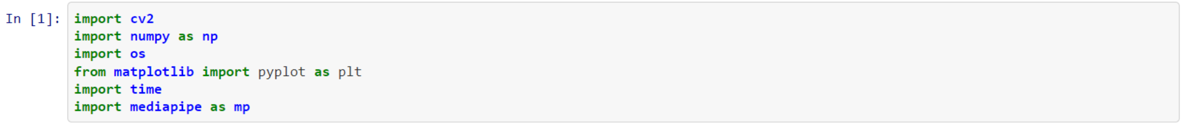
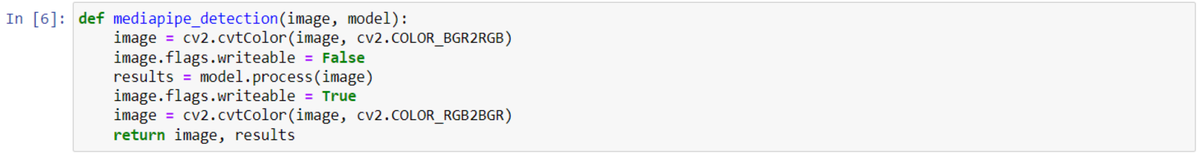
CHAPTER 5: EXPERIMENTATION

5.1 Installing Dependencies

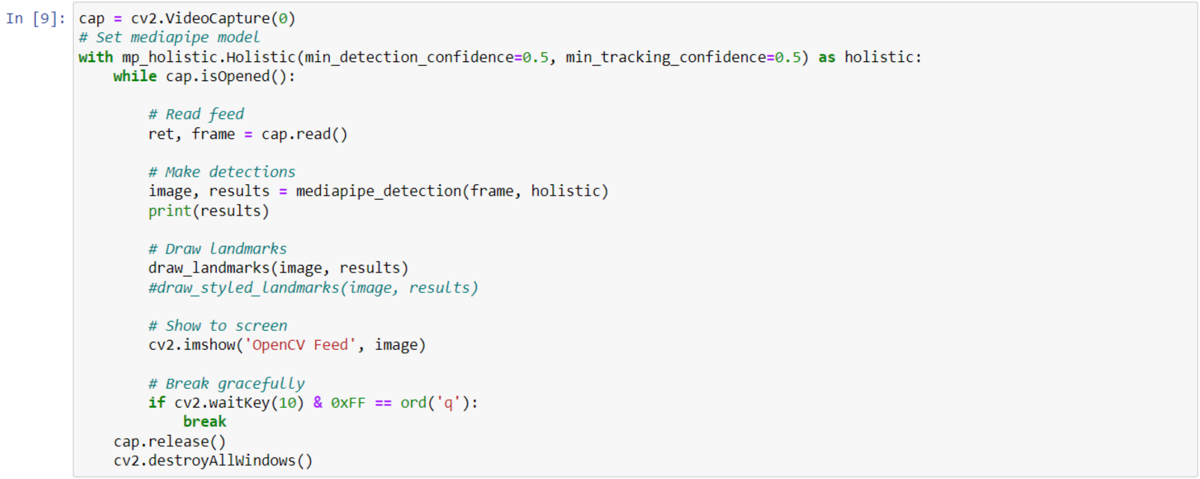


```
In [1]: import cv2
        import numpy as np
        import os
        from matplotlib import pyplot as plt
        import time
        import mediapipe as mp
```

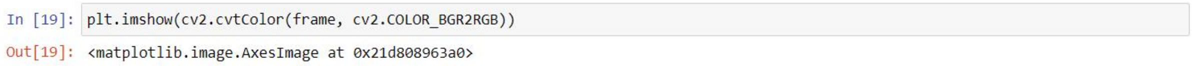
5.2 Keypoints detection using MP Holistic and OpenCV



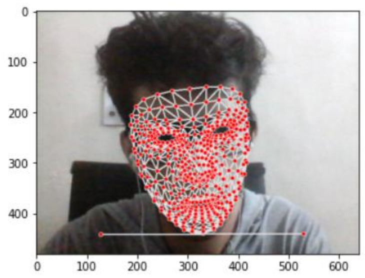
```
In [6]: def mediapipe_detection(image, model):
            image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
            image.flags.writeable = False
            results = model.process(image)
            image.flags.writeable = True
            image = cv2.cvtColor(image, cv2.COLOR_RGB2BGR)
            return image, results
```



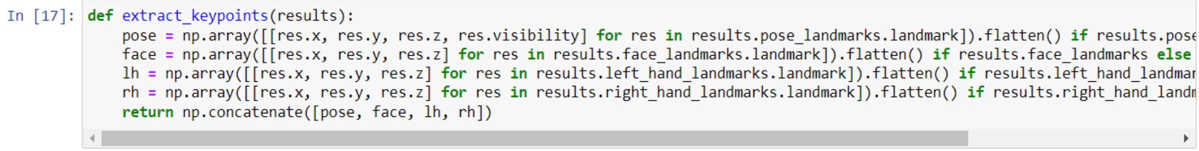
```
In [9]: cap = cv2.VideoCapture(0)
        # Set mediapipe model
        with mp_holistic.Holistic(min_detection_confidence=0.5, min_tracking_confidence=0.5) as holistic:
            while cap.isOpened():

                # Read feed
                ret, frame = cap.read()

                # Make detections
                image, results = mediapipe_detection(frame, holistic)
                print(results)

                # Draw landmarks
                draw_landmarks(image, results)
                #draw_styled_landmarks(image, results)

                # Show to screen
                cv2.imshow('OpenCV Feed', image)

                # Break gracefully
                if cv2.waitKey(10) & 0xFF == ord('q'):
                    break
            cap.release()
            cv2.destroyAllWindows()
```



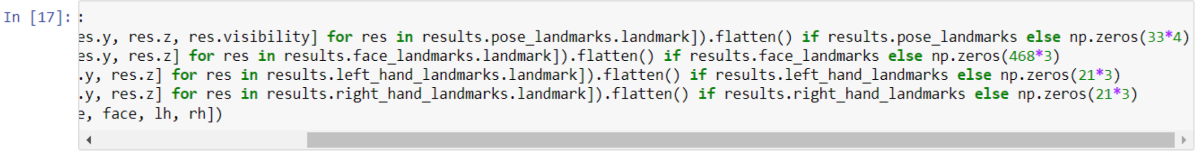
```
In [19]: plt.imshow(cv2.cvtColor(frame, cv2.COLOR_BGR2RGB))
Out[19]: <matplotlib.image.AxesImage at 0x21d808963a0>
```



5.3 Extract Key point Values

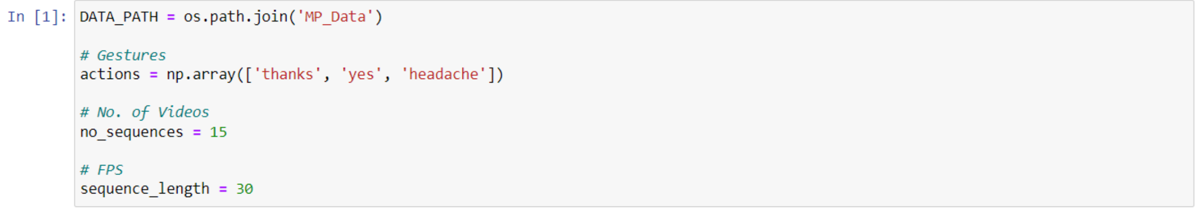


```
In [17]: def extract_keypoints(results):
             pose = np.array([[res.x, res.y, res.z, res.visibility] for res in results.pose_landmarks.landmark]).flatten() if results.pose
             face = np.array([[res.x, res.y, res.z] for res in results.face_landmarks.landmark]).flatten() if results.face_landmarks else
             lh = np.array([[res.x, res.y, res.z] for res in results.left_hand_landmarks.landmark]).flatten() if results.left_hand_landmar
             rh = np.array([[res.x, res.y, res.z] for res in results.right_hand_landmarks.landmark]).flatten() if results.right_hand_landm
             return np.concatenate([pose, face, lh, rh])
```

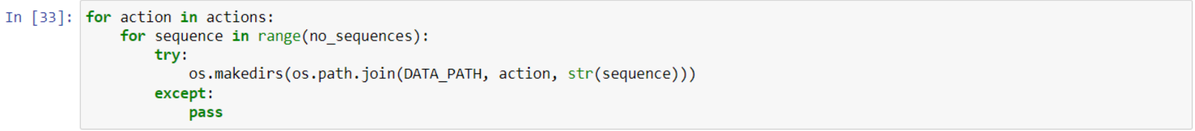


```
In [17]: :
         es.y, res.z, res.visibility] for res in results.pose_landmarks.landmark]).flatten() if results.pose_landmarks else np.zeros(33*4)
         es.y, res.z] for res in results.face_landmarks.landmark]).flatten() if results.face_landmarks else np.zeros(468*3)
         .y, res.z] for res in results.left_hand_landmarks.landmark]).flatten() if results.left_hand_landmarks else np.zeros(21*3)
         .y, res.z] for res in results.right_hand_landmarks.landmark]).flatten() if results.right_hand_landmarks else np.zeros(21*3)
         e, face, lh, rh])
```

5.4 Setup Folders for Collection

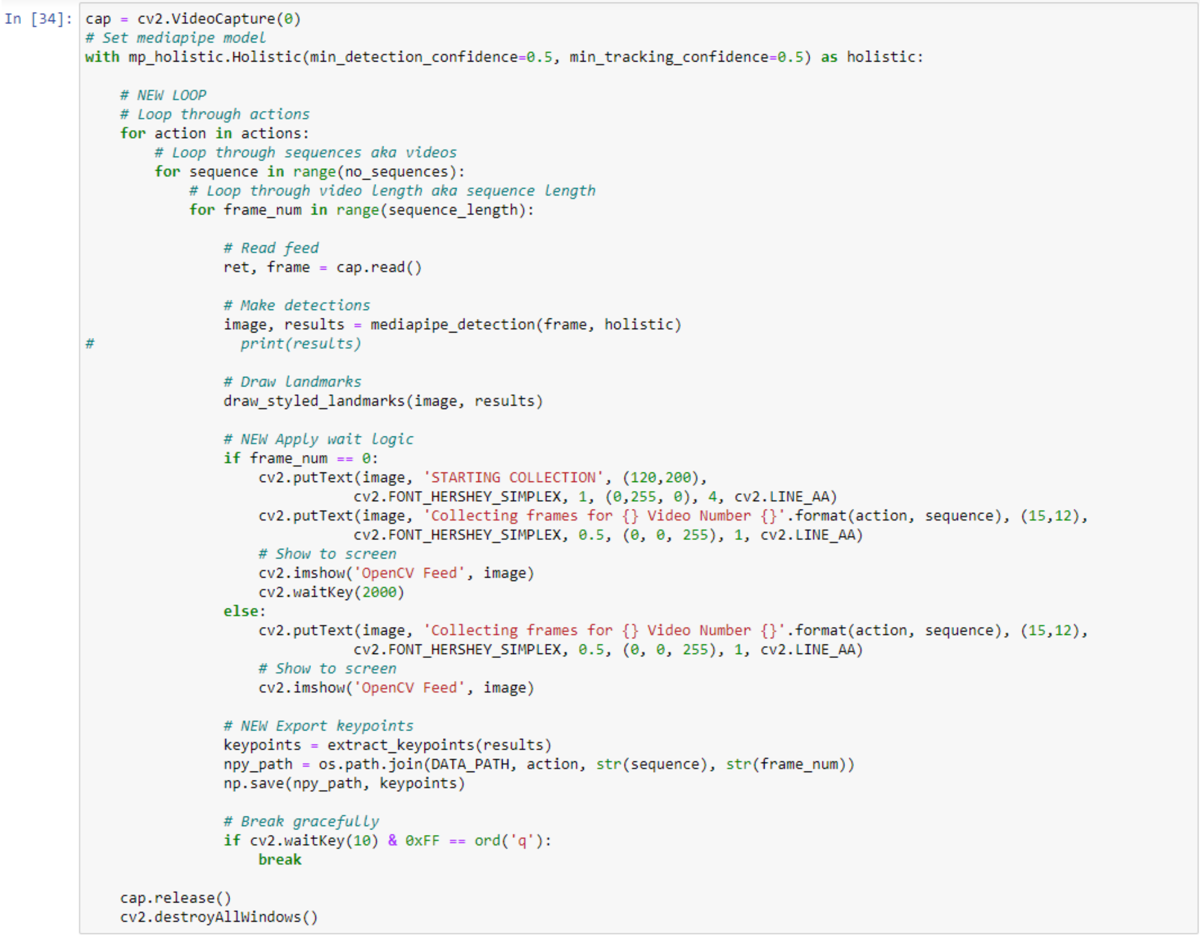


```
In [1]: DATA_PATH = os.path.join('MP_Data')

        # Gestures
        actions = np.array(['thanks', 'yes', 'headache'])

        # No. of Videos
        no_sequences = 15

        # FPS
        sequence_length = 30
```



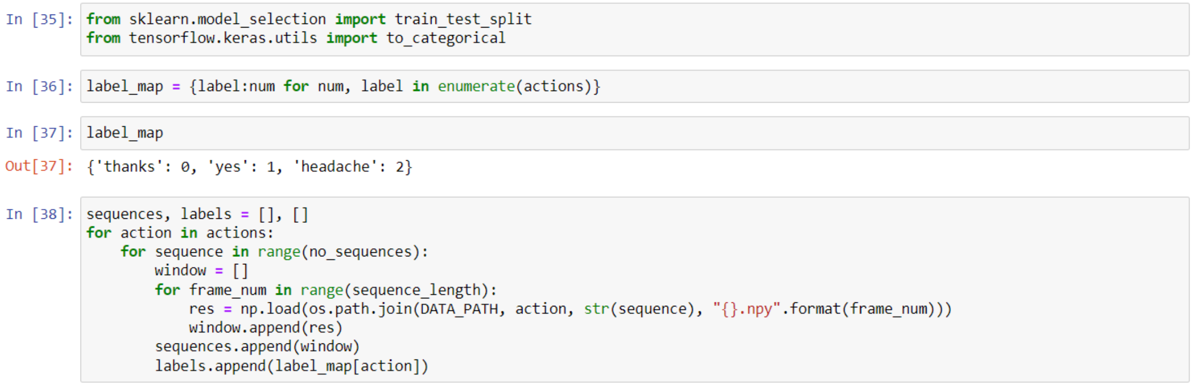
```
In [33]: for action in actions:
             for sequence in range(no_sequences):
                 try:
                     os.makedirs(os.path.join(DATA_PATH, action, str(sequence)))
                 except:
                     pass
```

5.5 Collect Key point Values for Training and Testing



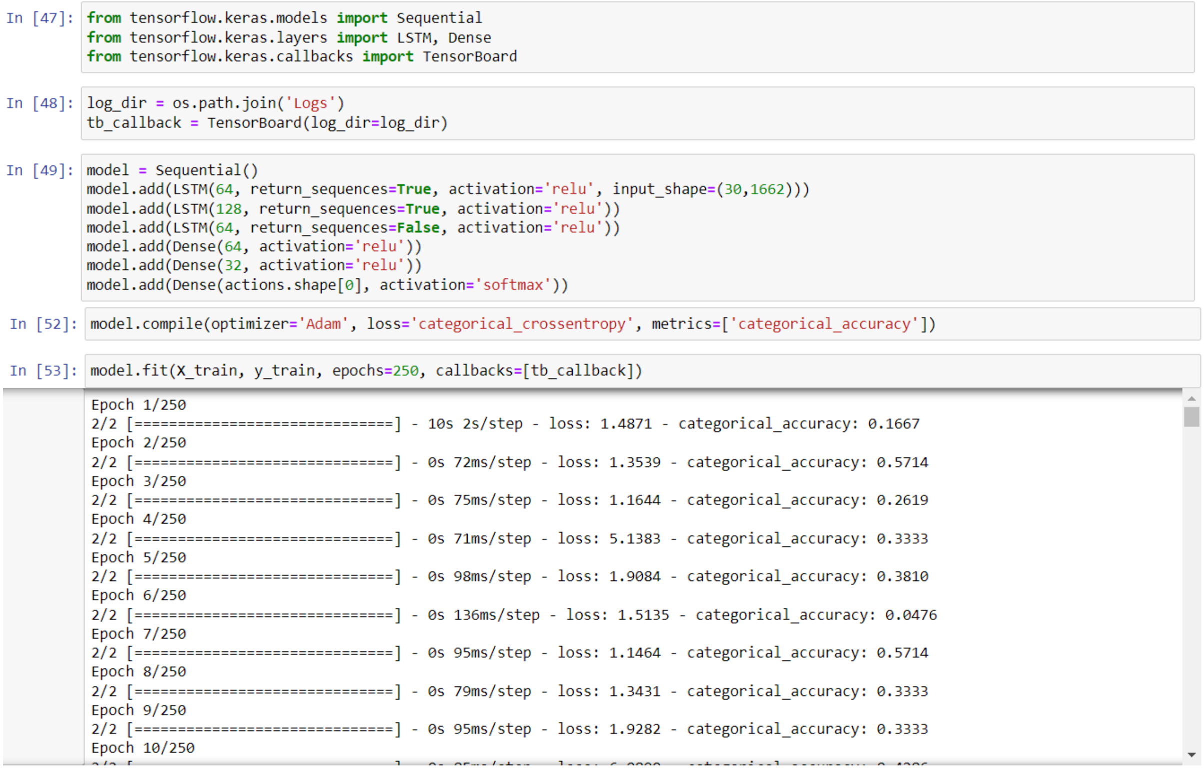
```
In [34]: cap = cv2.VideoCapture(0)
         # Set mediapipe model
         with mp_holistic.Holistic(min_detection_confidence=0.5, min_tracking_confidence=0.5) as holistic:

             # NEW LOOP
             # Loop through actions
             for action in actions:
                 # Loop through sequences aka videos
                 for sequence in range(no_sequences):
                     # Loop through video length aka sequence length
                     for frame_num in range(sequence_length):

                         # Read feed
                         ret, frame = cap.read()

                         # Make detections
                         image, results = mediapipe_detection(frame, holistic)
         #                 print(results)

                         # Draw landmarks
                         draw_styled_landmarks(image, results)

                         # NEW Apply wait logic
                         if frame_num == 0:
                             cv2.putText(image, 'STARTING COLLECTION', (120,200),
                                        cv2.FONT_HERSHEY_SIMPLEX, 1, (0,255, 0), 4, cv2.LINE_AA)
                             cv2.putText(image, 'Collecting frames for {} Video Number {}'.format(action, sequence), (15,12),
                                        cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0, 0, 255), 1, cv2.LINE_AA)
                             # Show to screen
                             cv2.imshow('OpenCV Feed', image)
                             cv2.waitKey(2000)
                         else:
                             cv2.putText(image, 'Collecting frames for {} Video Number {}'.format(action, sequence), (15,12),
                                        cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0, 0, 255), 1, cv2.LINE_AA)
                             # Show to screen
                             cv2.imshow('OpenCV Feed', image)

                         # NEW Export keypoints
                         keypoints = extract_keypoints(results)
                         npy_path = os.path.join(DATA_PATH, action, str(sequence), str(frame_num))
                         np.save(npy_path, keypoints)

                         # Break gracefully
                         if cv2.waitKey(10) & 0xFF == ord('q'):
                             break

             cap.release()
             cv2.destroyAllWindows()
```

5.6 Pre-process Data and Create Labels and Features

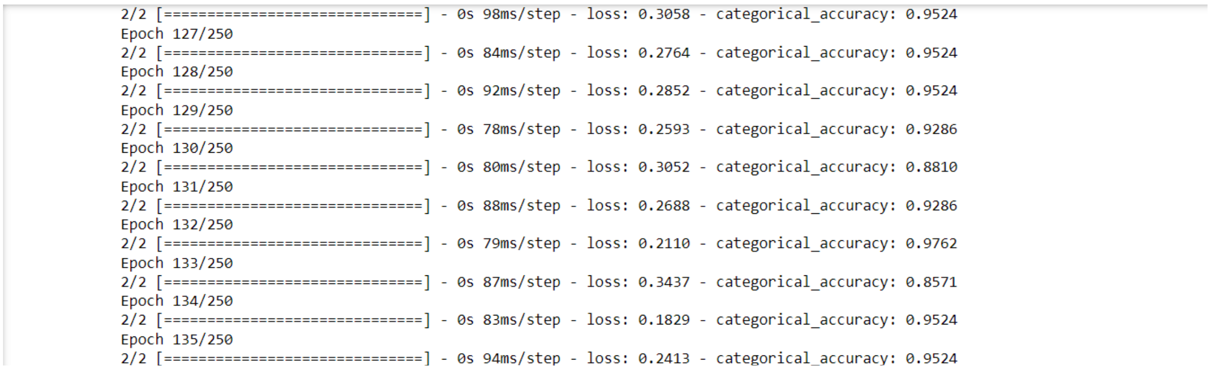


```
In [35]: from sklearn.model_selection import train_test_split
         from tensorflow.keras.utils import to_categorical

In [36]: label_map = {label:num for num, label in enumerate(actions)}

In [37]: label_map

Out[37]: {'thanks': 0, 'yes': 1, 'headache': 2}

In [38]: sequences, labels = [], []
         for action in actions:
             for sequence in range(no_sequences):
                 window = []
                 for frame_num in range(sequence_length):
                     res = np.load(os.path.join(DATA_PATH, action, str(sequence), "{}.npy".format(frame_num)))
                     window.append(res)
                 sequences.append(window)
                 labels.append(label_map[action])
```

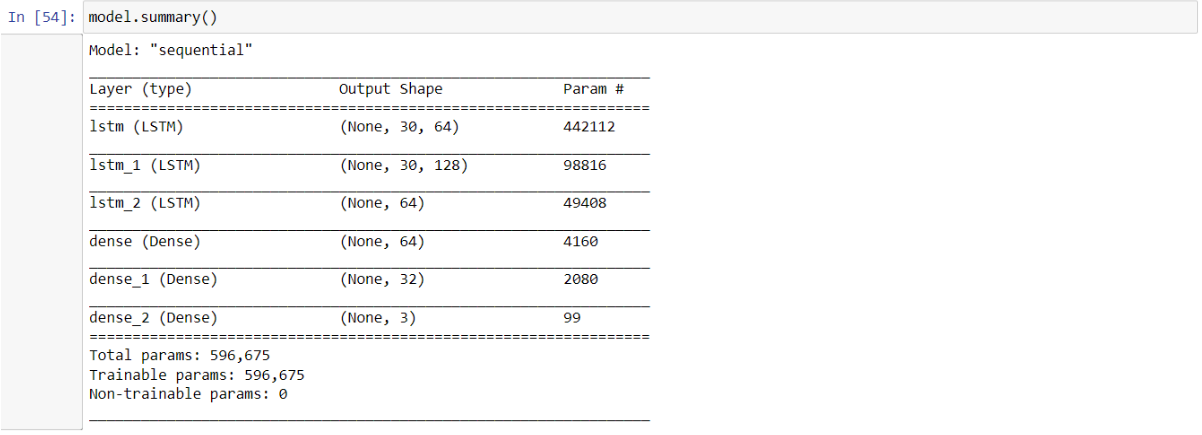
5.7 Build and Train LSTM Neural Network



```
In [47]: from tensorflow.keras.models import Sequential
         from tensorflow.keras.layers import LSTM, Dense
         from tensorflow.keras.callbacks import TensorBoard

In [48]: log_dir = os.path.join('Logs')
         tb_callback = TensorBoard(log_dir=log_dir)

In [49]: model = Sequential()
         model.add(LSTM(64, return_sequences=True, activation='relu', input_shape=(30,1662)))
         model.add(LSTM(128, return_sequences=True, activation='relu'))
         model.add(LSTM(64, return_sequences=False, activation='relu'))
         model.add(Dense(64, activation='relu'))
         model.add(Dense(32, activation='relu'))
         model.add(Dense(actions.shape[0], activation='softmax'))

In [52]: model.compile(optimizer='Adam', loss='categorical_crossentropy', metrics=['categorical_accuracy'])

In [53]: model.fit(X_train, y_train, epochs=250, callbacks=[tb_callback])

         Epoch 1/250
         2/2 [==============================] - 10s 2s/step - loss: 1.4871 - categorical_accuracy: 0.1667
         Epoch 2/250
         2/2 [==============================] - 0s 72ms/step - loss: 1.3539 - categorical_accuracy: 0.5714
         Epoch 3/250
         2/2 [==============================] - 0s 75ms/step - loss: 1.1644 - categorical_accuracy: 0.2619
         Epoch 4/250
         2/2 [==============================] - 0s 71ms/step - loss: 5.1383 - categorical_accuracy: 0.3333
         Epoch 5/250
         2/2 [==============================] - 0s 98ms/step - loss: 1.9084 - categorical_accuracy: 0.3810
         Epoch 6/250
         2/2 [==============================] - 0s 136ms/step - loss: 1.5135 - categorical_accuracy: 0.0476
         Epoch 7/250
         2/2 [==============================] - 0s 95ms/step - loss: 1.1464 - categorical_accuracy: 0.5714
         Epoch 8/250
         2/2 [==============================] - 0s 79ms/step - loss: 1.3431 - categorical_accuracy: 0.3333
         Epoch 9/250
         2/2 [==============================] - 0s 95ms/step - loss: 1.9282 - categorical_accuracy: 0.3333
         Epoch 10/250
```

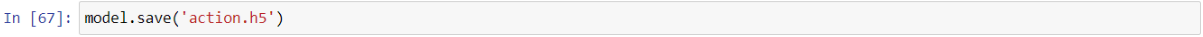


```
2/2 [==============================] - 0s 98ms/step - loss: 0.3058 - categorical_accuracy: 0.9524
Epoch 127/250
2/2 [==============================] - 0s 84ms/step - loss: 0.2764 - categorical_accuracy: 0.9524
Epoch 128/250
2/2 [==============================] - 0s 92ms/step - loss: 0.2852 - categorical_accuracy: 0.9524
Epoch 129/250
2/2 [==============================] - 0s 78ms/step - loss: 0.2593 - categorical_accuracy: 0.9286
Epoch 130/250
2/2 [==============================] - 0s 80ms/step - loss: 0.3052 - categorical_accuracy: 0.8810
Epoch 131/250
2/2 [==============================] - 0s 88ms/step - loss: 0.2688 - categorical_accuracy: 0.9286
Epoch 132/250
2/2 [==============================] - 0s 79ms/step - loss: 0.2110 - categorical_accuracy: 0.9762
Epoch 133/250
2/2 [==============================] - 0s 87ms/step - loss: 0.3437 - categorical_accuracy: 0.8571
Epoch 134/250
2/2 [==============================] - 0s 83ms/step - loss: 0.1829 - categorical_accuracy: 0.9524
Epoch 135/250
2/2 [==============================] - 0s 94ms/step - loss: 0.2413 - categorical_accuracy: 0.9524
```



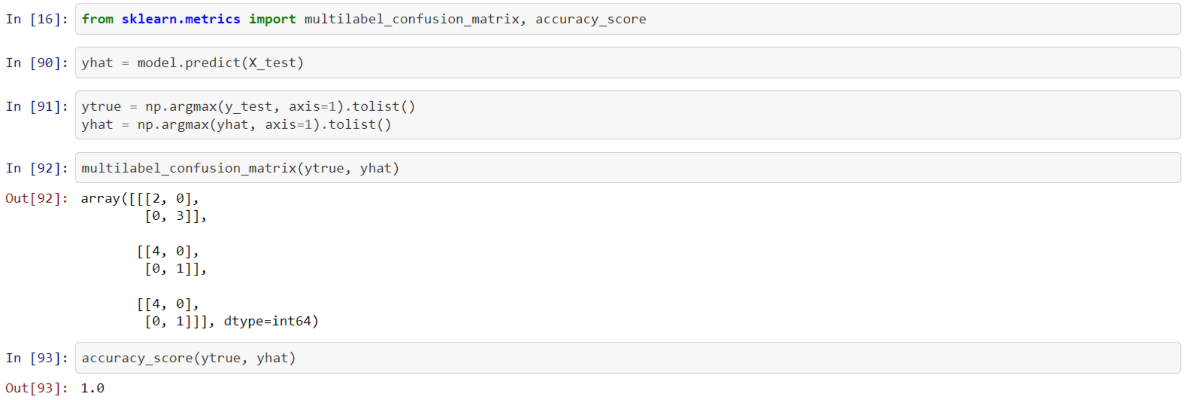
```
In [54]: model.summary()

         Model: "sequential"
         _________________________________________________________________
         Layer (type)                 Output Shape              Param #
         =================================================================
         lstm (LSTM)                  (None, 30, 64)            442112
         _________________________________________________________________
         lstm_1 (LSTM)                (None, 30, 128)           98816
         _________________________________________________________________
         lstm_2 (LSTM)                (None, 64)                49408
         _________________________________________________________________
         dense (Dense)                (None, 64)                4160
         _________________________________________________________________
         dense_1 (Dense)              (None, 32)                2080
         _________________________________________________________________
         dense_2 (Dense)              (None, 3)                 99
         =================================================================
         Total params: 596,675
         Trainable params: 596,675
         Non-trainable params: 0
         _________________________________________________________________
```

5.8 Save Weights



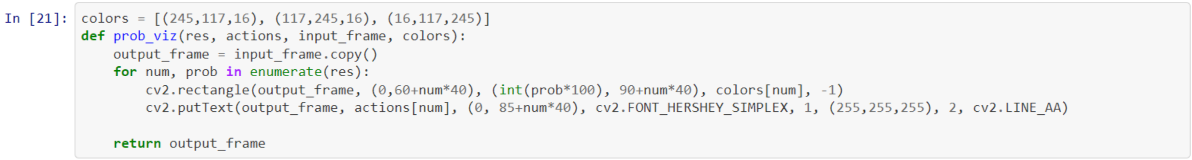
```
In [67]: model.save('action.h5')
```

5.9 Evaluation using Confusion Matrix and Accuracy

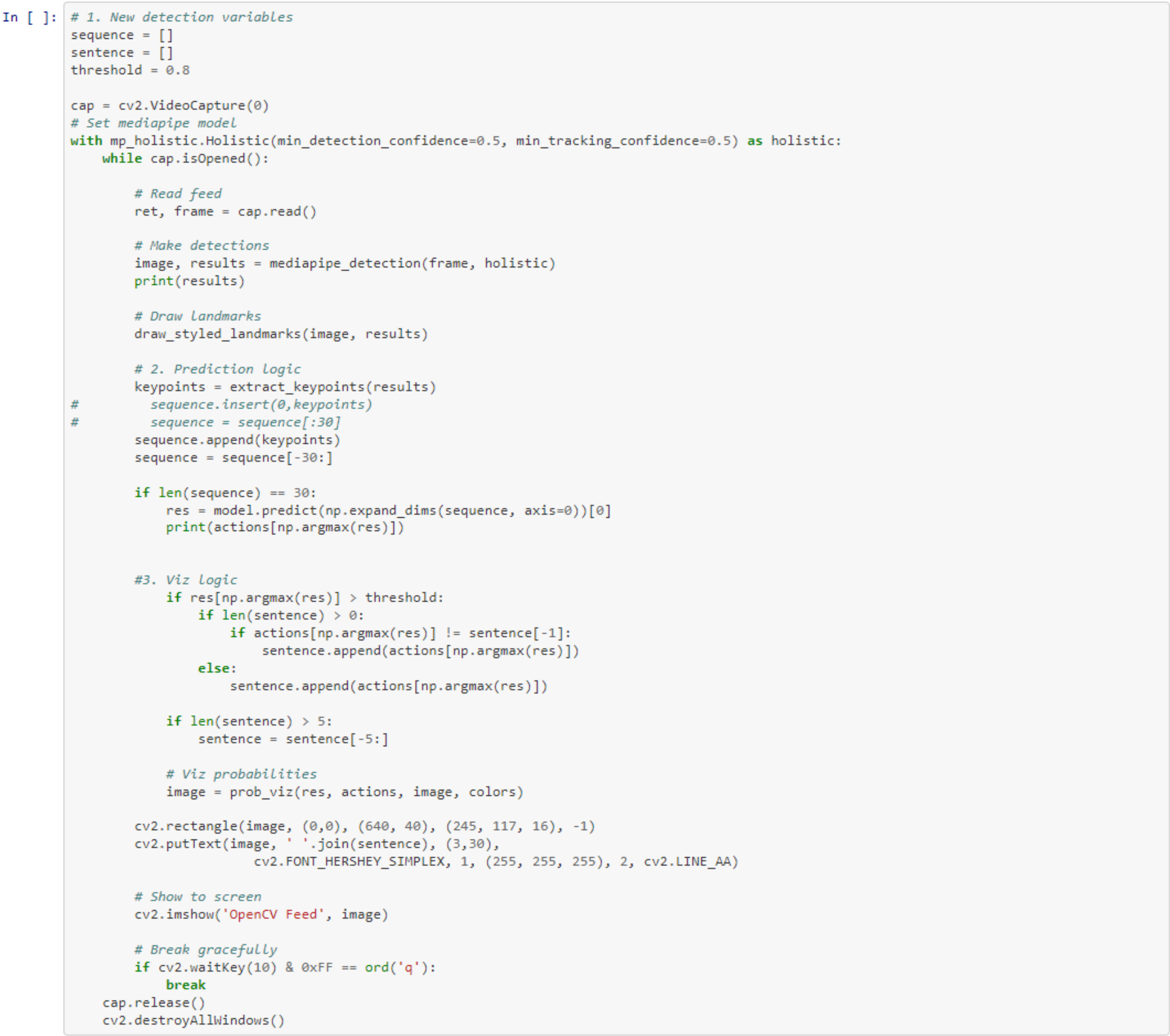


```
In [16]: from sklearn.metrics import multilabel_confusion_matrix, accuracy_score

In [90]: yhat = model.predict(X_test)

In [91]: ytrue = np.argmax(y_test, axis=1).tolist()
         yhat = np.argmax(yhat, axis=1).tolist()

In [92]: multilabel_confusion_matrix(ytrue, yhat)

Out[92]: array([[[2, 0],
                 [0, 3]],

                [[4, 0],
                 [0, 1]],

                [[4, 0],
                 [0, 1]]], dtype=int64)

In [93]: accuracy_score(ytrue, yhat)

Out[93]: 1.0
```

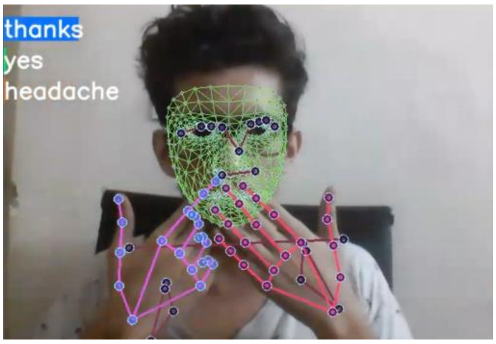
5.10 Test in Real Time

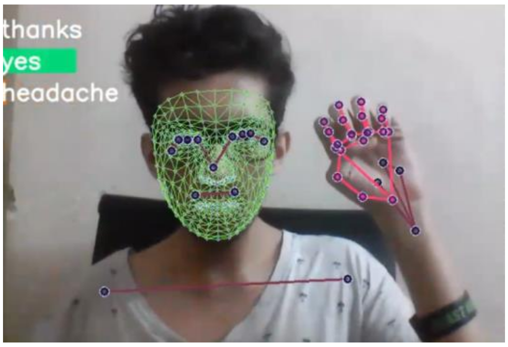


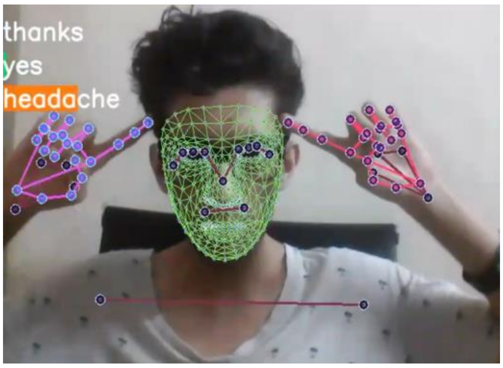
```
In [21]: colors = [(245,117,16), (117,245,16), (16,117,245)]
         def prob_viz(res, actions, input_frame, colors):
             output_frame = input_frame.copy()
             for num, prob in enumerate(res):
                 cv2.rectangle(output_frame, (0,60+num*40), (int(prob*100), 90+num*40), colors[num], -1)
                 cv2.putText(output_frame, actions[num], (0, 85+num*40), cv2.FONT_HERSHEY_SIMPLEX, 1, (255,255,255), 2, cv2.LINE_AA)

             return output_frame
```



```
In [ ]: # 1. New detection variables
        sequence = []
        sentence = []
        threshold = 0.8

        cap = cv2.VideoCapture(0)
        # Set mediapipe model
        with mp_holistic.Holistic(min_detection_confidence=0.5, min_tracking_confidence=0.5) as holistic:
            while cap.isOpened():

                # Read feed
                ret, frame = cap.read()

                # Make detections
                image, results = mediapipe_detection(frame, holistic)
                print(results)

                # Draw landmarks
                draw_styled_landmarks(image, results)

                # 2. Prediction logic
                keypoints = extract_keypoints(results)
        #         sequence.insert(0,keypoints)
        #         sequence = sequence[:30]
                sequence.append(keypoints)
                sequence = sequence[-30:]

                if len(sequence) == 30:
                    res = model.predict(np.expand_dims(sequence, axis=0))[0]
                    print(actions[np.argmax(res)])


                #3. Viz logic
                    if res[np.argmax(res)] > threshold:
                        if len(sentence) > 0:
                            if actions[np.argmax(res)] != sentence[-1]:
                                sentence.append(actions[np.argmax(res)])
                        else:
                            sentence.append(actions[np.argmax(res)])

                    if len(sentence) > 5:
                        sentence = sentence[-5:]

                    # Viz probabilities
                    image = prob_viz(res, actions, image, colors)

                cv2.rectangle(image, (0,0), (640, 40), (245, 117, 16), -1)
                cv2.putText(image, ' '.join(sentence), (3,30),
                               cv2.FONT_HERSHEY_SIMPLEX, 1, (255, 255, 255), 2, cv2.LINE_AA)

                # Show to screen
                cv2.imshow('OpenCV Feed', image)

                # Break gracefully
                if cv2.waitKey(10) & 0xFF == ord('q'):
                    break
            cap.release()
            cv2.destroyAllWindows()
```


thanks
yes
headache


thanks
yes
headache


thanks
yes
headache