Final Project Part B: Hand Gesture Recognition in Live Video

CEG4133[A] – Computer Methods in Picture Processing and Analysis

Fall 2022

School of Electrical Engineering and Computer Science

University of Ottawa

Course Coordinator: WonSook Lee

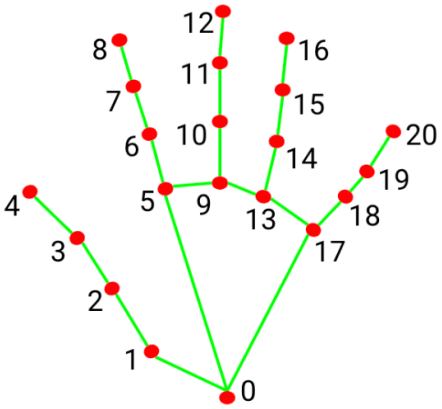
Teaching Assistant: Cristopher McIntyre Garciahao

Kelvin Phan 300057061

Submission Date: December 2nd, 2022

For part B of the final project of the course, the objective is to both detect my own choice of hand gestures in live video. I have chosen to recognize the ASL alphabet. This will be done using three libraries, OpenCV to deal with processing the video, MediaPipe to deal with processing hand detection, and finally Scikit Learn as a machine learning tool to train a model to recognize the ASL alphabet.

The basic concept behind MediaPipe is that it can recognize hands in a video and for each hand it will assign 21 landmarks to the points illustrated below in Figure 1. These landmarks are assigned a normalized x, y, and z point between 0 and 1.



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

Figure 1. Hand Landmarks according to MediaPipe documentation

With the use of the MediaPipe library, we can recognize landmark coordinates and their normalized position for each of the signs in the ASL alphabet.

To train a model to classify live video signs into its corresponding letter, we first need a dataset to train our data. The dataset I have chosen to use comes from https://github.com/MinorvaFalk/KNN_Alphabet/tree/main/Dataset. This dataset contains 1000 sets of x and y landmark coordinates for each of the 21 landmarks taken from training images for each respective letter of the alphabet. In total, there are 24000 labelled data points in our dataset. The dataset was generated using the static_image_mode of MediaPipe to get the coordinates from the thousands of images provided in https://www.kaggle.com/datasets/grassknoted/asl-alphabet.

The first objective is to create and train a model to classify our hand using Scikit Learn. In this section, we will be referring to the lines of code in Figure 2. We begin by defining the path of our dataset on line 19. Next, we split our dataset into X and y. X will contain the coordinates for each of the 21 landmarks while y will contain the corresponding letter that it represents. This splitting step in done in lines 24-25. After splitting up our data we move on to normalization. With the help of normalization, we can transform of landmark's coordinates into a finite range so that detection can be made easier no matter where on the screen the detected hand will be. This step is done in lines 28-30. After our data has been processed, we now must choose a model to train. On line 34-53 we define a helper function named test_model() to test various models to see which one will yield the highest accuracy. In test_model() we using cross validation to split our data into 10 folds and train our model on 9 of the folds while reserving one

for testing. We then tested Gaussian Naïve Bayes, K-nearest neighbours, Logistic Regression, Decision Tree and Support Vector models and K-nearest neighbours produced the highest accuracy score of 95.2% as seen in Figure 3. Moving forward with the most accurate model, on lines 57-60 we define our classifier as the K-nearest neighbour model and train it using Scikit-learn's built-in fit() method using our dataset. Finally, with our model training using the provided data, we create a joblib file containing our model to be used in the live gesture detection.

```
18  ## Read dataset using pandas
19  dataset = pd.read_csv(datasetPath)
20
21  ## Split dataset into X and y
22  ## @param X: the landmark coodinates data
23  ## @param y: the corresponding letter
24  X = dataset.drop('class', axis = 1)
25  y = dataset['class']
26
27  ## Create normalizer and normalize coordinate data
28  normalizer = Normalizer().fit(X)
29
30  X = normalizer.transform(X)
31
32  '''
33  ## Code used to test various models accuracy score on data
34  cross_Validate = KFold(n_splits = 10, random_state=7, shuffle = True)
35
36  def test_model(modelName, model):
37
38    print(f'Testing Model {modelName} ...')
39    scoring = ['accuracy']
40    scores = cross_validate(model, X, y, scoring = scoring, cv=cross_Validate, n_jobs = -1)
41
42    accuracy = np.mean(scores['test_accuracy'])
43    print('Mean Accuracy: %.3f\n' % (accuracy))
44
45  test_model("GNB", GaussianNB())
46
47  test_model("KNN", KNeighborsClassifier())
48
49  test_model("LR", LogisticRegression(max_iter = 1000))
50
51  test_model("DT", DecisionTreeClassifier())
52
53  test_model("SVC", SVC())
54  '''
55
56  ## Classifier chosen in KNeighbors
57  classifier = KNeighborsClassifier()
58
59  ## Train our classifier using dataset
60  classifier.fit(X, y)
61
62  ## Dump classifier to be used in handDetection.py
63  dump(classifier, "model.joblib")
```

Figure 2. Model training process in model.py

```
Testing Model GNB ...
Mean Accuracy: 0.464

Testing Model KNN ...
Mean Accuracy: 0.952

Testing Model LR ...
Mean Accuracy: 0.870

Testing Model DT ...
Mean Accuracy: 0.897

Testing Model SVC ...
Mean Accuracy: 0.949
```

Figure 3. Model testing results

After the machine learning part is complete, we move on to live gesture detection using OpenCV and the model that we have created. To begin we go through the normal steps of opening our capture, defining a MediaPipe hands object and setting the maximum number of hands to 1 to improve detection performance, all done on lines 10-14 of Figure 4. In part b of this project, we are also defining an object of mp_drawing to help visualize the landmark positions as they are detected on line 15 of Figure 4. Finally, we also load up the model we trained in the previous section and defining a normalizer object as well on line 18-19.

```
 9    ## Open capture with video path
10    capture = cv2.VideoCapture(0)
11
12    ## Initialize mediapipe hand detection function
13    mpHands = mp.solutions.hands
14    hands = mpHands.Hands(max_num_hands=1)
15    mp_drawing = mp.solutions.drawing_utils
16
17    ## Load trained model and initialize a normalizer
18    model = load("model.joblib")
19    normalizer = Normalizer()
```

Figure 4. Preliminary steps of handDetection.py

Once the preliminary steps are done, we move on to the process of detecting the gestures from the live video feed. Most of the steps are similar to Project Part A where we have the helper method createBoundingBox() and the normal steps of detecting the hand and drawing a green box around it. The process differs on lines 91-102 of Figure 5. The code for this section will refer to Figure 5. After defining our bounding box and drawing it on our frame on lines 85-88, we draw the connections between the detected landmarks on line 91. Next, we retrieve our detected landmark coordinates and create a list of only the x and y coordinates of each landmark on lines 94-95. Once we have this list, we use the normalize defined earlier to normalize our detected coordinates on line 96. Using the normalized coordinate, we predict which gesture is being detected using the model trained in model.py using Scikit-learn's built-in predict() method on line 99. The variable predicted_letter now contains the letter that was predicted using the model that we trained and using this variable we write the predicted letter on top of our bounding box. This putText() method is done on line 102. Finally, we show our frame with the results until we exit.

```
## While capture is open
while(capture.isOpened()):

    ## Read the frame from capture
    read, frame = capture.read()

    frame = cv2.flip(frame,1)

    ## If frame was properly read
    if read == True:

        ## Convert frame to RGB for proper mediapipe detection
        rgbFrame = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)

        ## Process each frame to get hand landmarks
        results = hands.process(rgbFrame)

        ## If results exists
        if results.multi_hand_landmarks:

            ## For each hand detected
            for handLms in results.multi_hand_landmarks:

                ## Call upon createBoudningBox() method to get bounding box coordinates
                boudingBox = createBoundingBox(frame, handLms)

                ## Draw a rectangle around each processed bounding box
                cv2.rectangle(frame, (boudingBox[0], boudingBox[1]), (boudingBox[2], boudingBox[3]), (0, 255, 0), 2)

                ## Draw the connections between landmarks for better visualization
                mp_drawing.draw_landmarks(frame, handLms, mpHands.HAND_CONNECTIONS)

                ## Define coords as the landmark's x and y coordinates and normalize them
                coords = handLms.landmark
                coords = list(np.array([[landmark.x, landmark.y] for landmark in coords]).flatten())
                coords = normalizer.transform([coords])

                ## Predict which letter is being gestured using the trained model
                predicted_letter = model.predict(coords)

                # Write above the bouding box the predicted letter
                cv2.putText(frame, str(predicted_letter[0]),(boudingBox[0], boudingBox[1]), cv2.FONT_HERSHEY_SIMPLEX, 1, (255, 255, 255), 2, cv2.LINE_AA)

        cv2.imshow("Frame", frame)

        if cv2.waitKey(1) & 0xFF == ord('q'):
            break

    else:
        break

capture.release()
cv2.destroyAllWindows()
```

Figure 5. Hand gesture recognition code in handDetection.py

In Figures 6-8 examples of handDetection.py can be seen where we make the sign for A, B and C in ASL and the program will write the predicted letter above the green box.

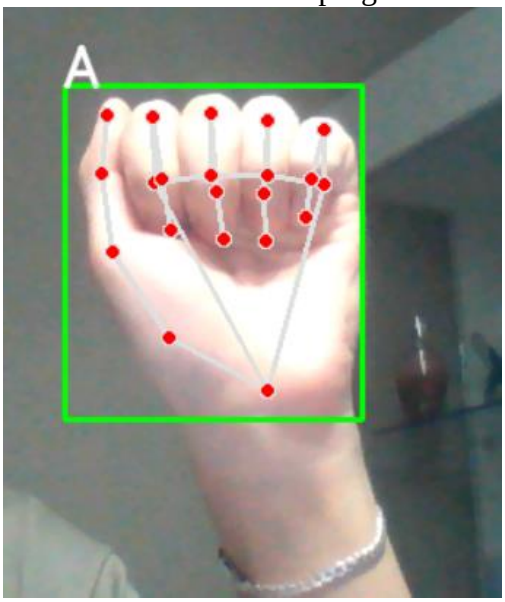


Figure 6. Detected gesture for A in ASL

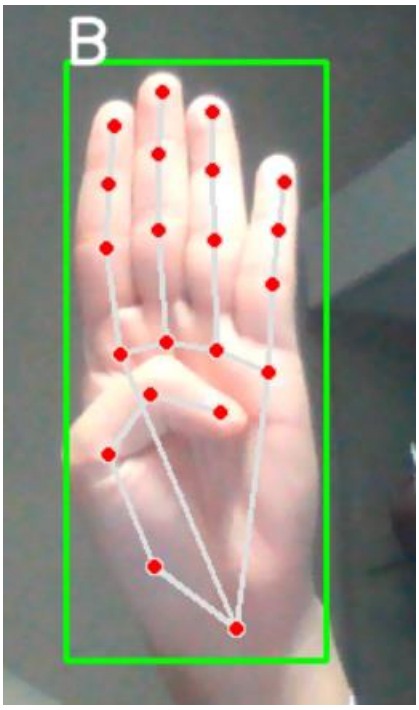


Figure 7. Detected gesture for B in ASL

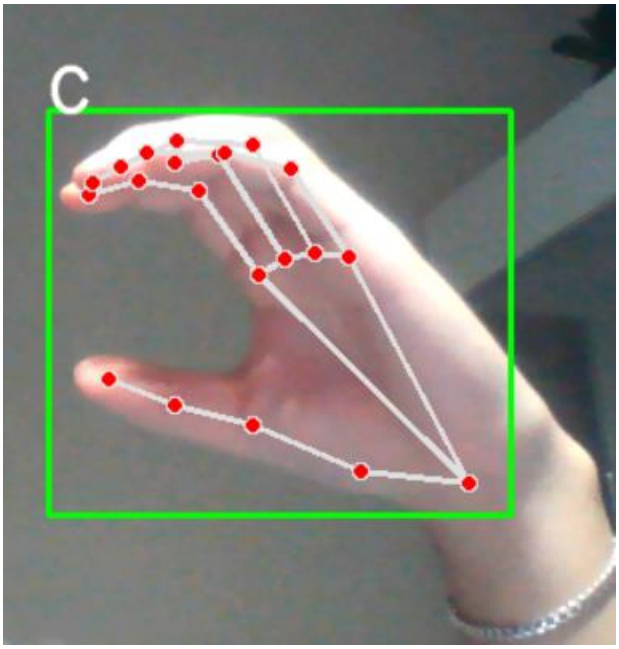


Figure 8. Detected gesture for C in ASL