacks import TensorBoard

encoding_dim = 2
input_dim = 4

input_img = Input(shape=(input_dim,))
encoded = Dense(encoding_dim, activation='relu')(input_img)
decoded = Dense(input_dim, activation='relu')(encoded)

autoencoder = Model(input_img, decoded)
encoder = Model(input_img, encoded)

encoded_input = Input(shape=(encoding_dim,))

decoder_layer = autoencoder.layers[-1]

decoder = Model(encoded_input, decoder_layer(encoded_input))

autoencoder.compile(loss='mean_squared_error', optimizer='sgd')
```

```
plot_model(autoencoder, to_file='model_plot.png', show_shapes=True, show_layer_names=True)

from matplotlib import pyplot as plt
import matplotlib.image as mpimg
plt.axis("off")
plt.imshow(mpimg.imread('model_plot.png'))
plt.show()
```




```
x_hist = Histories()
while (True):
    x_hist = Histories()
    input_img = Input(shape=(input_dim,))
    encoded = Dense(encoding_dim, activation='relu')(input_img)
```

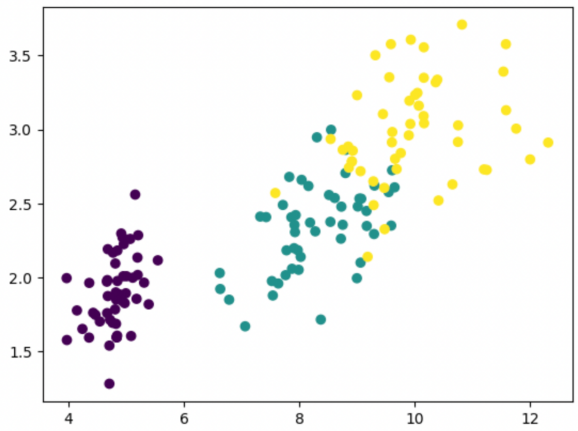
```
    decoded = Dense(input_dim, activation='relu')(encoded)

    autoencoder = Model(input_img, decoded)
    encoder = Model(input_img, encoded)

    encoded_input = Input(shape=(encoding_dim,))
    decoder_layer = autoencoder.layers[-1]
    decoder = Model(encoded_input, decoder_layer(encoded_input))

    autoencoder.compile(loss='mean_squared_error', optimizer='sgd')

    weight_list = []

    history = autoencoder.fit(x_train, x_train,
                    epochs=50,
                    batch_size=135,
                    shuffle=True,
                    validation_data=(x_test, x_test),
                    verbose=0,
                   callbacks=[TensorBoard(log_dir='/tmp/autoencoder'),x_hist])

    if (autoencoder.history.history['loss'][-1] < 1):
        break

encoded_datapoints = encoder.predict(x_test)
decoded_datapoints = decoder.predict(encoded_datapoints)

print('Original Datapoints :')
print(x_test)
print('Reconstructed Datapoints :')
print(decoded_datapoints)
```

```
1/1 [==============================] - 0s 70ms/step
Original Datapoints :
     SepalLengthCm  SepalWidthCm  PetalLengthCm  PetalWidthCm
14             5.8           4.0            1.2           0.2
98             5.1           2.5            3.0           1.1
75             6.6           3.0            4.4           1.4
16             5.4           3.9            1.3           0.4
131            7.9           3.8            6.4           2.0
56             6.3           3.3            4.7           1.6
141            6.9           3.1            5.1           2.3
44             5.1           3.8            1.9           0.4
29             4.7           3.2            1.6           0.2
120            6.9           3.2            5.7           2.3
94             5.6           2.7            4.2           1.3
5              5.4           3.9            1.7           0.4
102            7.1           3.0            5.9           2.1
51             6.4           3.2            4.5           1.5
78             6.0           2.9            4.5           1.5
Reconstructed Datapoints :
[[3.8861866  2.2641852  2.6718545  0.59395707]
 [4.8458805  2.4137897  3.3992846  1.0821751 ]
 [6.515975   3.0912404  4.618319   1.6196669 ]
 [3.7240713  2.316532   2.540253   0.45340264]
 [8.319489   4.104227   5.9031186  1.9895104 ]
 [6.4128423  3.3208048  4.5125465  1.3833852 ]
 [7.2638073  3.7592976  5.1231976  1.5874282 ]
 [3.8124924  2.3305445  2.6072485  0.4982134 ]
 [3.4923735  1.9265575  2.4043896  0.60032666]
 [7.5148363  3.8732777  5.305063   1.6591232 ]
 [5.758922   2.8043814  4.063459   1.3608848 ]
 [3.908957   2.3689847  2.6777375  0.52977467]
 [7.7510424  3.7347913  5.503801   1.9104713 ]
 [6.392281   3.2122216  4.508801   1.4517801 ]
 [6.1842356  3.0671654  4.363762   1.4302212 ]]
```

Plotting Encoded Features

```
[ ] encoded_dataset = encoder.predict(data[['SepalLengthCm', 'SepalWidthCm','PetalLengthCm', 'PetalWidthCm']])

    plt.scatter(encoded_dataset[:,0], encoded_dataset[:,1], c=data['Species'].astype('category').cat.codes)
    plt.show()
```

```
5/5 [==============================] - 0s 2ms/step
```




```
[ ] autoencoder.get_weights()
```

```
[array([[ 1.0686948 , -0.09897055],
        [-0.4750585 ,  0.62393624],
        [ 0.6740227 ,  0.0390758 ],
        [ 0.27554888,  0.76896095]], dtype=float32),
 array([0.03079419, 0.01261467], dtype=float32),
 array([[ 0.68016714,  0.15491283,  0.5100615 ,  0.3094366 ],
        [ 0.09637549,  0.683724  , -0.00208456, -0.45136687]],
       dtype=float32),
 array([ 1.4836501e-01, -2.5794725e-05,  2.7592625e-02, -4.9250249e-02],
       dtype=float32)]
```

```
encoded_datapoints = encoder.predict(x_test)
decoded_datapoints = decoder.predict(encoded_datapoints)

print('Original Datapoints :')
print(x_test)
print('Reconstructed Datapoints :')
print(decoded_datapoints)
```

```
1/1 [==============================] - 0s 64ms/step
1/1 [==============================] - 0s 69ms/step
Original Datapoints :
     SepalLengthCm  SepalWidthCm  PetalLengthCm  PetalWidthCm
14             5.8           4.0            1.2           0.2
98             5.1           2.5            3.0           1.1
75             6.6           3.0            4.4           1.4
16             5.4           3.9            1.3           0.4
131            7.9           3.8            6.4           2.0
56             6.3           3.3            4.7           1.6
141            6.9           3.1            5.1           2.3
44             5.1           3.8            1.9           0.4
29             4.7           3.2            1.6           0.2
120            6.9           3.2            5.7           2.3
94             5.6           2.7            4.2           1.3
5              5.4           3.9            1.7           0.4
102            7.1           3.0            5.9           2.1
51             6.4           3.2            4.5           1.5
78             6.0           2.9            4.5           1.5
Reconstructed Datapoints :
[[0.         0.         3.6658497  0.86394614]
 [0.         0.         3.2553642  1.0032729 ]
 [0.         0.         4.3556786  1.4257594 ]
 [0.         0.         3.475586   0.8218364 ]
 [0.         0.         5.6398067  1.9474009 ]
 [0.         0.         4.426694   1.4626595 ]
 [0.         0.         4.474325   1.4828995 ]
 [0.         0.         3.5645828  0.9332083 ]
 [0.         0.         3.1544197  0.8261607 ]
 [0.         0.         4.736352   1.6288234 ]
```

Conclusion: It is observed that eventhough both autoencoders & PCA are used in dimensionality reduction, PCA is preferred is preferred in case where data is linearly distributed. In case of non-linear distribution, autoencoders are preferred. PCA decides appropriate number of principle components based on the cummulative variance ratio whereas Autoencoders consider loss function to decide appropriate number of principle components.