ML_HOMEWORK_5
ELIZABETH AMEKE
662055975

Question_1
Construct a convolutional neural network model for classifying the CIFAR-10 dataset. Use the test set of the CIFAR-10 dataset as validation data for the model

(a) Implement the convolutional neural network architecture given below.
i. The first layer is a 2D convolutional layer with 64 filters, each of size (5, 5), and uses the ReLU activation function.
The input shape of the layer should correspond to the dimensions of the input image.
ii. The second layer is a max pooling layer of size (2, 2).
iii. The third layer is another 2D convolutional layer with 32 filters, each of size (3, 3), and uses the ReLU activation function.
iv. The fourth layer is another max pooling layer of size (2, 2).
v. The fifth layer is another 2D convolutional layer with 32 filters, each of size (3, 3), and uses the ReLU activation function.
vi. The sixth layer is a flattened layer which converts the output of the previous layer into a one-dimensional vector.
vii. The seventh layer is a dense layer with 64 neurons and uses the ReLU activation function.
viii. The eighth and final layer produces estimated probabilities to classify the CIFAR-10 classes.

```
#(a)
##Importing the data

import tensorflow as tf
from tensorflow.keras.datasets import cifar10

# Loading the CIFAR-10 dataset
(train_images, train_labels), (test_images, test_labels) = cifar10.load_data()

# Printing the shapes of the loaded data
print("Train images shape:", train_images.shape)
print("Train labels shape:", train_labels.shape)
print("Test images shape:", test_images.shape)
print("Test labels shape:", test_labels.shape)
```

```
Train images shape: (50000, 32, 32, 3)
Train labels shape: (50000, 1)
```

```
Test images shape: (10000, 32, 32, 3)
Test labels shape: (10000, 1)
```

```
#i)
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, Activation

# Creating a Sequential model
model = Sequential()

# Adding the first convolutional layer
model.add(Conv2D(64, (5, 5), activation='relu', input_shape=train_images.shape[1:]))

# Printing a summary of the model to verify the architecture
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 28, 28, 64)        4864

=================================================================
Total params: 4864 (19.00 KB)
Trainable params: 4864 (19.00 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

```
#ii)
from tensorflow.keras.layers import MaxPooling2D

# Adding the second max pooling layer
model.add(MaxPooling2D(pool_size=(2, 2)))

# Printing a summary of the model to verify the architecture
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 28, 28, 64)        4864

 max_pooling2d (MaxPooling2  (None, 14, 14, 64)        0
```

```
 D)

=================================================================
Total params: 4864 (19.00 KB)
Trainable params: 4864 (19.00 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

```
#iii)
# Adding the third convolutional layer
model.add(Conv2D(32, (3, 3), activation='relu'))

# Printing a summary of the model to verify the architecture
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 28, 28, 64)        4864

 max_pooling2d (MaxPooling2  (None, 14, 14, 64)        0
 D)

 conv2d_1 (Conv2D)           (None, 12, 12, 32)        18464

=================================================================
Total params: 23328 (91.12 KB)
Trainable params: 23328 (91.12 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

```
#iv)
# Adding the fourth max pooling layer
model.add(MaxPooling2D(pool_size=(2, 2)))

# Printing a summary of the model to verify the architecture
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 28, 28, 64)        4864
```

```
 max_pooling2d (MaxPooling2  (None, 14, 14, 64)        0
 D)

 conv2d_1 (Conv2D)           (None, 12, 12, 32)        18464

 max_pooling2d_1 (MaxPoolin  (None, 6, 6, 32)          0
 g2D)

=================================================================
Total params: 23328 (91.12 KB)
Trainable params: 23328 (91.12 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

```
#v)
# Adding the fifth convolutional layer
model.add(Conv2D(32, (3, 3), activation='relu'))

# Printing a summary of the model to verify the architecture
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 28, 28, 64)        4864

 max_pooling2d (MaxPooling2  (None, 14, 14, 64)        0
 D)

 conv2d_1 (Conv2D)           (None, 12, 12, 32)        18464

 max_pooling2d_1 (MaxPoolin  (None, 6, 6, 32)          0
 g2D)

 conv2d_2 (Conv2D)           (None, 4, 4, 32)          9248

=================================================================
Total params: 32576 (127.25 KB)
Trainable params: 32576 (127.25 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

```
#vi)
from tensorflow.keras.layers import Flatten
```

```
# Adding the sixth flattened layer
model.add(Flatten())

# Printing a summary of the model to verify the architecture
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 conv2d (Conv2D)             (None, 28, 28, 64)        4864      
                                                                 
 max_pooling2d (MaxPooling2  (None, 14, 14, 64)        0         
 D)                                                              
                                                                 
 conv2d_1 (Conv2D)           (None, 12, 12, 32)        18464     
                                                                 
 max_pooling2d_1 (MaxPoolin  (None, 6, 6, 32)          0         
 g2D)                                                            
                                                                 
 conv2d_2 (Conv2D)           (None, 4, 4, 32)          9248      
                                                                 
 flatten (Flatten)           (None, 512)               0         
                                                                 
=================================================================
Total params: 32576 (127.25 KB)
Trainable params: 32576 (127.25 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

```
#vii)
from tensorflow.keras.layers import Dense

# Adding the seventh dense layer
model.add(Dense(64, activation='relu'))

# Printing a summary of the model to verify the architecture
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 conv2d (Conv2D)             (None, 28, 28, 64)        4864      
```

```
 max_pooling2d (MaxPooling2  (None, 14, 14, 64)        0
 D)

 conv2d_1 (Conv2D)           (None, 12, 12, 32)        18464

 max_pooling2d_1 (MaxPoolin  (None, 6, 6, 32)          0
 g2D)

 conv2d_2 (Conv2D)           (None, 4, 4, 32)          9248

 flatten (Flatten)           (None, 512)               0

 dense (Dense)               (None, 64)                32832

=================================================================
Total params: 65408 (255.50 KB)
Trainable params: 65408 (255.50 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

```
#viii)
# Adding the eighth dense layer for classification
model.add(Dense(10, activation='softmax'))

# Printing a summary of the final model architecture
model.summary()
```

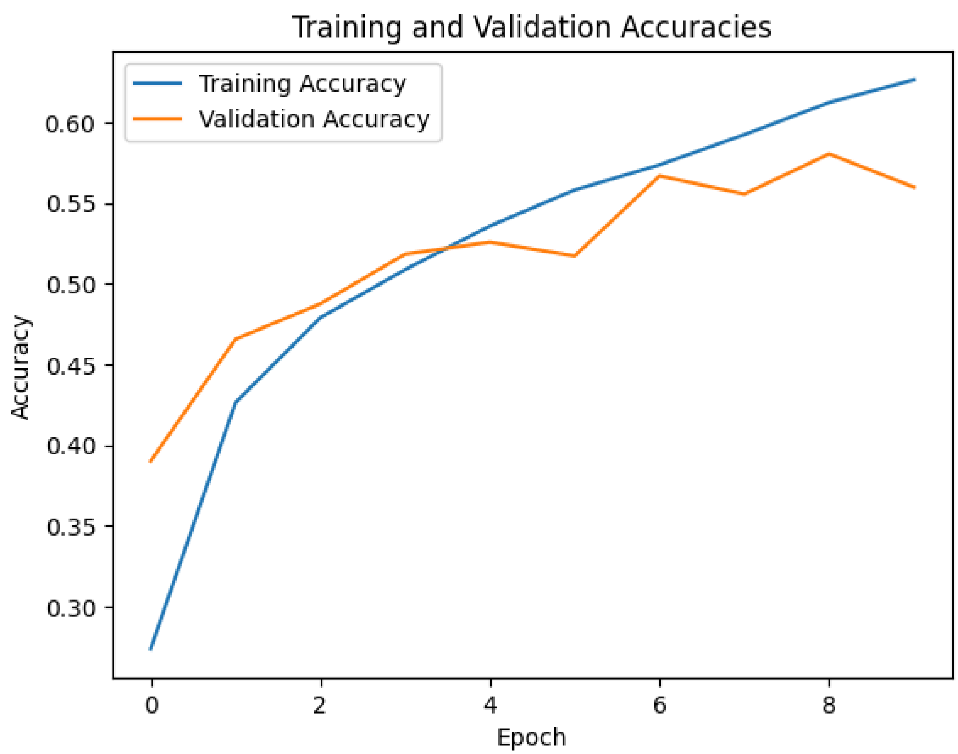
```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 28, 28, 64)        4864

 max_pooling2d (MaxPooling2  (None, 14, 14, 64)        0
 D)

 conv2d_1 (Conv2D)           (None, 12, 12, 32)        18464

 max_pooling2d_1 (MaxPoolin  (None, 6, 6, 32)          0
 g2D)

 conv2d_2 (Conv2D)           (None, 4, 4, 32)          9248

 flatten (Flatten)           (None, 512)               0

 dense (Dense)               (None, 64)                32832
```

```
 dense_1 (Dense)             (None, 10)                650

=================================================================
Total params: 66058 (258.04 KB)
Trainable params: 66058 (258.04 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

.

```
#(b)
import matplotlib.pyplot as plt

# Compiling the model
model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])

# Training the model
history = model.fit(train_images, train_labels, epochs=10, batch_size=64, validation_data=(test_images, test_labels))

# Plotting the training and validation accuracies
plt.plot(history.history['accuracy'], label='Training Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()
plt.title('Training and Validation Accuracies')
plt.show()
```

```
Epoch 1/10
782/782 [==============================] - 93s 115ms/step - loss: 2.2659 - accuracy: 0.2739 - val_loss: 1.6834 - val_accuracy: 0.3902
Epoch 2/10
782/782 [==============================] - 84s 108ms/step - loss: 1.5808 - accuracy: 0.4264 - val_loss: 1.4632 - val_accuracy: 0.4657
Epoch 3/10
782/782 [==============================] - 82s 104ms/step - loss: 1.4493 - accuracy: 0.4790 - val_loss: 1.4274 - val_accuracy: 0.4876
Epoch 4/10
782/782 [==============================] - 94s 121ms/step - loss: 1.3762 - accuracy: 0.5088 - val_loss: 1.3485 - val_accuracy: 0.5184
Epoch 5/10
782/782 [==============================] - 89s 114ms/step - loss: 1.3105 - accuracy: 0.5358 - val_loss: 1.3293 - val_accuracy: 0.5258
Epoch 6/10
782/782 [==============================] - 81s 104ms/step - loss: 1.2514 - accuracy: 0.5582 - val_loss: 1.3504 - val_accuracy: 0.5173
Epoch 7/10
782/782 [==============================] - 100s 128ms/step - loss: 1.2099 - accuracy: 0.5737 - val_loss: 1.2314 - val_accuracy: 0.5669
Epoch 8/10
782/782 [==============================] - 83s 107ms/step - loss: 1.1570 - accuracy: 0.5924 - val_loss: 1.2784 - val_accuracy: 0.5556
Epoch 9/10
782/782 [==============================] - 84s 107ms/step - loss: 1.1063 - accuracy: 0.6123 - val_loss: 1.2187 - val_accuracy: 0.5805
Epoch 10/10
782/782 [==============================] - 83s 106ms/step - loss: 1.0644 - accuracy: 0.6264 - val_loss: 1.3053 - val_accuracy: 0.5599
```



.

.

Question_2

Use the Scikit-learn breast cancer Wisconsin dataset and support vector machine classifiers to classify breast cancers. You must use worst compactness, worst concavity, and worst area features only to perform the classification.
(a) What is the accuracy of the classification model with a linear kernel?
(b) What is the accuracy of the classification model with a radial basis function kernel with regularization strength parameter, C=2?

```
from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.svm import SVC
from sklearn.metrics import classification_report, accuracy_score

# Loading the breast cancer dataset
data = load_breast_cancer()

# Extracting only the required features: worst compactness, worst concavity, and worst area
X = data.data[:, [20, 22, 23]]  # Features at indices 20, 22, and 23
y = data.target  # Target variable (0: malignant, 1: benign)

# Splitting the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Standardizing the features
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Initializing and training the SVM classifier
svm_classifier = SVC(kernel='linear', random_state=42)
svm_classifier.fit(X_train_scaled, y_train)

# Making predictions
y_pred = svm_classifier.predict(X_test_scaled)

# Classifing
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)

# Displaying the classification report
```

```
print("\nClassification Report:")
print(classification_report(y_test, y_pred))
```

```
Accuracy: 0.956140350877193

Classification Report:
              precision    recall  f1-score   support

           0       1.00      0.88      0.94        43
           1       0.93      1.00      0.97        71

    accuracy                           0.96       114
   macro avg       0.97      0.94      0.95       114
weighted avg       0.96      0.96      0.96       114
```

.

```
#(a)
# Evaluating the classifier
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)
```

```
Accuracy: 0.956140350877193
```

Accuracy with linear kernel = 0.956140350877193

.

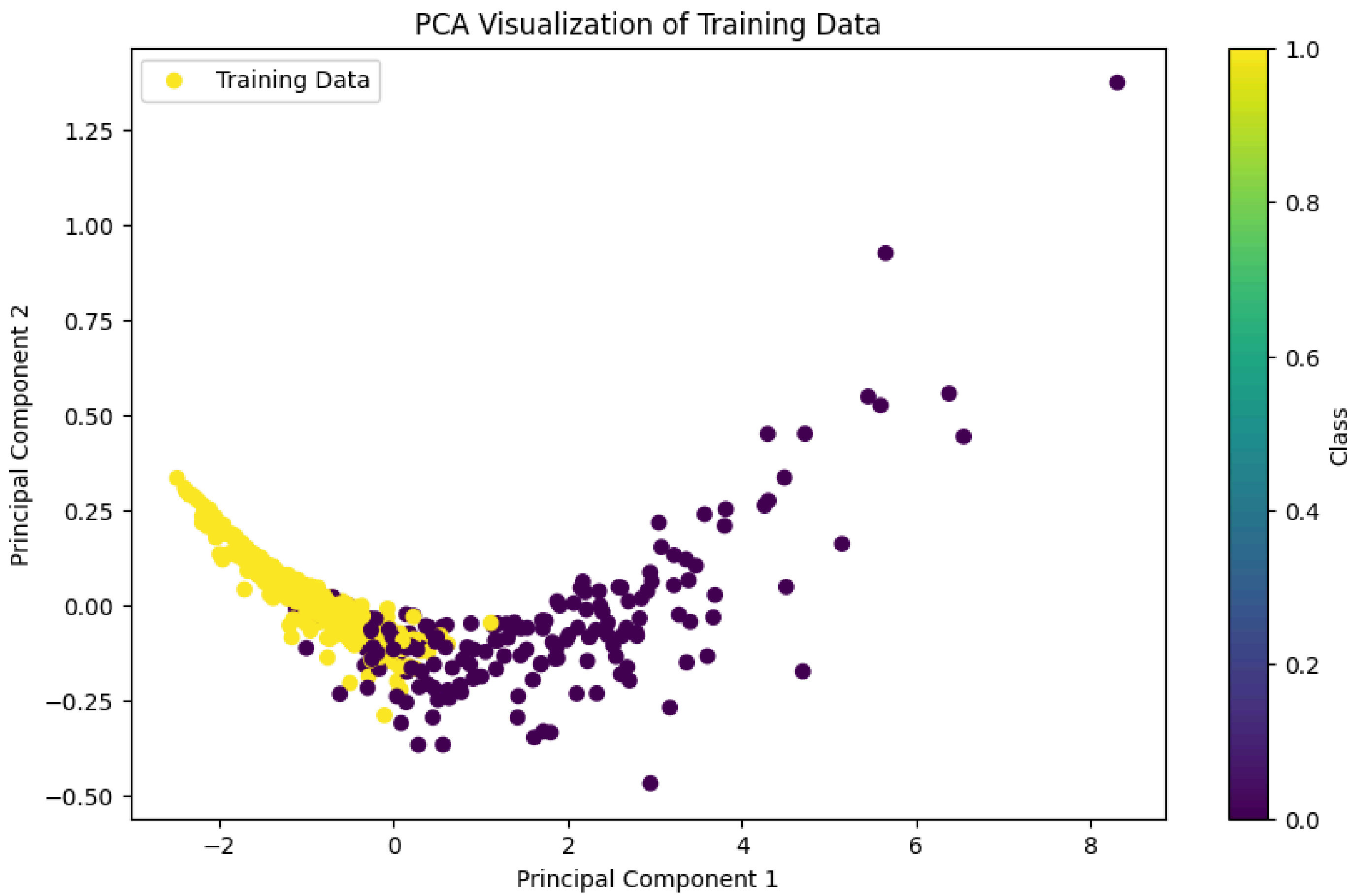
```
#(b)
# Initializing and training the SVM classifier with RBF kernel and C=2
svm_classifier_rbf = SVC(kernel='rbf', C=2, random_state=42)
svm_classifier_rbf.fit(X_train_scaled, y_train)

# Make predictions using the RBF kernel classifier
y_pred_rbf = svm_classifier_rbf.predict(X_test_scaled)

# Evaluating the RBF kernel classifier
accuracy_rbf = accuracy_score(y_test, y_pred_rbf)
print("Accuracy with RBF kernel and C=2:", accuracy_rbf)
```

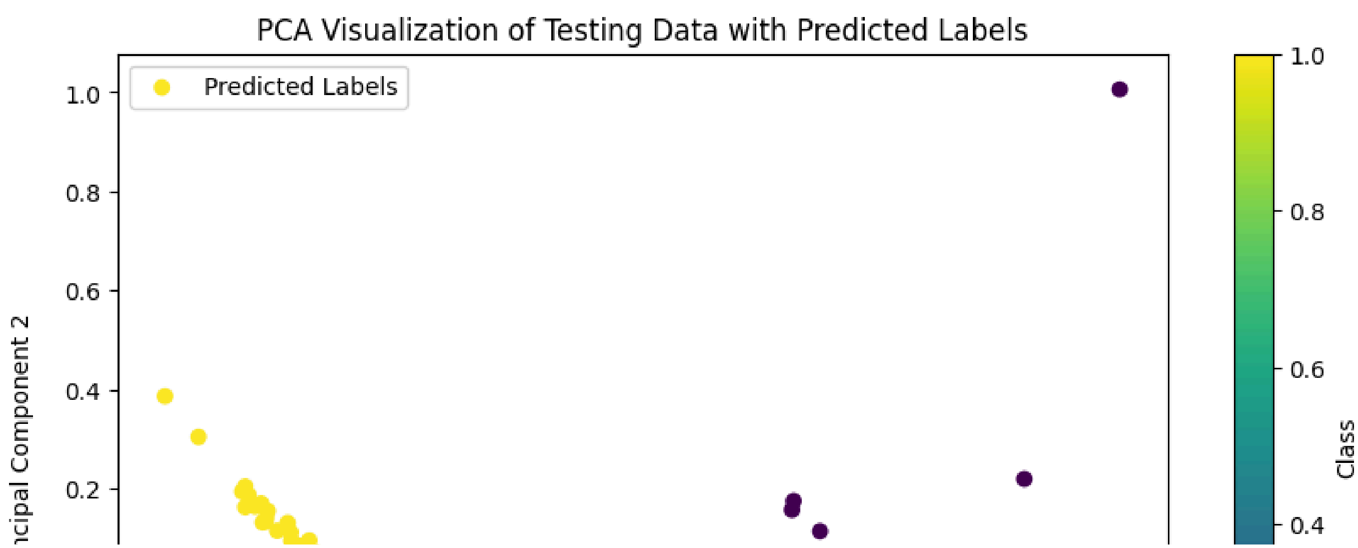
```
Accuracy with RBF kernel and C=2: 0.9473684210526315
```

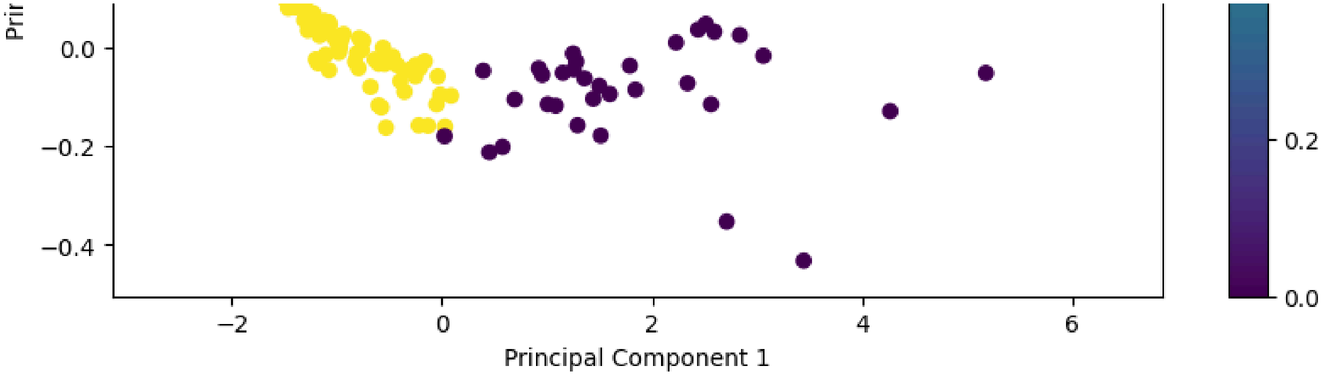
Accuracy with RBF kernel and C=2: 0.9473684210526315

.

```
#Visualizing the results

import matplotlib.pyplot as plt
from sklearn.decomposition import PCA

# Performing dimensionality reduction using PCA for visualization purposes
pca = PCA(n_components=2)
X_train_pca = pca.fit_transform(X_train_scaled)
X_test_pca = pca.transform(X_test_scaled)

# Plotting the training data
plt.figure(figsize=(10, 6))
plt.scatter(X_train_pca[:, 0], X_train_pca[:, 1], c=y_train, cmap='viridis', label='Training Data')
plt.colorbar(label='Class')
plt.xlabel('Principal Component 1')
plt.ylabel('Principal Component 2')
plt.title('PCA Visualization of Training Data')
plt.legend()
plt.show()

# Plotting the testing data with predicted labels
plt.figure(figsize=(10, 6))
plt.scatter(X_test_pca[:, 0], X_test_pca[:, 1], c=y_pred, cmap='viridis', label='Predicted Labels')
plt.colorbar(label='Class')
plt.xlabel('Principal Component 1')
plt.ylabel('Principal Component 2')
plt.title('PCA Visualization of Testing Data with Predicted Labels')
plt.legend()
plt.show()
```


PCA Visualization of Training Data
Training Data
Principal Component 1
Principal Component 2
Class


PCA Visualization of Testing Data with Predicted Labels
Predicted Labels
Class


Prir
0.0
−0.2
−0.4
−2
0
2
4
6
Principal Component 1
0.2
0.0

.