**EXPERIMENT-17**

**Aim:** Implementing Dimensionality Reduction using Principal Component Analysis (PCA).

**Theory:**

Dimensionality reduction techniques like PCA help reduce the number of input variables in a dataset while retaining most of the important information.

PCA achieves this by finding a set of new uncorrelated variables (principal components) that successively maximize variance.

It is widely used in preprocessing for machine learning, visualization of high-dimensional data, and noise reduction.

**Code:**

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

from sklearn.decomposition import PCA

from sklearn.preprocessing import StandardScaler

df = pd.read\_csv('/content/tmnist.csv') # Replace with appropriate data file

print("Labels: ", df['labels'].unique())

no\_of\_classes = df['labels'].nunique()

X = df.drop(columns=['names', 'labels'], axis=1) # Drop unwanted columns

y = df['labels']

X\_std = StandardScaler().fit\_transform(X)

pca = PCA(n\_components=2) # Reduce to 2 dimensions

principal\_components = pca.fit\_transform(X\_std)

principal\_df = pd.DataFrame(data=principal\_components, columns=['Principal Component 1', 'Principal Component 2'])

# Concatenate with target labels for visualization

final\_df = pd.concat([principal\_df, y.reset\_index(drop=True)], axis=1)

# Visualize the data in the 2D PCA space

plt.figure(figsize=(10, 8))

targets = df['labels'].unique()

colors = ['r', 'g', 'b', 'c', 'm', 'y', 'k'] \* (len(targets) // 7 + 1) # Adjust number of colors

for target, color in zip(targets, colors):

indices\_to\_keep = final\_df['labels'] == target

plt.scatter(final\_df.loc[indices\_to\_keep, 'Principal Component 1'],

final\_df.loc[indices\_to\_keep, 'Principal Component 2'], c=color, s=50)

plt.xlabel('Principal Component 1')

plt.ylabel('Principal Component 2')

plt.title('2 Component PCA')

plt.legend(targets, loc='best')

plt.show()

explained\_variance = pca.explained\_variance\_ratio\_

print("Explained Variance Ratio:", explained\_variance)

print("Total Variance Explained by 2 Components:", sum(explained\_variance))

**Learning Outcomes:**

1. PCA effectively reduces the dimensionality of the dataset to 2 components.

2. Visualization in the 2D space helps understand class separability and data clustering.

3. The explained variance shows the amount of information retained by the selected principal components.

**EXPERIMENT-18**

**Aim:** Build an Artificial Neural Network (ANN) with backpropagation.

**Theory:**

An **ANN** is a model inspired by the human brain, consisting of input, hidden, and output layers. During training, data passes through these layers (forward pass), and the error is calculated using a loss function.

**Backpropagation** then adjusts the weights and biases by calculating gradients and updating them via optimization (e.g., gradient descent). This process is repeated over multiple epochs to minimize error.

**Hyperparameters:** Training an ANN involves tuning several hyperparameters:

* **Learning Rate:** Determines the step size for weight updates. A higher learning rate may lead to faster convergence, but it could also overshoot the optimal solution. A lower rate might make training slower but can yield a more accurate solution.
* **Batch Size:** Controls the number of samples passed through the network before updating the weights.
* **Epochs:** The number of complete passes through the entire training dataset.

**Code:**

import keras

import numpy as np

from keras.datasets import mnist

from keras.models import Sequential

from keras.layers import Dense

from keras.utils import to\_categorical

# Load dataset

(x\_train, y\_train), (x\_test, y\_test) = mnist.load\_data()

# Scale input values to [0, 1]

x\_train = x\_train.reshape(60000, 784).astype('float32') / 255

x\_test = x\_test.reshape(10000, 784).astype('float32') / 255

# Convert target values to one-hot encoding

y\_train = to\_categorical(y\_train, num\_classes=10)

y\_test = to\_categorical(y\_test, num\_classes=10)

# Build the model

model = Sequential([

Dense(10, activation='sigmoid', input\_shape=(784,)), # Hidden layer

Dense(10, activation='softmax') # Output layer

])

# Compile the model

model.compile(loss="categorical\_crossentropy",

optimizer="sgd",

metrics=['accuracy'])

# Train the model

history = model.fit(x\_train, y\_train, batch\_size=100, epochs=20) # here 20 epochs

# Evaluate the model on test data

test\_loss, test\_acc = model.evaluate(x\_test, y\_test)

print(f'Test accuracy: {round(test\_acc, 4)}')

# Display a sample prediction

example\_index = 11

prediction = model.predict(x\_test[example\_index].reshape(1, 784))

print("Predicted label:", np.argmax(prediction))

print("True label:", np.argmax(y\_test[example\_index]))

**Learning Outcomes:**

1. The ANN trained with backpropagation achieves a certain accuracy on the MNIST test data.

2. The confusion matrix visualizes the performance of the model across different classes.

3. Predictions for a sample input provide insight into model's confidence levels for each class.

**Result:**

**Accuracy: 87.37% for 20 epochs**

More epoch, better the model is trained on dataset and higher is the accuracy.

**Output:**



**EXPERIMENT-19**

**Aim:** Image Classification on Fashion MNIST Dataset using Artificial Neural Network (ANN)

**Theory:**

**Fashion MNIST** is a dataset of grayscale images of 10 different categories of clothing items (like shirts, shoes, and coats). The dataset includes **60,000** training images and 10,000 test images, each of **28x28 pixels.**

Each image is labeled with one of the **10 classes**. Fashion MNIST is often used as a drop-in replacement for MNIST to test image classification models in a slightly more complex setting.

**Artificial Neural Networks (ANN)** are a powerful model type for image classification. They consist of layers of nodes (neurons) where each neuron takes input data, applies a transformation, and passes the output to the next layer.

Using backpropagation, ANN adjusts weights through training to improve accuracy. In image classification, ANN learns patterns across pixel values to classify images accurately.

**Code:**

import numpy as np

import tensorflow as tf

from tensorflow.keras.datasets import fashion\_mnist

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Dense, Flatten

from tensorflow.keras.utils import to\_categorical

import matplotlib.pyplot as plt

# Load the Fashion MNIST dataset

(X\_train, y\_train), (X\_test, y\_test) = fashion\_mnist.load\_data()

# Data Preprocessing

X\_train = X\_train / 255.0 # Scale pixel values to [0, 1]

X\_test = X\_test / 255.0

y\_train = to\_categorical(y\_train, 10) # One-hot encode labels

y\_test = to\_categorical(y\_test, 10)

# Building the Model

model = Sequential([

Flatten(input\_shape=(28, 28)), # Flatten 28x28 images to 784-element vectors

Dense(128, activation='relu'), # Hidden layer with 128 neurons

Dense(10, activation='softmax') # Output layer for 10 classes

])

# Compile the Model

model.compile(optimizer='adam',

loss='categorical\_crossentropy',

metrics=['accuracy'])

# Train the Model

history = model.fit(X\_train, y\_train, epochs=10, batch\_size=32, validation\_split=0.2)

# Evaluate the Model on Test Data

test\_loss, test\_accuracy = model.evaluate(X\_test, y\_test)

print(f'Test Accuracy: {test\_accuracy:.4f}')

# Plot Training and Validation Accuracy

plt.plot(history.history['accuracy'], label='Training Accuracy')

plt.plot(history.history['val\_accuracy'], label='Validation Accuracy')

plt.xlabel('Epochs')

plt.ylabel('Accuracy')

plt.legend()

plt.title('Training and Validation Accuracy Over Epochs')

plt.show()

# Predict a sample from test data

sample\_index = 0

sample\_image = X\_test[sample\_index].reshape(1, 28, 28) # Reshape for prediction

sample\_label = np.argmax(model.predict(sample\_image), axis=1)

print("Predicted label for sample image:", sample\_label[0])

**Learning Outcomes:**

1. **Data Normalization**: Scaling image pixel values between 0 and 1 improves ANN performance.
2. **Simple Architecture Works**: Even a basic ANN can effectively classify Fashion MNIST images.
3. **Validation Helps**: Monitoring validation accuracy during training detects overfitting early.

**EXPERIMENT-20**

**Aim:** Build an Artificial Neural Network (ANN) on MNIST Dataset.

**Theory:**

The MNIST dataset contains 60,000 training and 10,000 test images of handwritten digits (0-9), each of size 28x28 pixels.

An Artificial Neural Network (ANN) is a deep learning model inspired by the human brain, consisting of layers of interconnected neurons.

Key components of ANN : input layer, hidden layers, output layer, activation functions, optimizer, loss functions

In the case of image classification, an ANN learns to identify patterns in the pixel values of images to classify them into different categories.

**Code:**

import numpy as np

import tensorflow as tf

from tensorflow.keras.datasets import mnist

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Dense, Flatten

from tensorflow.keras.utils import to\_categorical

import matplotlib.pyplot as plt

# Load MNIST dataset

(X\_train, y\_train), (X\_test, y\_test) = mnist.load\_data()

# Data Preprocessing

X\_train = X\_train / 255.0 # Normalize pixel values to [0, 1]

X\_test = X\_test / 255.0

y\_train = to\_categorical(y\_train, 10) # One-hot encode labels

y\_test = to\_categorical(y\_test, 10)

# Build the model

model = Sequential([

Flatten(input\_shape=(28, 28)), # Flatten the 28x28 images into 784 values

Dense(128, activation='relu'), # Hidden layer with 128 neurons

Dense(10, activation='softmax') # Output layer for 10 classes

])

# Compile the model

model.compile(optimizer='adam',

loss='categorical\_crossentropy',

metrics=['accuracy'])

# Train the model

history = model.fit(X\_train, y\_train, epochs=10, batch\_size=32, validation\_split=0.2)

# Evaluate the model on the test dataset

test\_loss, test\_acc = model.evaluate(X\_test, y\_test)

print(f'Test Accuracy: {test\_acc:.4f}')

# Plot the training and validation accuracy

plt.plot(history.history['accuracy'], label='Training Accuracy')

plt.plot(history.history['val\_accuracy'], label='Validation Accuracy')

plt.xlabel('Epochs')

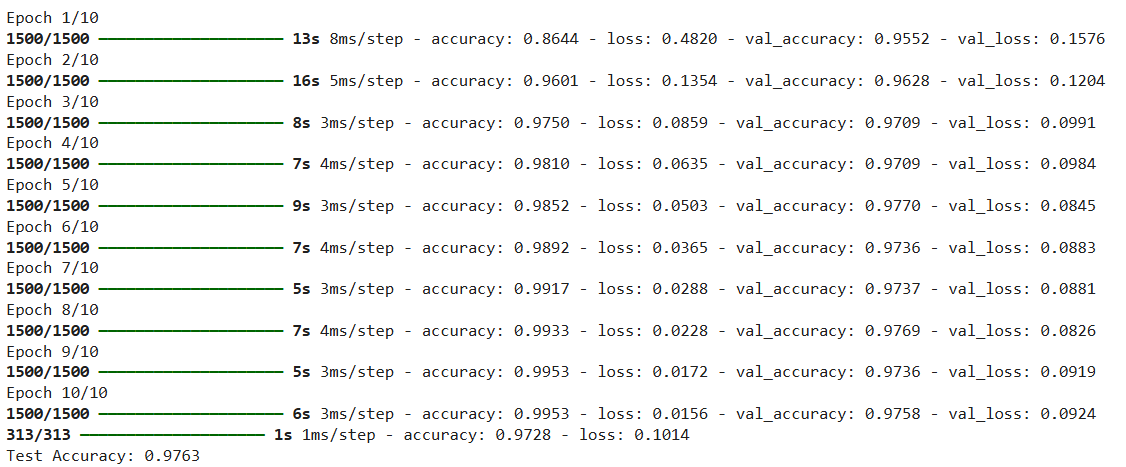
plt.ylabel('Accuracy')

plt.legend()

plt.title('Training and Validation Accuracy Over Epochs')

plt.show()

**Output:**

****

**Accuracy: 97.63 % for 10 epochs**

**EXPERIMENT-13**

**Aim**: To implement Data classification using K-Means.

**Theory:**  
K-Means is an unsupervised machine learning algorithm commonly used for clustering tasks. It groups data points into clusters based on their similarity, aiming to minimize the sum of squared distances between data points and their respective cluster centroids.

**Key Concepts:**

1. **Objective:** The main objective of K-Means is to partition data into clusters so that points within the same cluster are more similar to each other than to points in other clusters.
2. **K Centroids:** The algorithm requires specifying the number of clusters (K) in advance. It initializes K centroids randomly, then assigns each data point to the nearest centroid.
3. **Iterations:** K-Means operates iteratively, calculating the distance of each data point from the centroids, reassigning points to the nearest centroid, and recalculating centroids until convergence.
4. **Advantages:** Simple, fast, and efficient for clustering large datasets. Effectively discovers clusters with spherical shapes.
5. **Disadvantages:** Sensitive to the initial placement of centroids. May not perform well on data with non-spherical clusters.

**Code:**

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

from sklearn.datasets import load\_iris

from sklearn.cluster import KMeans

from sklearn.preprocessing import StandardScaler

# Load the Iris dataset

iris = load\_iris()

X = iris.data  # Features (sepal length, sepal width, petal length, petal width)

y = iris.target  # Actual species (used for comparison later)

# Feature scaling for better K-Means convergence

scaler = StandardScaler()

X\_scaled = scaler.fit\_transform(X)

# clustering with 3 clusters (since we know there are 3 species)

kmeans = KMeans(n\_clusters=3, random\_state=42)

y\_pred = kmeans.fit\_predict(X\_scaled)

# Visualize the clusters (using the first two features for simplicity)

plt.figure(figsize=(10, 6))

# Plot each cluster with a unique color and label

for cluster\_label in np.unique(y\_pred):

    plt.scatter(

        X\_scaled[y\_pred == cluster\_label, 0],

        X\_scaled[y\_pred == cluster\_label, 1],

        label=f"Cluster {cluster\_label}",

        s=50

    )

plt.scatter(

    kmeans.cluster\_centers\_[:, 0],

    kmeans.cluster\_centers\_[:, 1],

    c='red', marker='X', s=200, label="Centroids"

)

plt.xlabel('Scaled Sepal Length')

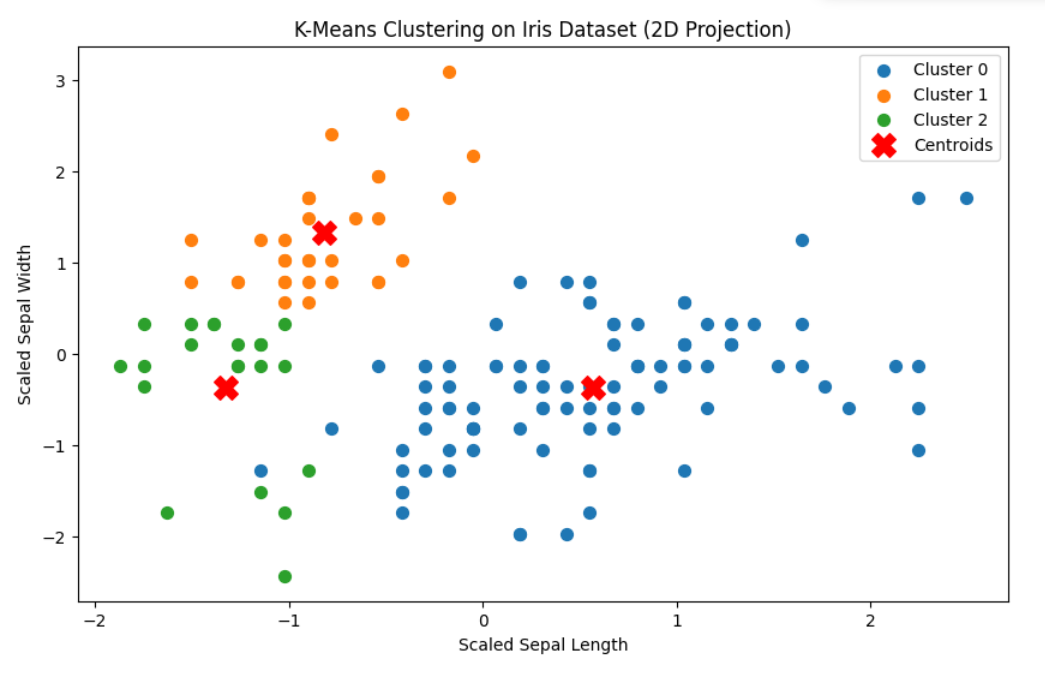
plt.ylabel('Scaled Sepal Width')

plt.title('K-Means Clustering on Iris Dataset (2D Projection)') 13

plt.legend()

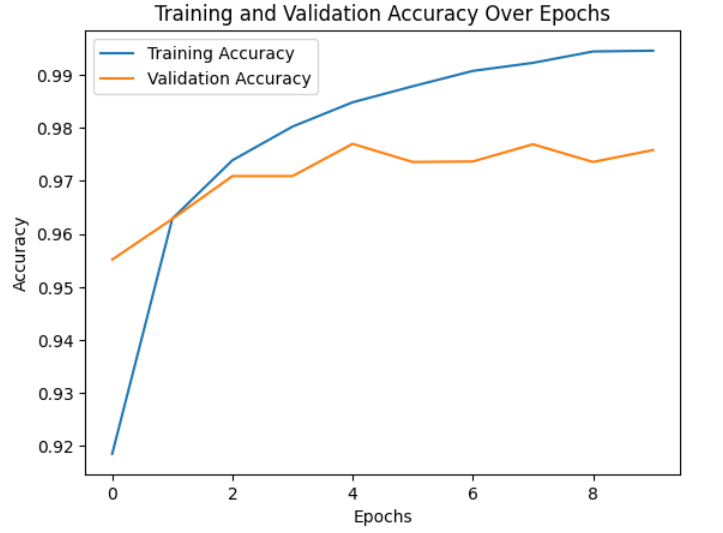
plt.show()

**Output:**



**Learning Outcomes:**

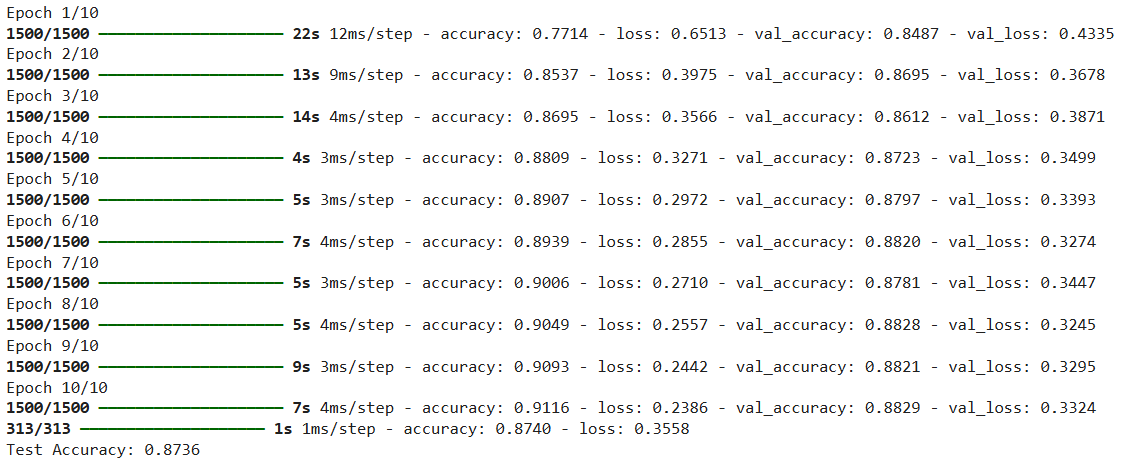
1. Gained an understanding of K-Means clustering, including its iterative process to achieve cluster convergence.
2. Recognized the simplicity and efficiency of K-Means for clustering tasks, especially in large datasets.
3. Identified the limitations of K-Means, such as sensitivity to centroid initialization and challenges with non-spherical clusters.

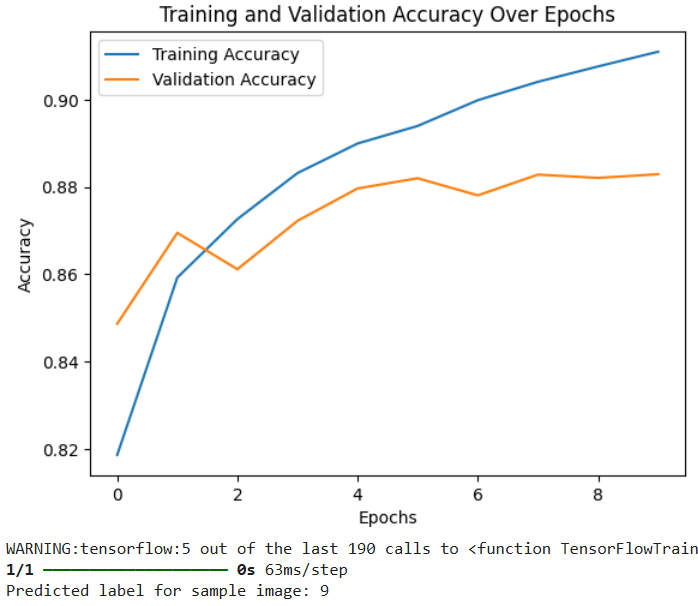
**20**

**Learning Outcomes:**

1. **Data Preprocessing**: Normalizing image pixel values improves the ANN's ability to learn effectively.
2. **Activation Functions**: Using functions like ReLU and softmax helps the model learn complex patterns and output probabilities for classification.
3. **Model Evaluation**: Tracking accuracy and loss during training helps ensure the model is learning correctly and prevents overfitting.

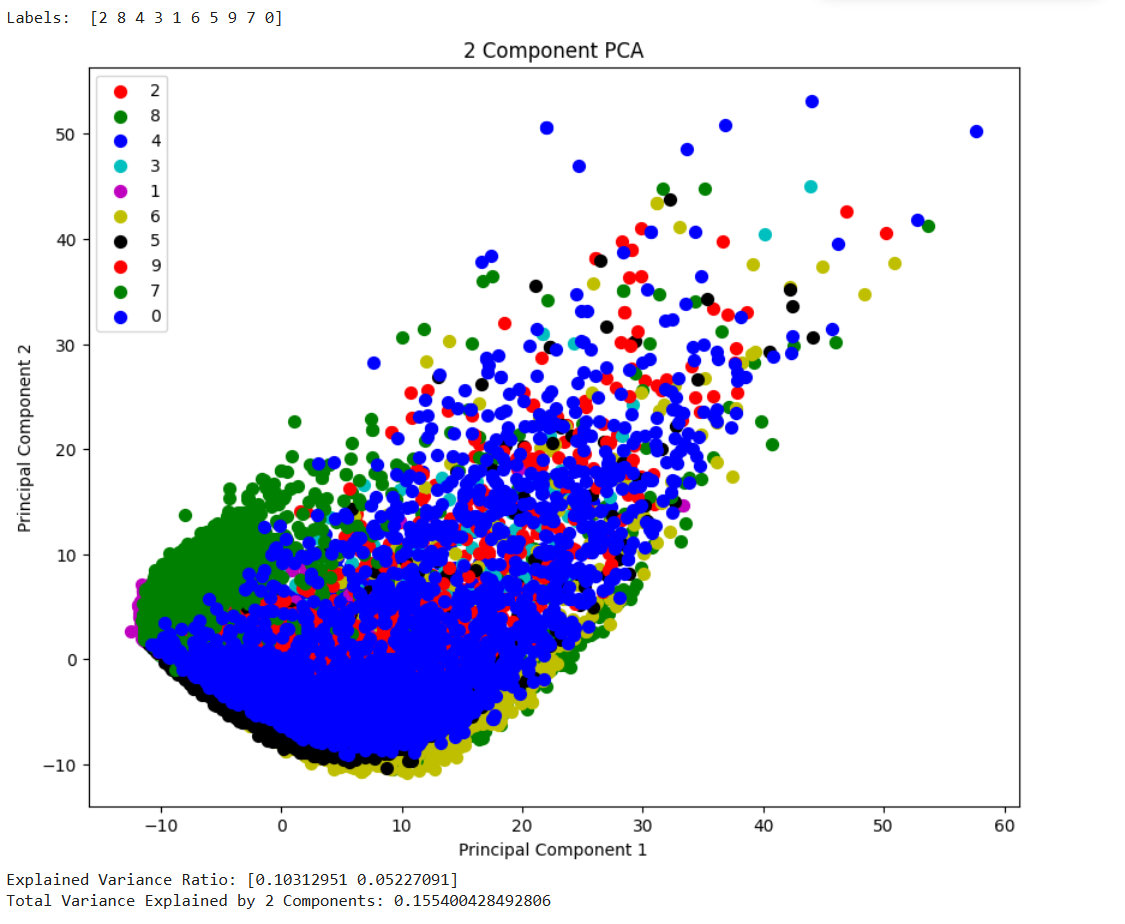
**Output:**

**  
Accuracy: 87.36% for 10 epochs**

****

19

**Output:**



17

**DELHI TECHNOLOGICAL UNIVERSITY**

**DEPARTMENT OF**

**COMPUTER SCIENCE ENGINEERING**

****

**CO327   
Machine Learning**

**SUBMITTED BY:**

Ayush Tandon

2K22/CO/133

**DELHI TECHNOLOGICAL UNIVERSITY**

**DEPARTMENT OF**

**COMPUTER SCIENCE ENGINEERING**

****

**CO301   
Software Engineering**

**SUBMITTED TO: SUBMITTED BY:**

Lavindra Gautam Ayush Tandon

2K22/CO/133