COL780-Assignment 3 Report
Eshan Jain
2020CS50424

1. Tools and Technology Used:

The tools used in this assignment are opencv library for basic image processing, sklearn library for svm and metrics, mediapipe for detecting hands in images and finding bounding boxes around them, pygame for playing music for real world app, time library for recording the running time of my code, pickle library for saving my trained model, and numpy for basic mathematical operations.

2. Brief Algorithm of Your Code:

```
# Initialize MediaPipe Hands
mpHands = mp.solutions.hands
hands = mpHands.Hands(static_image_mode=True,
                      max_num_hands=2,
                      min_detection_confidence=0.5,
                      min_tracking_confidence=0.5)
```

I first used mediapipe to detect hands and find the bounding boxes (using the code given in the ROI_coordinates.py script) for the hands in each image in the dataset.Then I extract the hog features from the focused region of interest (the bounding box enclosing the hand). This is done by first resizing the image to a fixed size (common for all images). This is followed by calculating the gradients using the Sobel Operator. Then the image is divided into blocks and 9 bin histograms are computed for each block. The block histogram values are then normalized to account for contrast changes and lighting variations in images. The code for this part is well commented to ensure clarity and understanding.

```python
# Process each image to generate bounding boxes and extract HOG features
for filename in os.listdir(dataset_folder):
    if filename.endswith(".jpg"):
        # Load the image
        img_path = os.path.join(dataset_folder, filename)
        img = cv2.imread(img_path)
        img = cv2.flip(img, 1)
        imgRGB = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)

        # Detect hands and extract landmarks
        results = hands.process(imgRGB)

        # check if hands are detected
        if results.multi_hand_landmarks:
            # Extract bounding boxes from hand landmarks
            bounding_boxes = extract_bounding_boxes(img, results.multi_hand_landmarks)
            # Extract HOG features from within bounding boxes
            for bbox in bounding_boxes:
                hog_features = extract_hog_from_roi(img, bbox)
                if hog_features is not None:
                    X.append(hog_features)
                    if "closed" in dataset_folder:
                        y.append(0)
                    else:
                        y.append(1)
```

These extracted hog features are used for training the svm on them. I did this by flattening the hog features for each image and then making a list of these hog features for all training images. For each training image, the label corresponding to each image's hog features is maintained. This pair of hog features and their corresponding labels (0 for closed and 1 for open) are fed to fit the SVM model. This model is then saved to be loaded for later use.

```python
# Convert lists to numpy arrays
X = np.array(X)
y = np.array(y)

# Train SVM classifier
clf.fit(X, y)

# save the model
with open("svm_model.pkl", "wb") as f:
    pickle.dump(clf, f)
```

This saved model is then used to predict the label for each bounding box in each of the test images and this label is written in the center of the bounding box.

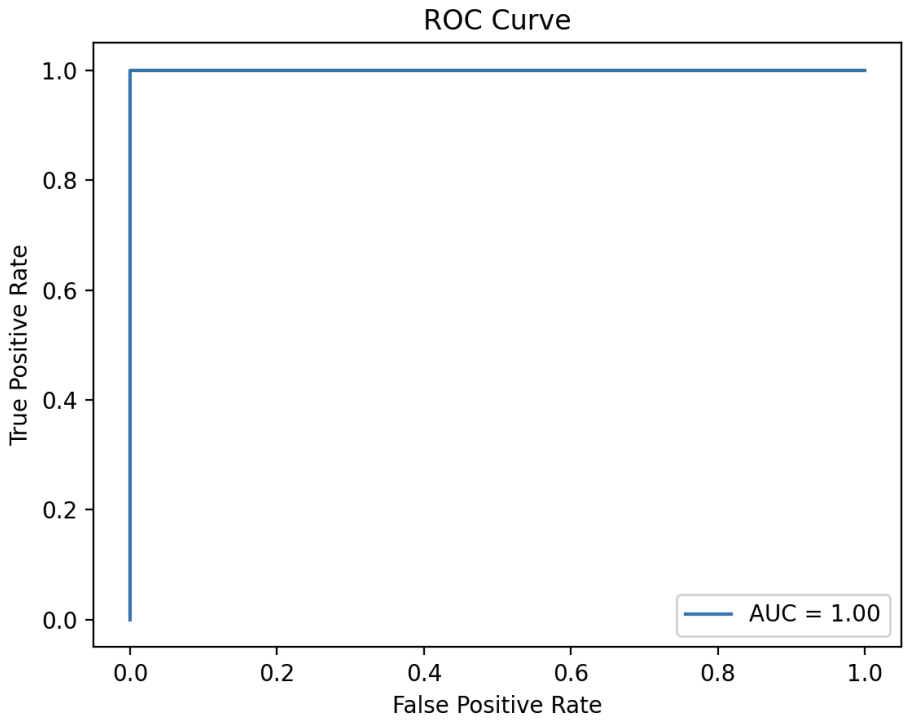
```python
# print("processing image: ", filename)
if results.multi_hand_landmarks:
    bounding_boxes = extract_bounding_boxes(img, results.multi_hand_landmarks)
    for bbox in bounding_boxes:
        hog_features = extract_hog_from_roi(img, bbox)
        # predict the hand state
        if hog_features is not None:
            hog_features = hog_features.reshape(1, -1)
            prediction = clf.predict(hog_features)
            # convert the prediction to integer
            prediction = int(prediction[0])
            # draw the bounding box on the image
            x, y, w, h = bbox
            cv2.rectangle(img_out, (x, y), (x+w, y+h), (0, 255, 0), 2)
            # add the prediction to the image
            if prediction == 0:
                prediction = "Closed"
            else:
                prediction = "Open"
            # cv2.putText(img_out, prediction, (x, y-10), cv2.FONT_HERSHEY_SIMPLEX, 0.9, (36,255,12), 2)
                # write at the center of the bounding box such that the text is center aligned
            text_size = cv2.getTextSize(prediction, cv2.FONT_HERSHEY_SIMPLEX, 0.9, 2)[0]
            text_x = x + (w - text_size[0]) // 2
            text_y = y + (h + text_size[1]) // 2
            cv2.putText(img_out, prediction, (text_x, text_y), cv2.FONT_HERSHEY_SIMPLEX, 0.9, (36,255,12), 2)

    # save the output image
    output_path = os.path.join(output_dir, filename)
    cv2.imwrite(output_path, img_out)
```

3. Results and Conclusion:

```
Time taken to train the model:  432.3725423812866 seconds
```

```
Time taken to test the model:  59.39766597747803 seconds
Accuracy: 1.0
Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00       328
           1       1.00      1.00      1.00       625

    accuracy                           1.00       953
   macro avg       1.00      1.00      1.00       953
weighted avg       1.00      1.00      1.00       953

False Positive Rate:  0.0
True Positive Rate:  1.0
False Negative Rate:  0.0
True Negative Rate:  1.0
```

```
AUC: 1.0
```



The time taken by my model (which includes calculating the hog descriptors for each image as well) to train is around 7 minutes when the model is trained on the entire train dataset, that is, open1, open2, closed1, closed2, closed3 data folders combined which is around 5-6k images.

The time taken by my model for testing on the validation dataset (all data folders combined which is around 800 images) is around 1 minute (which includes calculating the hog descriptors for each image as well).

We observe that the model gives 100% accuracy on the validation set, along with precision and recall values of 1, which indicate that all of the samples (open/closed) are correctly identified and that all the samples outputted by the model are of the correct class (open/closed).

Also, the false positive and false negative rate is 0, which indicates that the model does not incorrectly label any sample as open or closed.
The ROC curve plots the True Positive Rate vs False Positive Rate, and the graph clearly shows that the model has high true positive rate and low false positive rate.
The AUC (Area under ROC Curve) is 1, which indicates that the predictions are absolutely correct, and the model is well trained.

The conclusion is that the HOG-based feature detector works very well for this task of hand gesture recognition and can be employed in the real world to get the best result. Also, the time taken by this HOG-based approach is quite less as compared to the Deep Learning/CNN based approaches which require intensive GPU computation abilities and long periods of training time. The HOG based approach can be run locally and in a very reasonable amount of time to give extraordinary results.

4. Problems Faced:

The main problem I faced was that of building the hog feature extractor from scratch since it had various steps, and implementing each of them correctly took significant effort. I had to take care of the dimensions of the output after

performing each operation and verify what it should be, this required deeply understanding the hog feature extractor properly.

Another problem I faced was optimizing the parameters of the hog feature extractor, such as the block size and the window size to achieve the best possible accuracy.

Another problem I faced was that the ROI_Coordinates.py script , basically the mediapipe-based hands detector, often used to return negative or out of bounds coordinates of the bounding boxes. So for those cases i deliberately restricted the coordinates to be within the image, and the hog based feature detector was able to better detect the hands this way.

In creating the real world application, I faced the problem of firstly making a video dataset which looks somewhat like a real video from the given dataset since there were a lot of skipped frames in the dataset given. I somehow managed to combine the right images but then the problem was that while running on the video non in an infinite loop, my laptop used to run out of memory and the os used to kill the task. So I solved this problem by speeding up the playback of the video to reduce the load on the machine.

5. **Readme file:**

It has the instructions for running the code and the file structure mentioned in it.

6. Issues and Challenges Faced:

In the bonus part, while implementing a real time application of music player, I faced the challenge of running my model in realtime on the video and use it predictions to control the music player. This required the model to have fast inferences (which was effected by resizing the image to a small size) and appropriate gaps between capturing frames from the video.

I had also tried running the model on the live video, but even mediapipe was unable to detect hands in most cases and even if it detected in some frames, then the model was unable to classify correctly because the dataset given was of a very specific type. To make it work in real time video setting, we will have to

train the model on a significantly large dataset which covers various kinds of hands and positions.

I had also tried to use PCA for reducing the dimensionality of the hog feature detector but it was leading to reduced accuracy and not much reduction in time. Since my model training was taking somewhere around 7 minutes which was pretty fast, so I did not use PCA to further reduce the time.

Another issue I faced was that due to my webcam being of quite low resolution, I was unable to implement the live video version of the bonus part. My code was working perfectly fine while feeding a video to it, but using the real time video from the webcam led to poor results because of the very low resolution.

7. **Bonus Part:**

App1:

```
cap = cv2.VideoCapture(vid_file)

while True:
    # Read frame from video every 1s
    ret, frame = cap.read()
    if not ret:
        break

    # Process frame to control music player
    process_frame(frame, model, actions)


    # put the video reader one second forward
    cap.set(cv2.CAP_PROP_POS_MSEC, cap.get(cv2.CAP_PROP_POS_MSEC) + 400)

    # show the frame
    cv2.imshow("Hand Gesture Music Player", frame)

    # Exit if 'e' is pressed
    if cv2.waitKey(1) & 0xFF == ord('e'):
        break
```

I implemented a music player system controlled by hand gestures. For this, I first created a video by placing together the frames given in the dataset to compile a video. Then, running in an infinite loop, the model continuously runs its inferences on the video frame and based on whether the hand detected is open or closed, it either plays the video or pauses it.

```
print("Detected Gesture:", action)


# Perform action based on detected gesture
if action == "Open Hand - Play":
    # check if music is already playing
    if pygame.mixer.music.get_busy():
        # dont do anything
        print("Music is already playing...")
        return
    # check if music is paused
    if pygame.mixer.music.get_pos() > 0:
        # unpause the music
        pygame.mixer.music.unpause()
        print("Unpausing music...")
    else:
        # start playing the music
        pygame.mixer.music.play()
        print("Playing music...")
elif action == "Closed Hand - Pause":
    # check if music is already paused
    if pygame.mixer.music.get_busy() and pygame.mixer.music.get_pos() > 0:
        # pause the music
        pygame.mixer.music.pause()
        print("Pausing music...")
    else:
        # dont do anything
        print("Music is already paused...")
```

If the video is already playing and an open hand is shown, then it just continues playing the audio, and if the audio is paused and an open hand is shown then it unpauses the audio and resumes playing it from where it was paused. If the audio is playing and a closed hand is shown, then the model pauses the video, and if the video is already paused, then it won't do anything.

This has various real world applications, as it can be used to control a music player based on hand gestures by a user. If trained on a large enough corpus of dataset, this can be used to control the music player app in real time, so that when the user opens their hand, the music starts playing, and when the user closes the hand it pauses. This can also be extended to add more features such as next song and previous song, if trained on a large enough dataset consisting of such sample poses.

```
Prediction: [1]
Playing music...
I0000 00:00:1712069386.912352       1 gl_context.cc:344] GL version: 2.1 (2.1 Metal - 88), renderer: Apple M1
Prediction: [1]
I0000 00:00:1712069388.028014       1 gl_context.cc:344] GL version: 2.1 (2.1 Metal - 88), renderer: Apple M1
Prediction: [1]
I0000 00:00:1712069389.149670       1 gl_context.cc:344] GL version: 2.1 (2.1 Metal - 88), renderer: Apple M1
Prediction: [1]
I0000 00:00:1712069390.268764       1 gl_context.cc:344] GL version: 2.1 (2.1 Metal - 88), renderer: Apple M1
Prediction: [1]
I0000 00:00:1712069391.385390       1 gl_context.cc:344] GL version: 2.1 (2.1 Metal - 88), renderer: Apple M1
Prediction: [0]
Pausing music...
I0000 00:00:1712069392.503959       1 gl_context.cc:344] GL version: 2.1 (2.1 Metal - 88), renderer: Apple M1
```

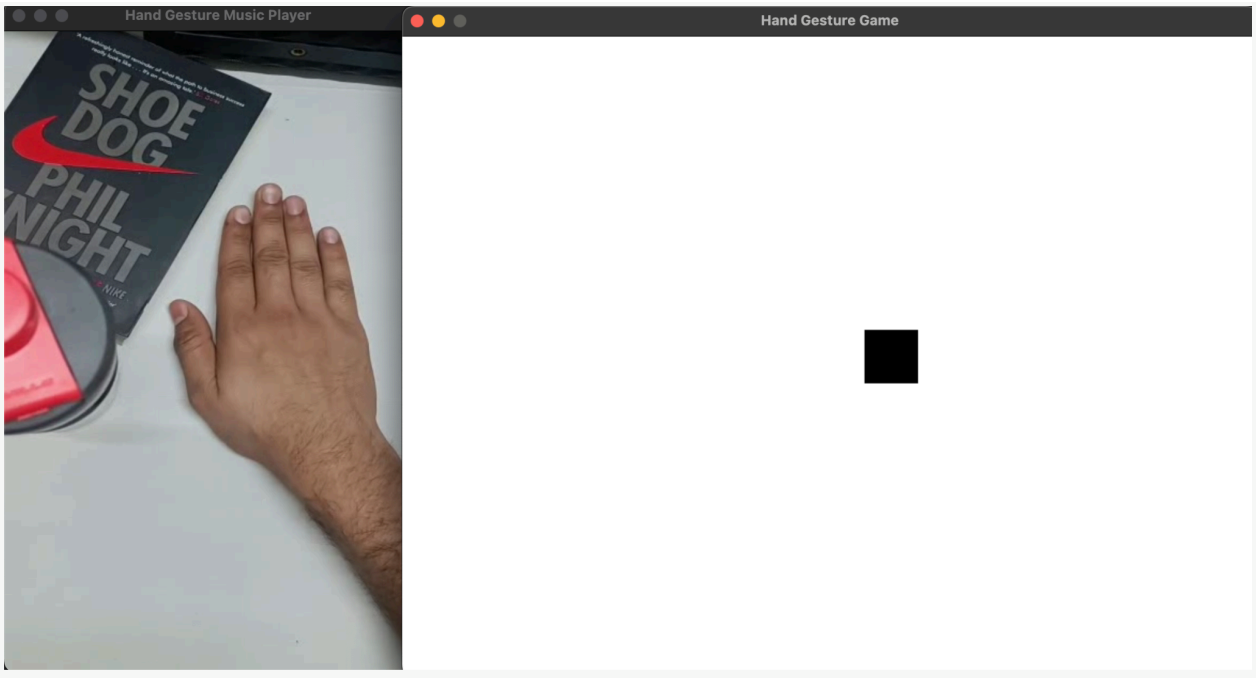
App2:

```python
# Extract hand gesture from frame
prediction = test_model_rt(frame, model)

# check if prediction is None
if prediction is None:
    print("No hand gesture detected.")
    return

# Get action corresponding to predicted hand gesture
action = prediction

# print the action
print("Detected Gesture:", action)

# Perform action based on detected gesture
if action == 0:  # Closed hand gesture moves object left
    move_object(-3, 0)
elif action == 1:  # Open hand gesture moves object right
    move_object(3, 0)
```

In the second real world app, I have built a simple game, in which the object's movement is controlled by the hand gesture recognition model we have built. So the object moves forward/right when the hand is open and it moves backward/left when the hand is closed.

```
# Initialize game object
object_width = 50
object_height = 50
global object_position
object_position = [screen_width // 2 - object_width // 2, screen_height // 2 - object_height // 2]
vid_file = video_file
cap = cv2.VideoCapture(vid_file)
# Game loop
running = True
while running:
    # Handle events
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            running = False

    # Capture frame from webcam
    ret, frame = cap.read()
    if not ret:
        break
    # Process frame to control game object movement
    process_frame(frame, model)
    # put the video reader one second forward
    cap.set(cv2.CAP_PROP_POS_MSEC, cap.get(cv2.CAP_PROP_POS_MSEC) + 400)
    # show the frame
    cv2.imshow("Hand Gesture Music Player", frame)
    # Draw background
    screen.fill(WHITE)
    # Draw game object
    pygame.draw.rect(screen, BLACK, [object_position[0], object_position[1], object_width, object_height])
    # Update display
    pygame.display.flip()
```

This has various applications in real world games, where the player movement can be controlled by hand gestures. Also on a large dataset having various kinds of hand gestures, like pointing or fist, it can be used to create a wider variety of actions and can make a more comprehensive experience of gameplay in games specially 3D and virtual reality games.



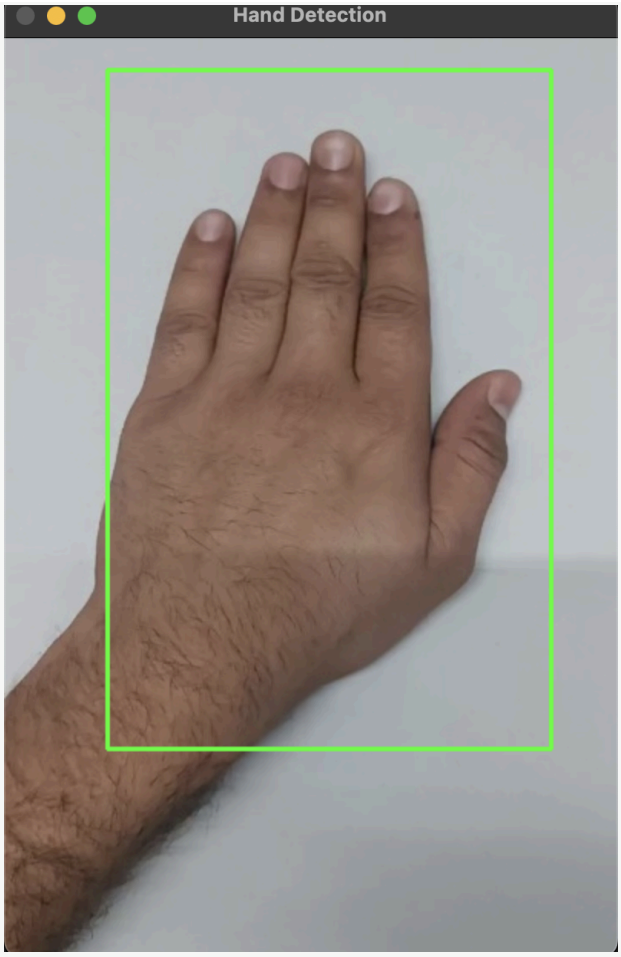
8. References:

Opencv reference- https://docs.opencv.org/4.x/index.html
Sklearn reference - https://scikit-learn.org/stable/
Pixabay(for sample audio): https://pixabay.com/sound-effects/search/samples/

Testing:

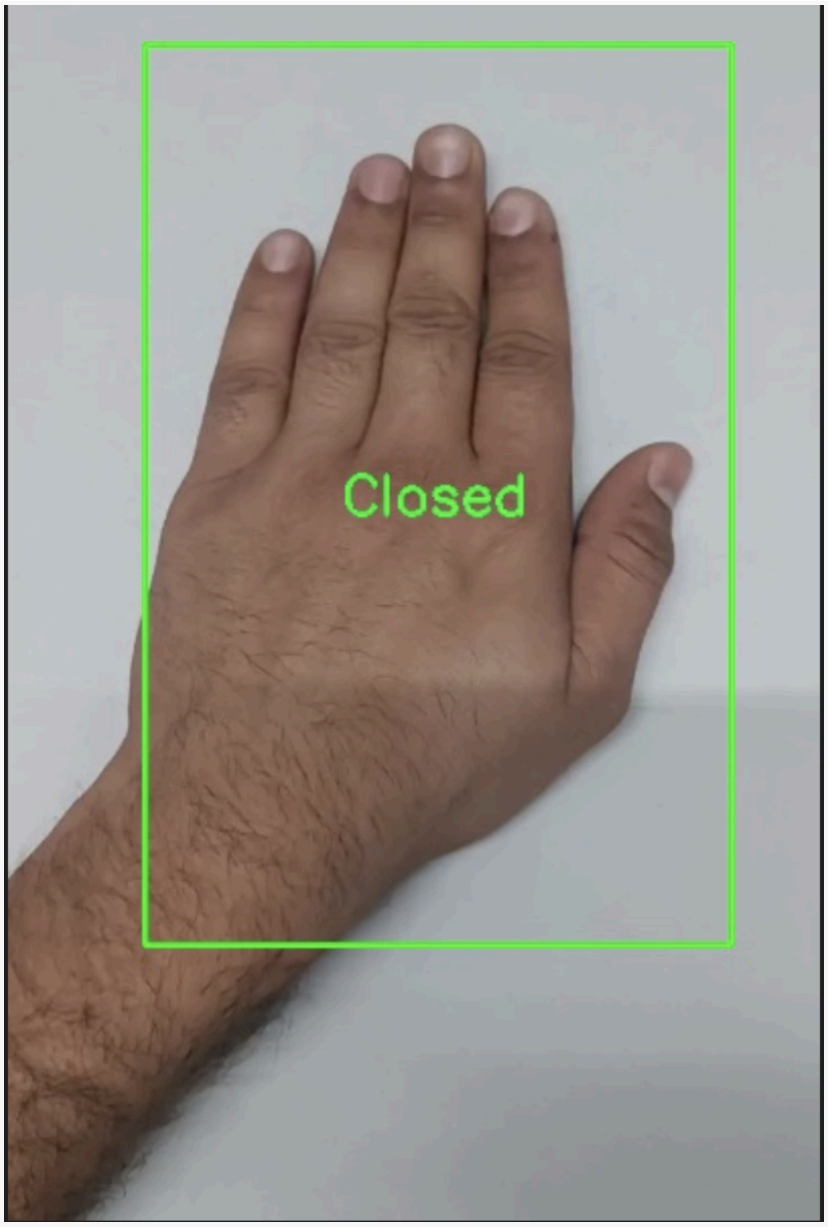
Test Image 1:

Bounding Box Detected on Image:




```
HOG Features:  [0.1790833  0.         0.0043959  ... 0.0111108  0.05669694 0.02735416]
Prediction: 0
Closed
```

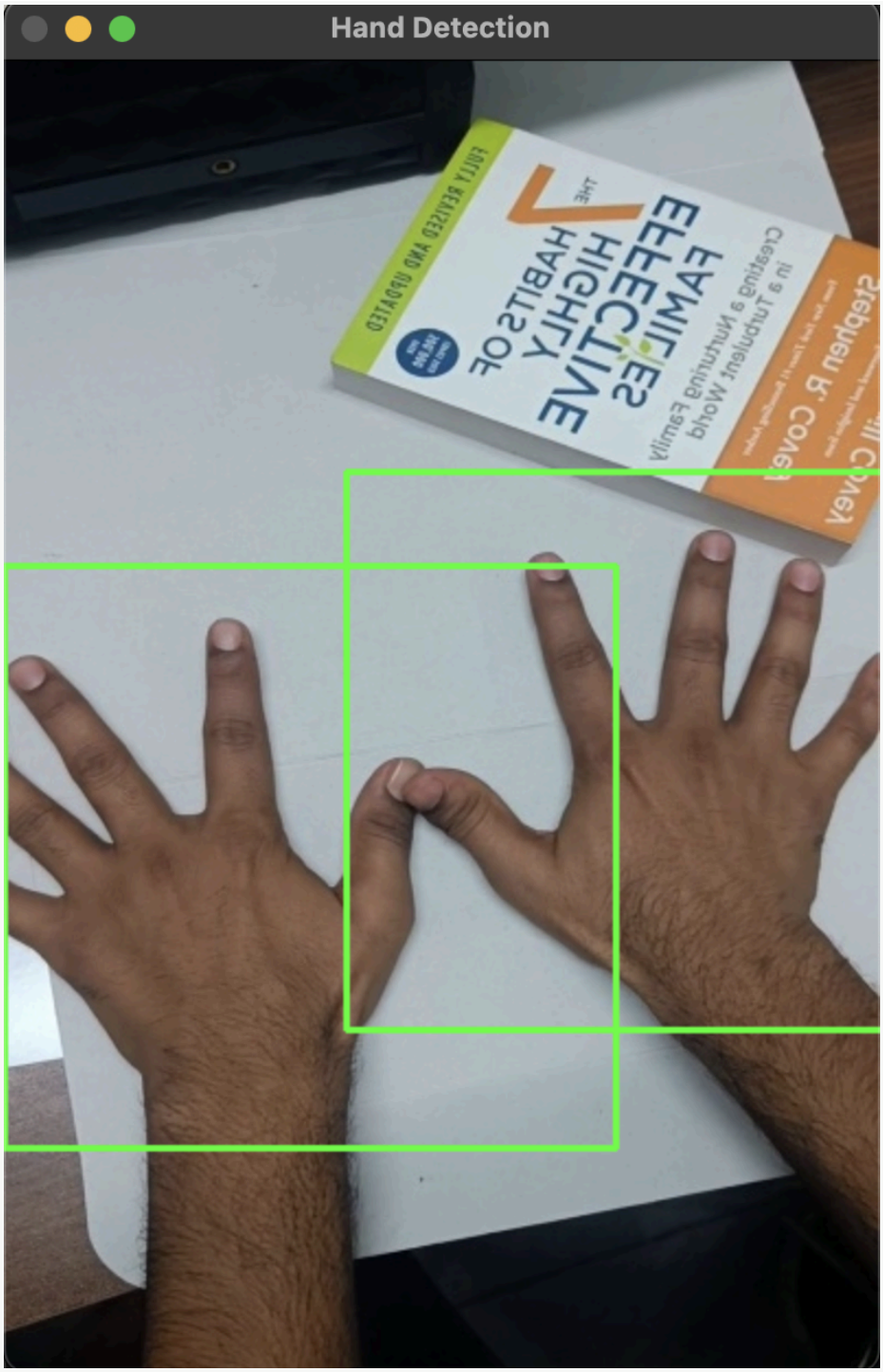
Final Output Image:



Test Image 2

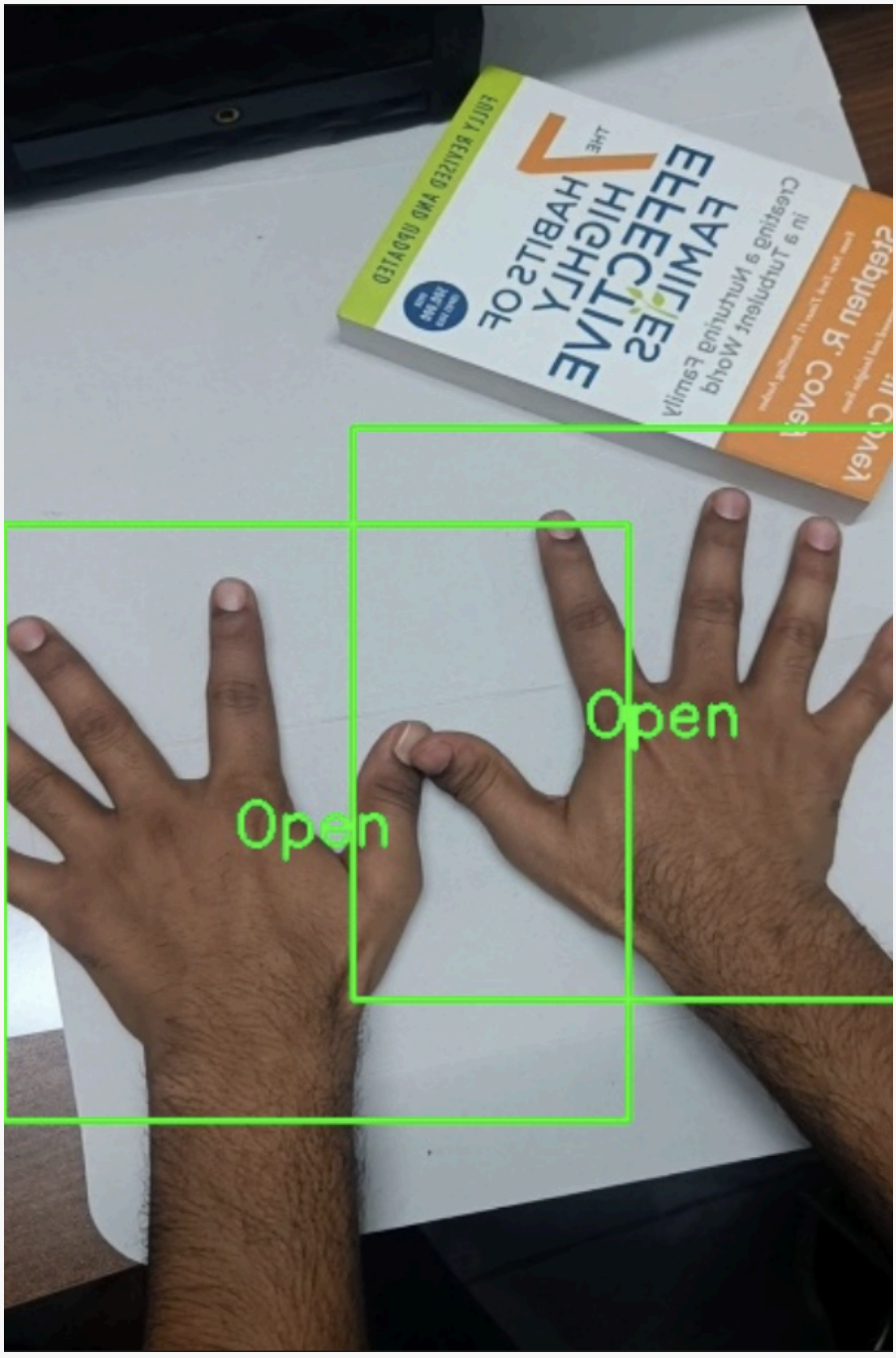

Image
THE 7
HABITS OF
HIGHLY
EFFECTIVE
FAMILIES
Stephen R. Covey

Bounding Boxes Detected on Image:



```
HOG Features:  [0.15370616 0.00332057 0.00315017 ... 0.00386166 0.162627   0.15106869]
Prediction: 1
HOG Features:  [0.26613076 0.00213667 0.10529172 ... 0.02221562 0.02369582 0.11246755]
Prediction: 1
Open Open
```

Final Output Image:

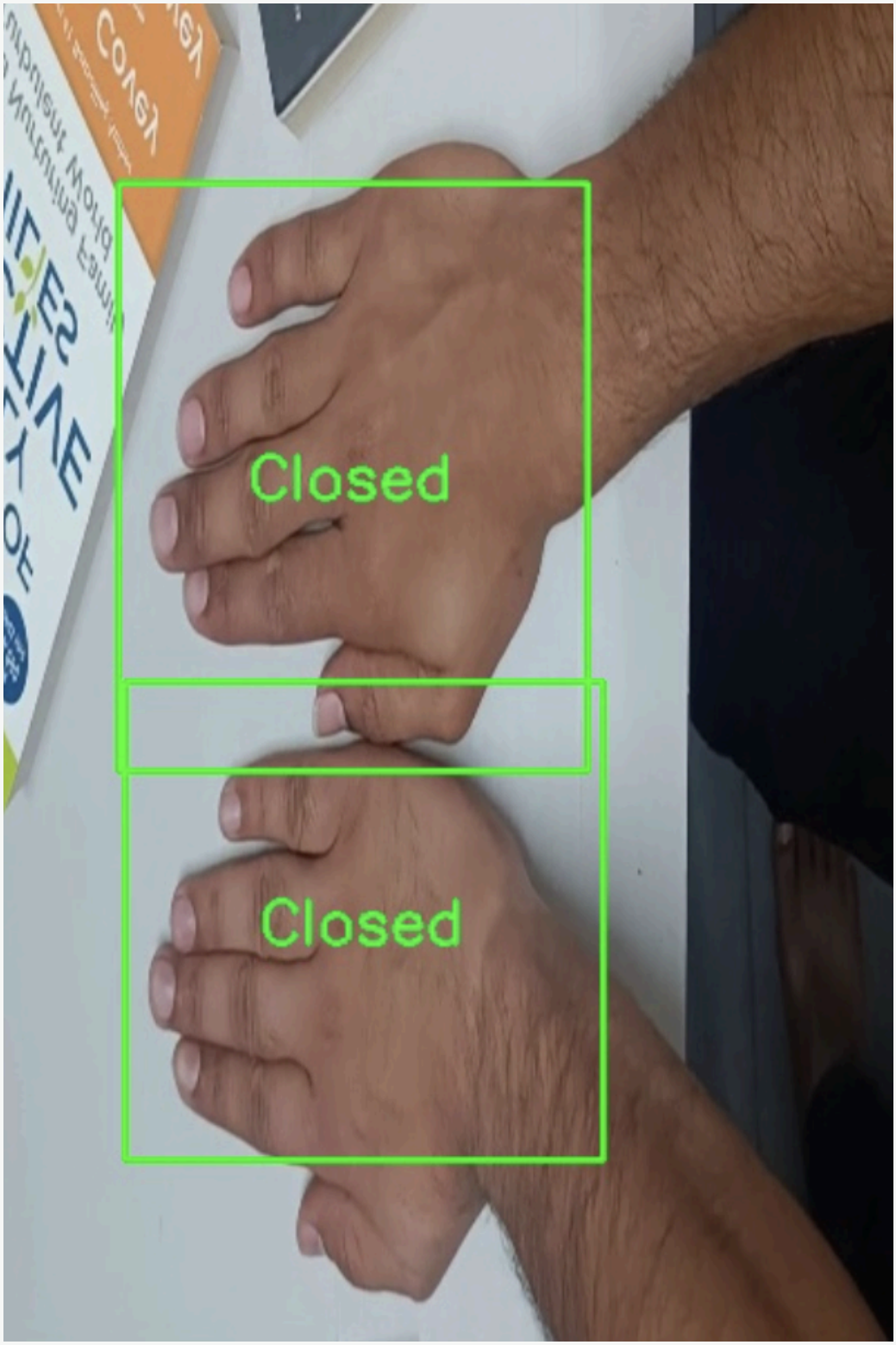


Test Image 3

Bounding Boxes Detected on Image:


```
HOG Features:  [0.1961194  0.01351446 0.10237973 ... 0.07565052 0.42134894 0.07106829]
Prediction: 0
HOG Features:  [0.33784258 0.         0.10285765 ... 0.08155744 0.04170271 0.14838202]
Prediction: 0
Closed Closed
```

Final Output Image:



Real World Application Output:

```
Prediction: [1]
Detected Gesture: Open Hand - Play
Playing music...
I0000 00:00:1712154120.348717       1 gl_context.cc:344] GL version: 2.1 (2.1 Metal - 88), renderer: Apple M1
Prediction: [1]
Detected Gesture: Open Hand - Play
Music is already playing...
```

```
Prediction: [0]
Detected Gesture: Closed Hand - Pause
Pausing music...
I0000 00:00:1712154128.205907       1 gl_context.cc:344] GL version: 2.1 (2.1 Metal - 88), renderer: Apple M1
Prediction: [0]
Detected Gesture: Closed Hand - Pause
Music is already paused...
```