Sign Language Recognition

MACHINE LEARNING MINI-PROJECT - ASSIGNMENT 8

Members -

1. Riya Shallya (UCE2021461)
2. Eesha Satvalekar (UCE2021463)
3. Mrunal Vibhute (UCE2021470)
4. Aditi Wagh (UCE2021472)

Problem Statement

Specially abled people use hand signs and gestures to communicate. Normal people face difficulty in understanding their language. Hence there is a need for a software application which recognizes the different signs, gestures and conveys the information to normal people. The machine learning model that recognises sign language and displays the output in text form is developed.Thus it bridges the gap between physically challenged people and normal people.

Introduction

The model uses machine learning and computer vision to recognize hand gestures from webcam footage using the K-Nearest Neighbors (KNN) algorithm.

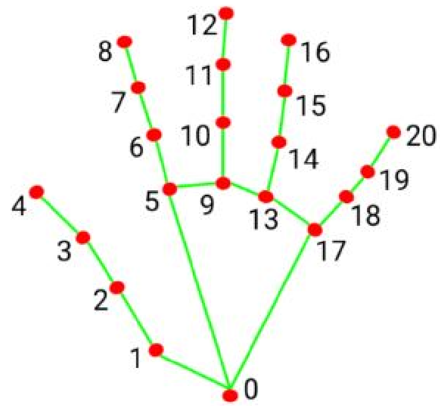
The script begins by importing the necessary libraries: OpenCV and Mediapipe for capturing hand coordinates, pandas and numpy for processing data, and sklearn for machine learning. It then loads a hand gesture dataset from a CSV file and splits it into training and testing sets.

Next, the script standardizes the dataset and trains a KNN classifier using the training data. It then uses the classifier to predict the classes of the test data and prints the classification report and accuracy score.

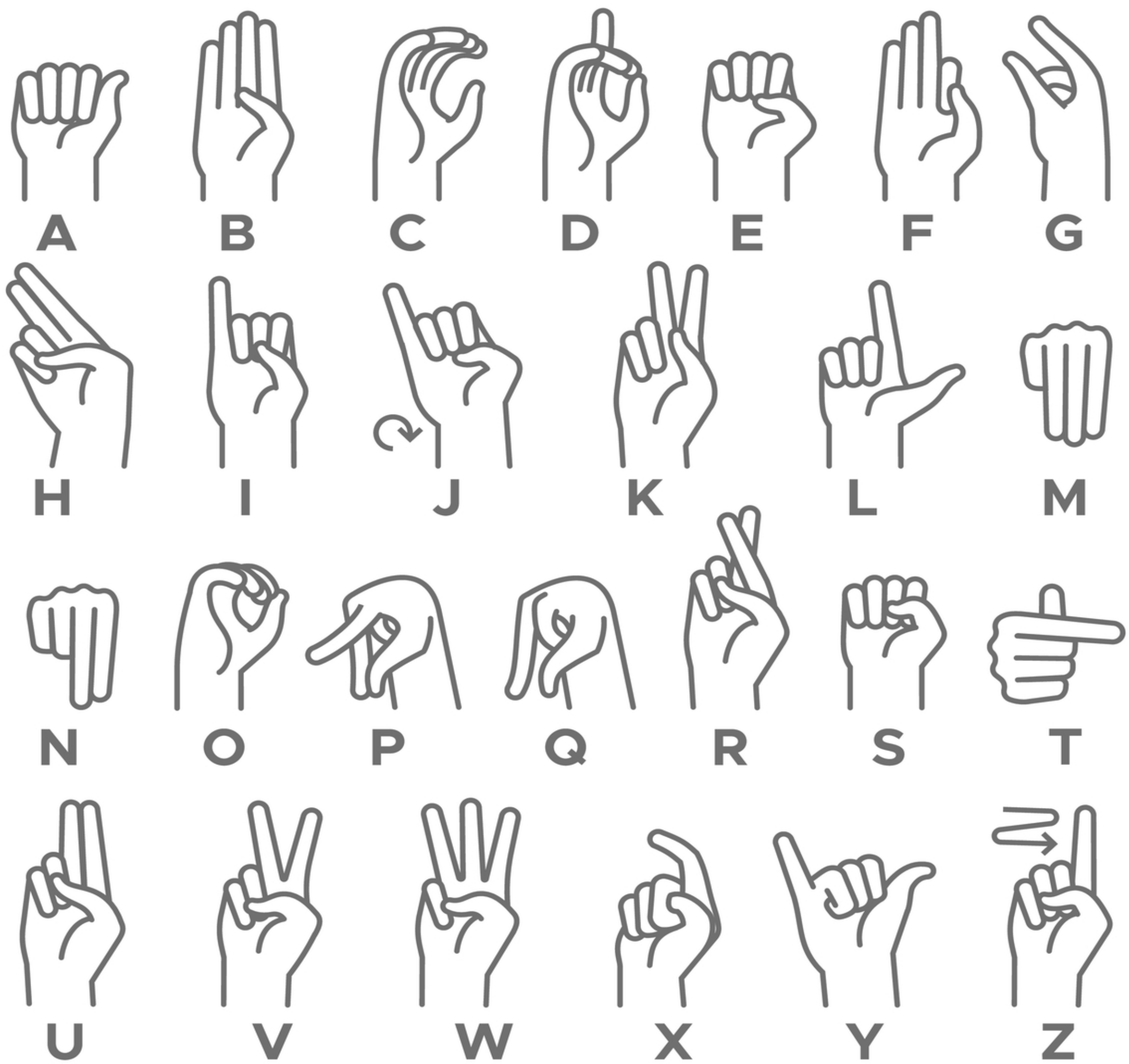
The error rate for k values from 1 to 40 is calculated and plotted on a graph to help choose the optimal k value.

After this, the script initializes the Mediapipe hand detection pipeline and captures webcam footage. It processes each frame using the pipeline and extracts the hand landmarks. It then flattens and normalizes the landmark coordinates and feeds them into the KNN classifier to predict the corresponding hand gesture.

Finally, the predicted gesture is displayed on the webcam footage and continues until the user presses the 'Esc' key to exit.



0. WRIST
1. THUMB_CMC
2. THUMB_MCP
3. THUMB_IP
4. THUMB_TIP
5. INDEX_FINGER_MCP
6. INDEX_FINGER_PIP
7. INDEX_FINGER_DIP
8. INDEX_FINGER_TIP
9. MIDDLE_FINGER_MCP
10. MIDDLE_FINGER_PIP
11. MIDDLE_FINGER_DIP
12. MIDDLE_FINGER_TIP
13. RING_FINGER_MCP
14. RING_FINGER_PIP
15. RING_FINGER_DIP
16. RING_FINGER_TIP
17. PINKY_MCP
18. PINKY_PIP
19. PINKY_DIP
20. PINKY_TIP



Data Set Information (link, few data samples etc)

https://raw.githubusercontent.com/MinorvaFalk/KNN_Alphabet/main/Dataset/hand_dataset_1000_24.csv

Dataset is generated from image to landmark in csv.

Data Samples :

'C'

'I'

'W'

'R'

Code & Output

I. Code

```
# For capturing hand coordinates
import cv2
import mediapipe as mp

# For processing data
import pandas as pd
import numpy as np
dataset =
pd.read_csv('https://raw.githubusercontent.com/MinorvaFalk/KNN_Alphabet/main/Dataset/hand_dataset_1000_24.csv')

# Show dataset first five data
dataset.head()

# Defining X and Y from dataset for training and testing
X = dataset.iloc[:, 1:].values
Y = dataset.iloc[:, 0].values
from sklearn.model_selection import train_test_split

# We will take 33% from 1000 for our test data.
X_train, X_test, y_train, y_test = train_test_split(X, Y,
test_size=0.33)
```

```
# Standardize dataset

from sklearn.preprocessing import StandardScaler
scaler = StandardScaler().fit(X_train)

X_train = scaler.transform(X_train)
X_test = scaler.transform(X_test)

from sklearn.neighbors import KNeighborsClassifier
classifier = KNeighborsClassifier(n_neighbors=3)
classifier.fit(X_train, y_train)

y_pred = classifier.predict(X_test)
from sklearn.metrics import classification_report,
accuracy_score
print(classification_report(y_test, y_pred))
print('The accuracy of model is:')
print(accuracy_score(y_test, y_pred)*100)

error = []

# Calculating error for K values between 1 and 40
for i in range(1, 40):
   knn = KNeighborsClassifier(n_neighbors=i)
   knn.fit(X_train, y_train)
   pred_i = knn.predict(X_test)
```

```
   error.append(np.mean(pred_i != y_test))
import matplotlib.pyplot as plt

plt.figure(figsize=(12, 6))
plt.plot(range(1, 40), error, color='red', linestyle='dashed',
marker='o',
        markerfacecolor='blue', markersize=10)
plt.title('Error Rate K Value')
plt.xlabel('K Value')
plt.ylabel('Error')
plt.show()

# Initialize mediapipe hand

mp_drawing = mp.solutions.drawing_utils
mp_hands = mp.solutions.hands
# Initialize mediapipe hand capture webcam

cap = cv2.VideoCapture(0)
with mp_hands.Hands(
   max_num_hands = 1,
   min_detection_confidence=0.5,
   min_tracking_confidence=0.5) as hands:
   while cap.isOpened():
       success, image = cap.read()
```

```
        if not success:
            print("Ignoring empty camera frame.")
            # If loading a video, use 'break' instead of
'continue'.
            continue

        # Flip the image horizontally for a later selfie-view
display, and convert
        # the BGR image to RGB.
        image = cv2.cvtColor(cv2.flip(image, 1),
cv2.COLOR_BGR2RGB)

        # To improve performance, optionally mark the image as
not writeable to
        # pass by reference.
        image.flags.writeable = False
        results = hands.process(image)

        # Draw the hand annotations on the image.
        image.flags.writeable = True
        image = cv2.cvtColor(image, cv2.COLOR_RGB2BGR)
        if results.multi_hand_landmarks:
            for hand_landmarks in results.multi_hand_landmarks:
                coords = hand_landmarks.landmark
                mp_drawing.draw_landmarks(image,
hand_landmarks, mp_hands.HAND_CONNECTIONS)
```

```
                coords = list(np.array([[landmark.x,
landmark.y] for landmark in coords]).flatten())
                coords = scaler.transform([coords])

                # Alternative for dataset using z coordinates.
                # Z coordinates is not recommended, since you
need to adjust your distance from camera.
#                 coords = list(np.array([[landmark.x,
landmark.y, landmark.z] for landmark in coords]).flatten())

                predicted = classifier.predict(coords)

            # Get status box
            cv2.rectangle(image, (0,0), (100, 60), (245, 90,
16), -1)

            # Display Class
            cv2.putText(image, 'CLASS'
                        , (20,15), cv2.FONT_HERSHEY_SIMPLEX,
0.5, (0, 0, 0), 1, cv2.LINE_AA)
            cv2.putText(image, str(predicted[0])
                        , (20,45), cv2.FONT_HERSHEY_SIMPLEX, 1,
(255, 255, 255), 2, cv2.LINE_AA)

        cv2.imshow('MediaPipe Hands', image)

        # Press esc to close webcam
```

```
        if cv2.waitKey(5) & 0xFF == 27:
            break
cap.release()
cv2.destroyAllWindows()
```

II. Output -

The output of code is as follows:

```
              precision    recall  f1-score   support

           A       0.93      1.00      0.96       315
           B       0.93      0.99      0.96       336
           C       0.95      0.99      0.97       352
           D       0.98      0.96      0.97       330
           E       0.94      0.98      0.96       346
           F       0.99      0.98      0.98       333
           G       0.98      1.00      0.99       331
           H       0.99      0.99      0.99       299
           I       0.98      0.97      0.98       357
           K       0.95      0.97      0.96       334
           L       0.98      0.99      0.98       322
           M       0.85      0.83      0.84       333
           N       0.84      0.84      0.84       304
           O       0.95      0.96      0.95       318
           P       0.97      0.95      0.96       358
           Q       0.93      0.93      0.93       307
           R       0.91      0.90      0.91       354
           S       0.93      0.87      0.90       331
           T       0.96      0.97      0.96       343
           U       0.82      0.90      0.86       316
           V       0.92      0.84      0.88       338
           W       0.99      0.94      0.96       315
           X       0.93      0.87      0.90       327
           Y       0.98      0.95      0.97       321

    accuracy                           0.94      7920
   macro avg       0.94      0.94      0.94      7920
weighted avg       0.94      0.94      0.94      7920

0.9401515151515152
```

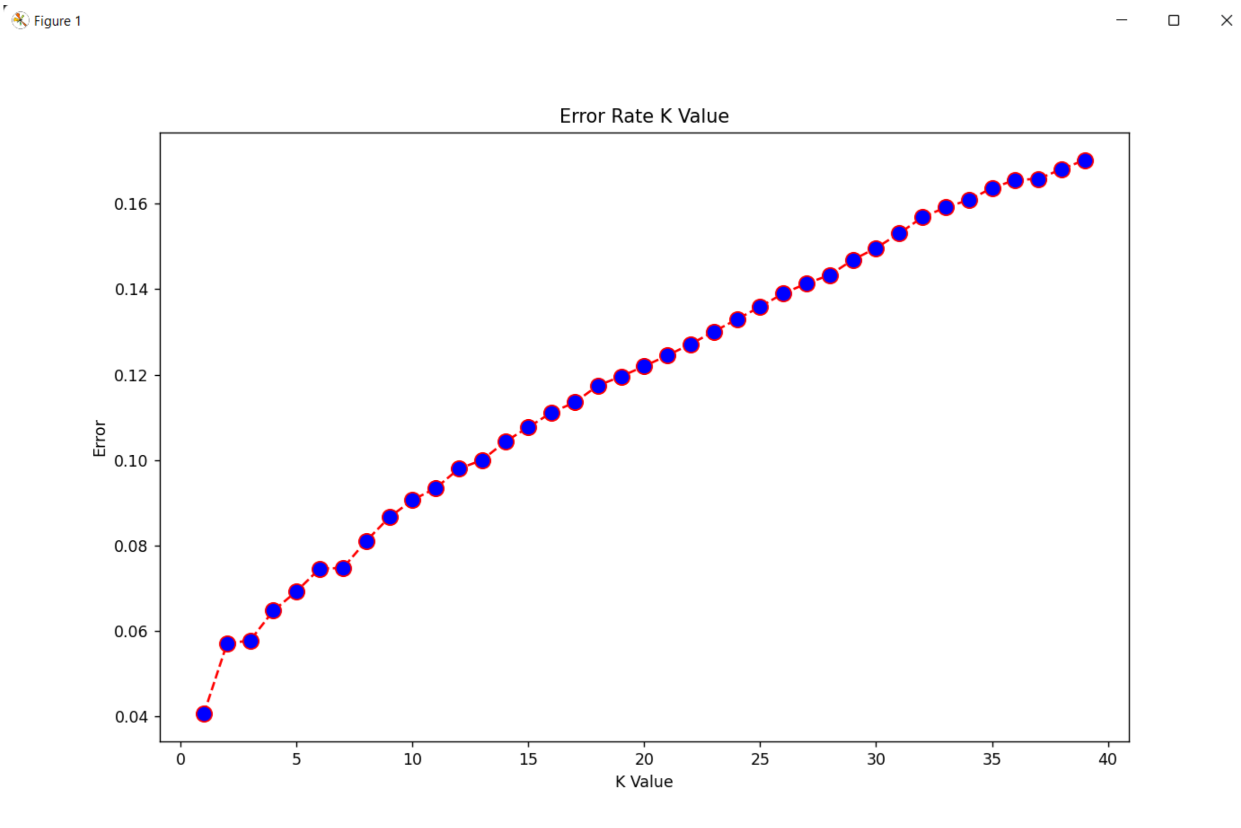
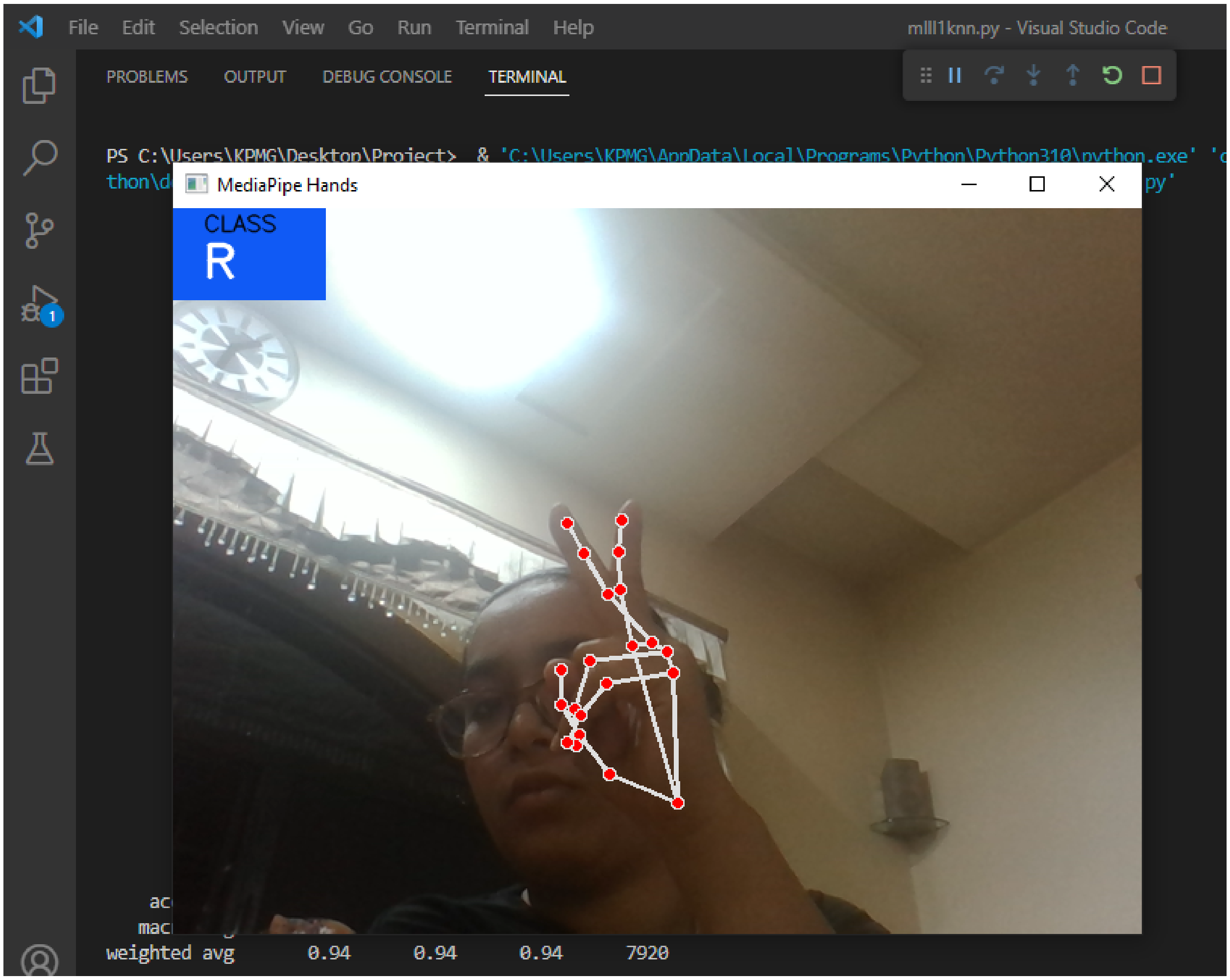
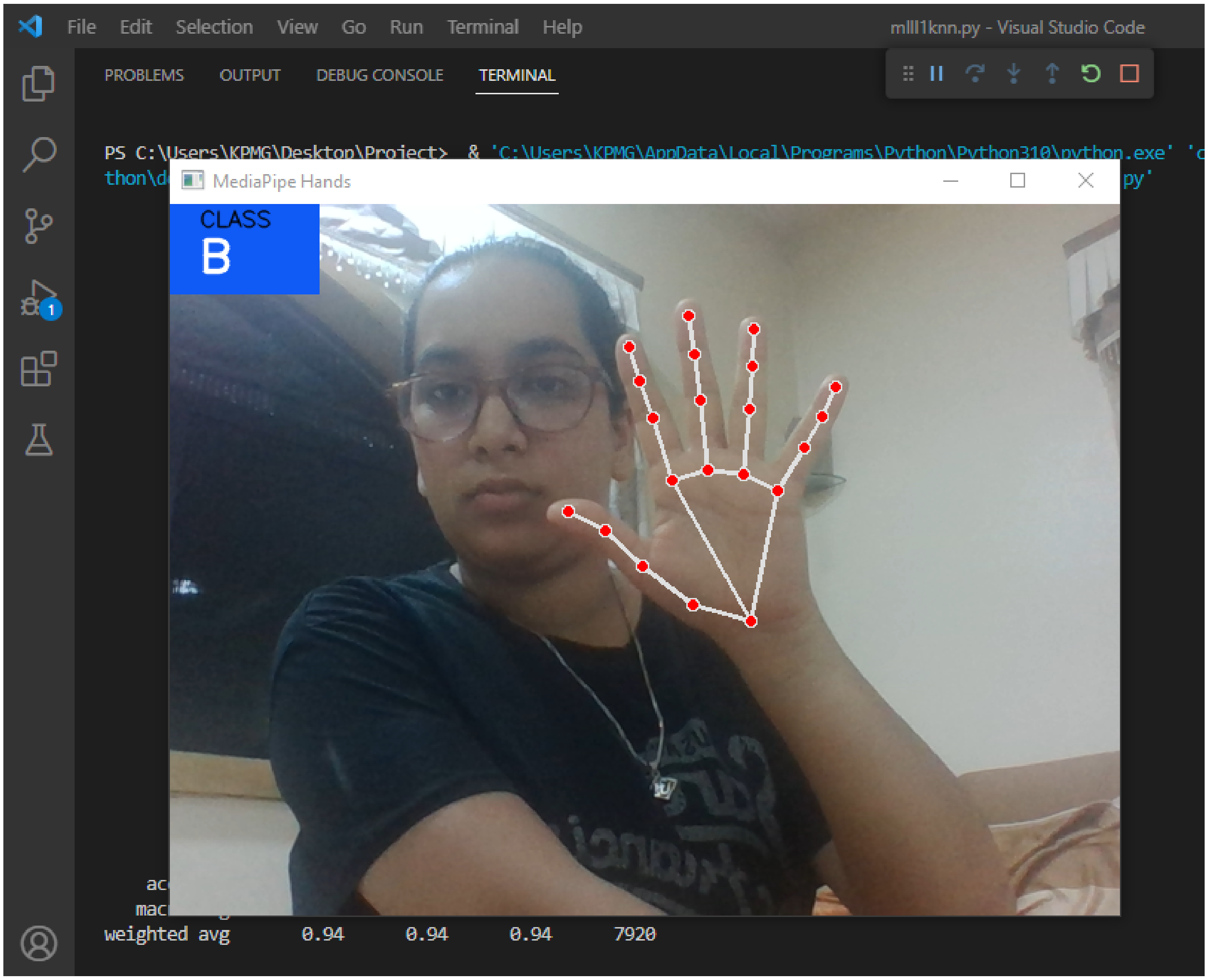
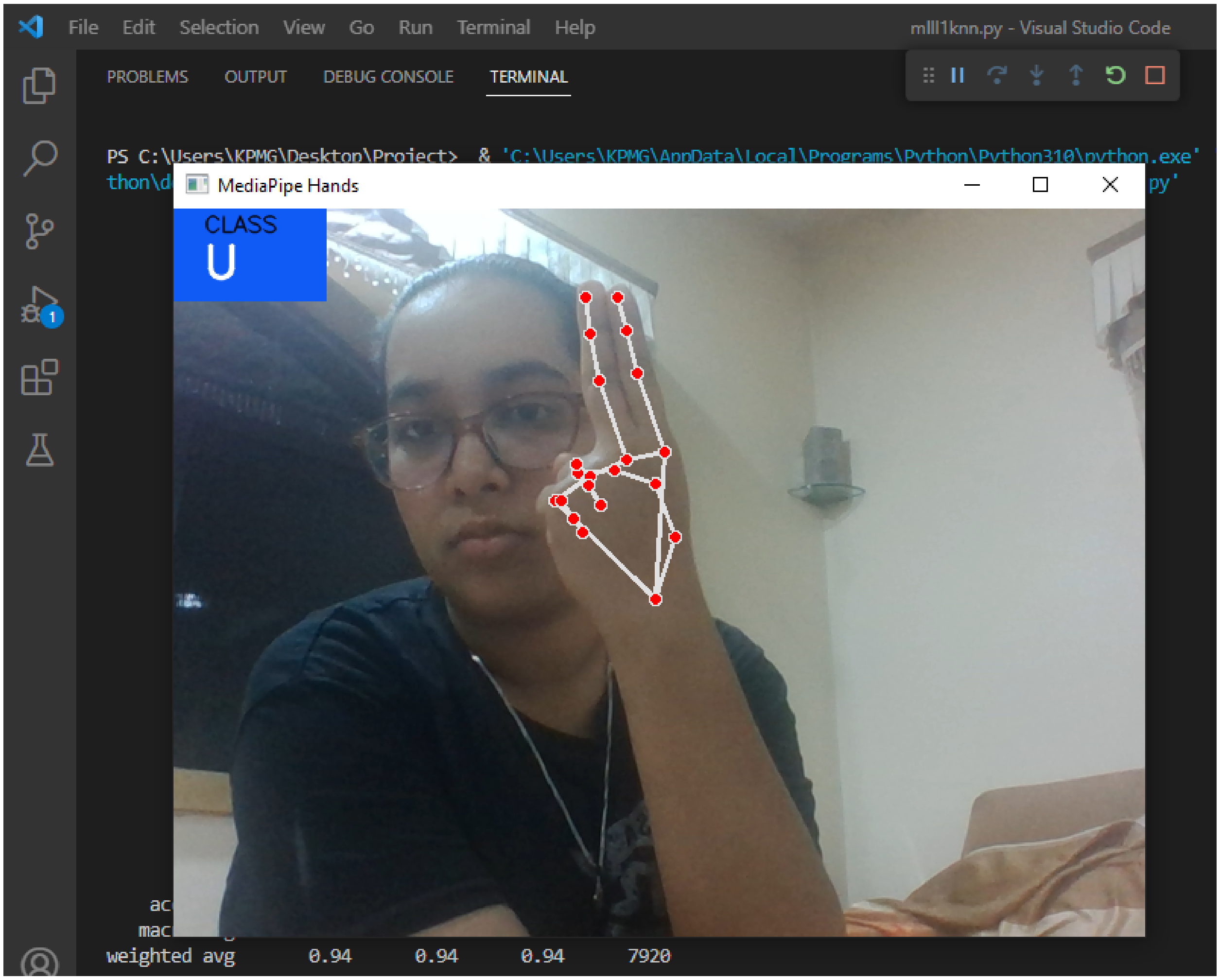
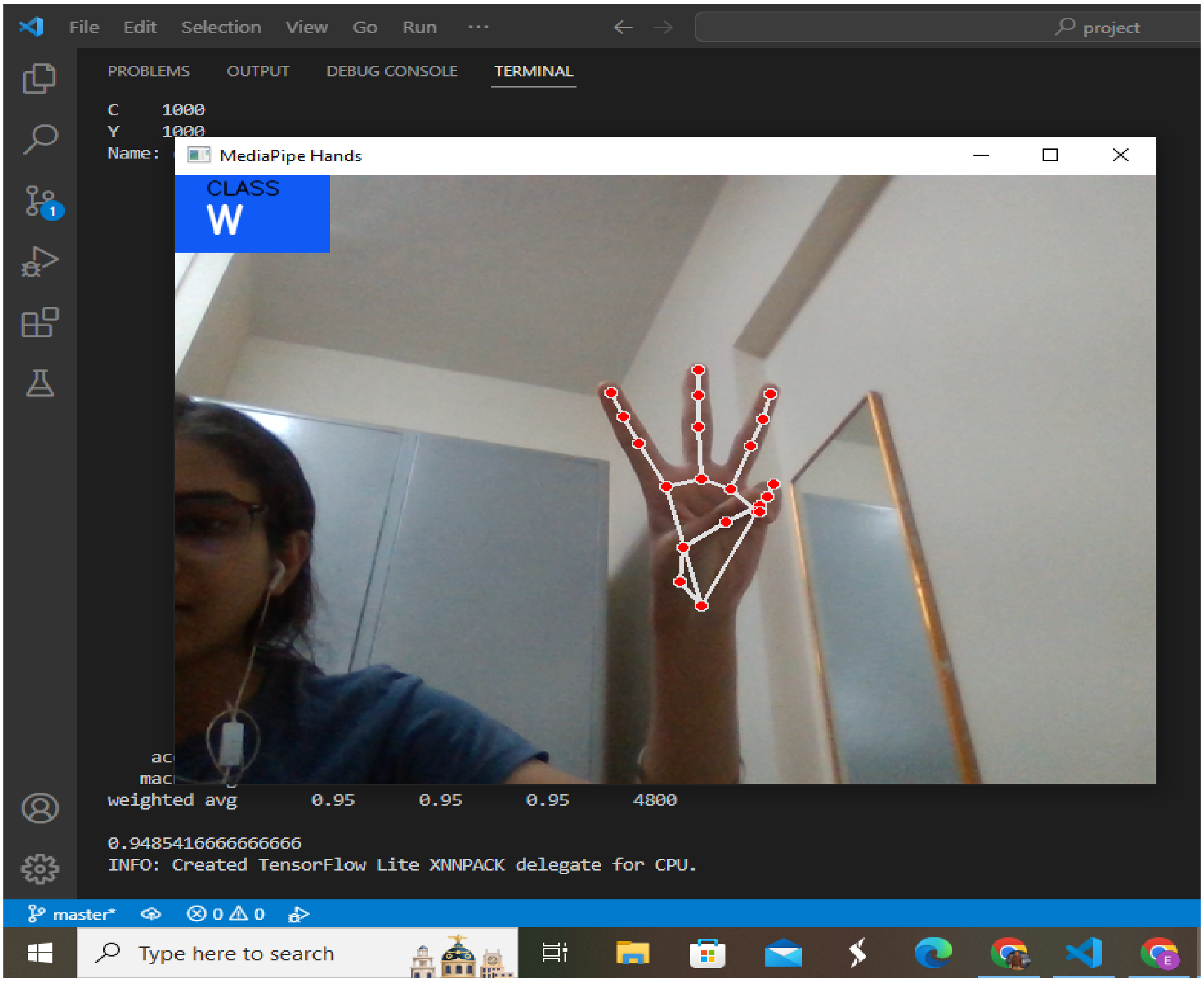


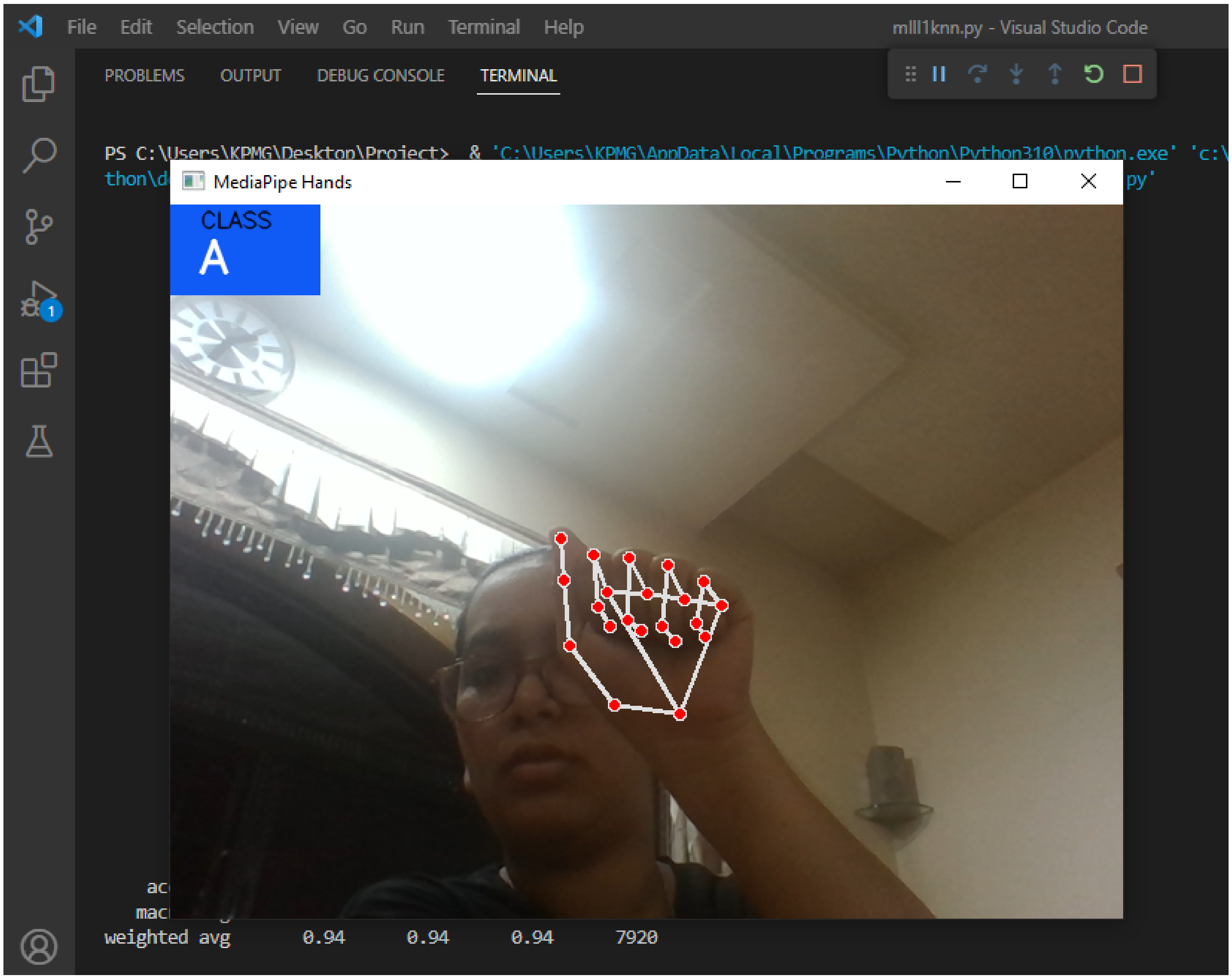
Figure 1
Error Rate K Value
Error
0.16
0.14
0.12
0.10
0.08
0.06
0.04
0
5
10
15
20
25
30
35
40
K Value


File Edit Selection View Go Run Terminal Help
mlll1knn.py - Visual Studio Code
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
PS C:\Users\KPMG\Desktop\Project> & 'C:\Users\KPMG\AppData\Local\Programs\Python\Python310\python.exe'
MediaPipe Hands
CLASS
R
weighted avg 0.94 0.94 0.94 7920


File Edit Selection View Go Run Terminal Help
mlll1knn.py - Visual Studio Code
PROBLEMS
OUTPUT
DEBUG CONSOLE
TERMINAL
PS C:\Users\KPMG\Desktop\Project> & 'C:\Users\KPMG\AppData\Local\Programs\Python\Python310\python.exe'
MediaPipe Hands
CLASS
B
weighted avg 0.94 0.94 0.94 7920


File Edit Selection View Go Run Terminal Help
mlll1knn.py - Visual Studio Code
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
PS C:\Users\KPMG\Desktop\Project> & 'C:\Users\KPMG\AppData\Local\Programs\Python\Python310\python.exe'
MediaPipe Hands
CLASS
U
weighted avg 0.94 0.94 0.94 7920


File Edit Selection View Go Run ...
project
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
C 1000
Y 1000
Name:
MediaPipe Hands
CLASS
W
weighted avg 0.95 0.95 0.95 4800
0.9485416666666666
INFO: Created TensorFlow Lite XNNPACK delegate for CPU.
master*
Type here to search


File Edit Selection View Go Run Terminal Help
mlll1knn.py - Visual Studio Code
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
PS C:\Users\KPMG\Desktop\Project> & 'C:\Users\KPMG\AppData\Local\Programs\Python\Python310\python.exe'
MediaPipe Hands
CLASS
A
weighted avg 0.94 0.94 0.94 7920

Conclusion

1. KNN (K-Nearest Neighbors) and CNN (Convolutional Neural Networks) are both popular algorithms used in image processing, but they have different strengths and weaknesses.
2. KNN is a simple algorithm that works by finding the k-nearest neighbors of a new data point based on its features. In the context of image processing, the features could be pixel values or image descriptors like histograms of oriented gradients (HOG).
3. KNN can be a good choice for image classification tasks where the number of classes is small, and the feature space is relatively low-dimensional.
4. On the other hand, CNNs are a powerful class of neural networks that are specifically designed for image processing tasks. They work by learning a hierarchy of features from raw pixels, and they are capable of capturing complex patterns and structures in images.
5. CNNs are particularly effective for tasks like object detection, segmentation, and image recognition, where the number of classes is large, and the feature space is high-dimensional.
6. Whether KNN or CNN is better for image processing depends on the specific task and the characteristics of the data. In general, KNN can be better than CNN for simple image classification tasks with low-dimensional feature spaces and small numbers of classes. However, CNNs are generally more powerful and versatile for most image processing tasks.

References

1. https://www.geeksforgeeks.org/python-opencv-cv2-imwrite-method/
2. https://developers.google.com/mediapipe/solutions/vision/hand_landmarker
3. https://scikit-learn.org/stable/
4. https://raw.githubusercontent.com/MinorvaFalk/KNN_Alphabet/main/Dataset/hand_dataset_1000_24.csv
5. http://www.jcomputers.us/vol14/jcp1401-06.pdf