1.Build a neural network to detect and localize objects within images, such as identifying and drawing bounding boxes around cars, pedestrians, and traffic signs in street images.

Aim: To implement an autoencoder for reducing the dimensionality of high-dimensional data (e.g., features in a dataset) while preserving essential information. Evaluate the performance by comparing it to traditional techniques like PCA.

Steps:

1.Load the Dataset: You can use any street images dataset (like the COCO dataset).

2.Preprocess the Data: Resize images and normalize pixel values.

3.Load the Pre-trained YOLOv5 Model: Use a pre-trained YOLOv5 model that can detect objects like cars, pedestrians, and traffic signs.

4.Train the Model (optional): Fine-tune the model on your specific dataset.

5.Evaluate the Model: Use performance metrics to evaluate accuracy.

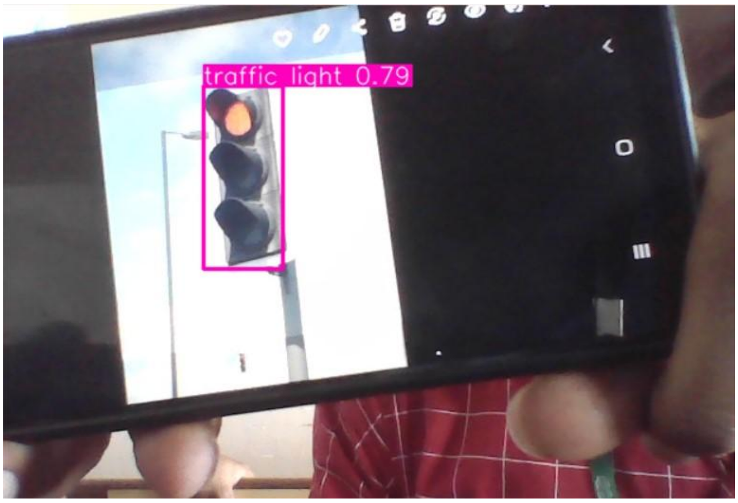
6.Visualize Results: Draw bounding boxes around detected objects in test images.

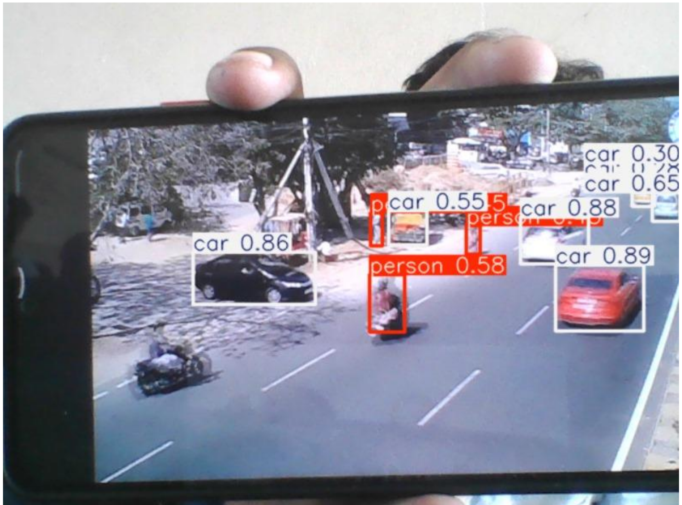
Code:

```
import torch
import cv2

# Load the YOLOv5 model from PyTorch Hub
model = torch.hub.load('ultralytics/yolov5', 'yolov5s', pretrained=True)

# Print the class names the model can detect
```

```python
print("Classes the model can detect:", model.names)

# Initialize the camera (0 is usually the default camera)
cap = cv2.VideoCapture(0)

# Set the camera resolution (optional)
cap.set(cv2.CAP_PROP_FRAME_WIDTH, 640)
cap.set(cv2.CAP_PROP_FRAME_HEIGHT, 480)

# Loop to continuously get frames from the camera and perform object detection
while True:
    # Capture frame-by-frame from the camera
    ret, frame = cap.read()

    # Check if the frame was captured correctly
    if not ret:
        print("Failed to capture image")
        break

    # Perform object detection on the frame
    results = model(frame)

    # Render the detection results on the frame
    # results.render() adds bounding boxes and labels directly on the image
    results.render()

    # Display the frame with bounding boxes in a window
    cv2.imshow('YOLOv5 Live Object Detection', frame)

    # Press 'q' to exit the loop and close the window
    if cv2.waitKey(1) & 0xFF == ord('q'):
        break

# Release the camera and close all OpenCV windows
cap.release()
cv2.destroyAllWindows()
```

OUTPUT:


traffic light 0.79


car 0.30
car 0.65
car 0.55
car 0.88
car 0.86
person 0.58
car 0.89


person 0.44
person 0.58
person 0.83
car 0.29
person 0.35
person 0.59
person 0.56
person 0.45
bicycle 0.59

Result:

The model outputs the detected objects along with their bounding box coordinates and confidence scores, while also visualizing the results on the input image.

2 . Implement an autoencoder for reducing the dimensionality of high-dimensional data (e.g., features in a dataset) while preserving essential information. Evaluate the performance by comparing it to traditional techniques like PCA.

Aim: The aim of this program is to implement an autoencoder for dimensionality reduction and compare its performance with traditional Principal Component Analysis (PCA) on the Iris dataset.

Steps:

1.Load the dataset: Load the Iris dataset using load_iris() from scikit-learn.
2.Split the data: Split the data into training and testing sets using train_test_split() from scikit-learn.
3.Define the autoencoder: Define the autoencoder architecture using Keras, consisting of an encoder and a decoder.
4.Compile the autoencoder: Compile the autoencoder with a mean squared error loss function and Adam optimizer.
5.Train the autoencoder: Train the autoencoder on the training data for 10 epochs with a batch size of 256.
6.Perform PCA: Perform PCA on the training data using PCA() from scikit-learn.
7.Calculate reconstruction error: Calculate the reconstruction error for both PCA and the autoencoder using mean_squared_error() from scikit-learn.
8.Visualize the results: Visualize the results using scatter plots to compare the performance of PCA and the autoencoder.

Code:

```
import numpy as np
```

```
from keras.layers import Input, Dense
from keras.models import Model
from sklearn.decomposition import PCA
from sklearn.metrics import mean_squared_error

# Load the dataset
from sklearn.datasets import load_iris
iris = load_iris()
X = iris.data
y = iris.target

# Split the data into training and testing sets
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)

# Define the input shape
input_shape = (X.shape[1],)

# Define the encoder
encoder_input = Input(shape=input_shape)
x = Dense(128, activation='relu')(encoder_input)
x = Dense(64, activation='relu')(x)
x = Dense(32, activation='relu')(x)
encoder_output = Dense(2, activation='relu')(x)

# Define the decoder
decoder_input = Input(shape=(2,))
x = Dense(32, activation='relu')(decoder_input)
x = Dense(64, activation='relu')(x)
x = Dense(128, activation='relu')(x)
decoder_output = Dense(X.shape[1], activation='sigmoid')(x)

# Define the autoencoder
autoencoder_input = Input(shape=input_shape)
encoder = Model(encoder_input, encoder_output)
decoder = Model(decoder_input, decoder_output)
autoencoder_output = decoder(encoder(autoencoder_input))
autoencoder = Model(autoencoder_input, autoencoder_output)
```

```
# Compile the autoencoder
autoencoder.compile(loss='mean_squared_error', optimizer='adam')

# Train the autoencoder
autoencoder.fit(X_train, X_train, epochs=10, batch_size=256,
validation_data=(X_test, X_test))

# Perform PCA on the data
pca = PCA(n_components=2)
X_pca = pca.fit_transform(X_train)

# Calculate the reconstruction error for PCA
reconstruction_error_pca = mean_squared_error(X_train,
pca.inverse_transform(X_pca))

# Calculate the reconstruction error for the autoencoder
reconstruction_error_autoencoder = mean_squared_error(X_train,
autoencoder.predict(X_train))

print("Reconstruction error (PCA):", reconstruction_error_pca)
print("Reconstruction error (Autoencoder):",
reconstruction_error_autoencoder)

# Visualize the results
import matplotlib.pyplot as plt
plt.scatter(X_pca[:, 0], X_pca[:, 1], c=y_train)
plt.title("PCA")
plt.xlabel("Principal Component 1")
plt.ylabel("Principal Component 2")
plt.show()

encoder_output = encoder.predict(X_train)
plt.scatter(encoder_output[:, 0], encoder_output[:, 1], c=y_train)
plt.title("Autoencoder")
plt.xlabel("Dimension 1")
plt.ylabel("Dimension 2")
plt.show()
```

Output:

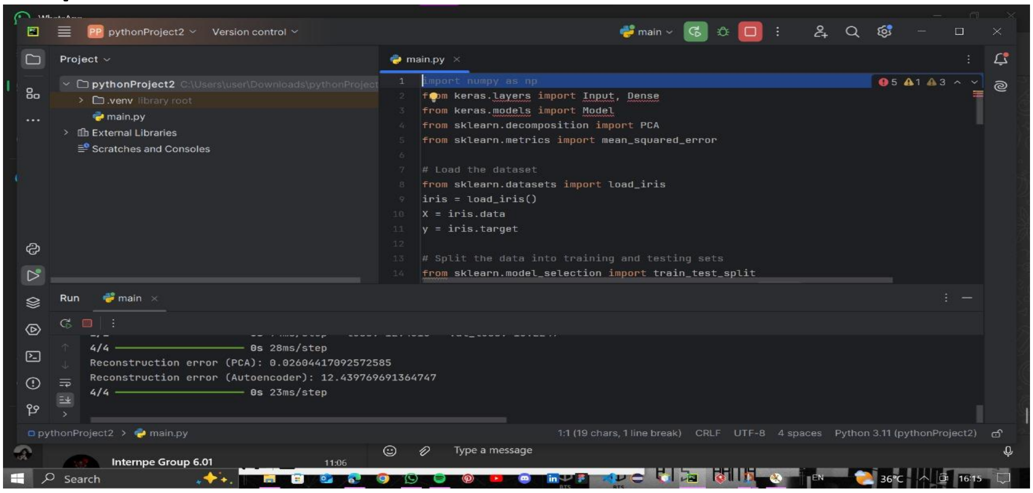
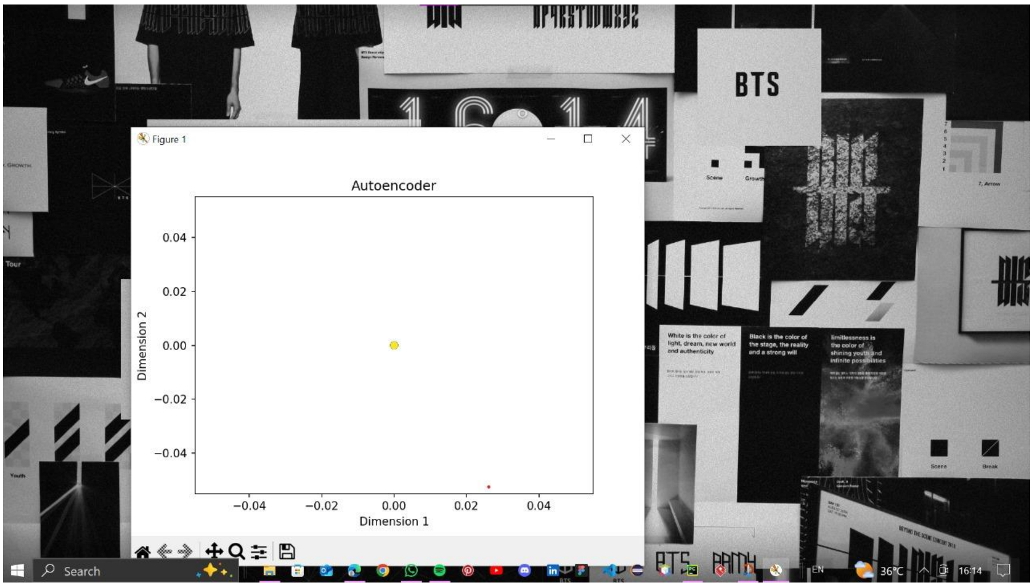

pythonProject2
Version control
main.py
import numpy as np
from keras.layers import Input, Dense
from keras.models import Model
from sklearn.decomposition import PCA
from sklearn.metrics import mean_squared_error
# Load the dataset
from sklearn.datasets import load_iris
iris = load_iris()
X = iris.data
y = iris.target
# Split the data into training and testing sets
from sklearn.model_selection import train_test_split
4/4 0s 28ms/step
Reconstruction error (PCA): 0.02604417092572585
Reconstruction error (Autoencoder): 12.439769691364747
4/4 0s 23ms/step


Figure 1
PCA
Principal Component 1
Principal Component 2



Result:

The program will output the reconstruction error for both PCA and the autoencoder, as well as visualize the results using scatter plots.