Finger Detection Using KNN

Introduction:

This project utilizes the K-Nearest Neighbors (KNN) machine learning algorithm to recognize the number
of fingers shown in an image. The images are preprocessed to extract numerical features, which are then used to train
the KNN classifier. Once trained, the model can predict the number of fingers in an unseen image.
This project demonstrates fundamental machine learning concepts, including data preprocessing, model training,
and classification.

Python Code:

```
from PIL import Image, ImageFilter
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier

X = []
y = []
for i in range(1500):  # 1500 different images
  for j in range(6):   # Each image has 6 variations (0 to 5 fingers)
              feature = preprocess_image(f'C:/Users/HP/Downloads/archive (2)/training_images
(copy)/{i}_{j}.png')
    X.append(feature)
    y.append(j)

X = np.array(X)
y = np.array(y)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)
knn = KNeighborsClassifier(n_neighbors=3)
knn.fit(X_train, y_train)
accuracy = knn.score(X_test, y_test)
print(f'KNN Accuracy: {accuracy}')

def detect_fingers(image):
    processed_image = preprocess_image(image)
    finger_count = knn.predict([processed_image])[0]
    return finger_count


fingers = detect_fingers('C:/Users/HP/Downloads/archive (2)/hand_check.png')
print(f'Fingers detected: {fingers}')
```

Detailed Explanation:

1. **Import Required Libraries**:
   - `PIL` is used for image loading and preprocessing.
   - `numpy` helps with numerical operations.
   - `train_test_split` is used to divide the dataset into training and testing sets.
   - `KNeighborsClassifier` is the machine learning model used.

2. **Data Loading and Preprocessing**:
   - The code loops through 1500 images, each having 6 variations labeled 0-5 (number of fingers).
   - Each image is processed using `preprocess_image()` and stored in the dataset.

3. **Converting Data to NumPy Arrays**:
   - The processed images and their labels are converted into NumPy arrays for efficient handling.

4. **Splitting Data into Training and Testing Sets**:
   - `train_test_split(X, y, test_size=0.2)` splits 80% of the data for training and 20% for testing.

5. **Training the KNN Model**:
   - `KNeighborsClassifier(n_neighbors=3)` initializes a KNN classifier with `k=3` (3 nearest neighbors).
   - `knn.fit(X_train, y_train)` trains the model on the training dataset.

6. **Evaluating Model Accuracy**:
   - `knn.score(X_test, y_test)` calculates accuracy based on test images.

7. **Finger Detection Function (`detect_fingers`)**:
   - Takes an image as input.
   - Preprocesses it using `preprocess_image()`.
   - Uses the trained KNN model to predict the number of fingers.

8. **Testing on a New Image**:
   - The model is tested on `hand_check.png`, and the detected number of fingers is printed.

Conclusion:

This project demonstrates how machine learning can be used for image classification. The KNN algorithm
is simple yet effective for finger detection tasks. With additional training data and advanced feature extraction techniques,
the accuracy can be improved. This project serves as a foundation for more complex image recognition applications.