Sistemi ad Agenti

eLis

a.a. 2021-2022

VR Group

[Informatica e Tecnologie per la Produzione del Software]

Link GitHub:
https://github.com/CatinoRosalba/WebApp_LisDetection

Realizzato da:

Catino Rosalba	718326	r.catino1@studenti.uniba.it
Lotito Vito	717828	v.lotito10@studenti.uniba.it

Sommario

Introduzione

eLis nasce come un progetto mirato ad insegnare ai bambini il **LIS** (linguaggio italiano dei segni).
Il funzionamento consiste nel catturare il movimento richiesto all'utente attraverso la camera del dispositivo in utilizzo. Il riconoscimento avviene attraverso una rete neurale allenata su un dataset iniziale di 6 parole.
Il tipo di riconoscimento utilizzato è l'"**action detection**" in quanto i segni del LIS sono caratterizzati da movimenti.

Dataset

Dopo alcune ricerche per un dataset formato da video di segni del LIS abbiamo constatato che non esiste un dataset adatto all'action detection. Sono stati, quindi, **sviluppati** degli script per la raccolta dei video che formeranno il dataset di partenza del progetto.
Il dataset è stato realizzato non solo utilizzando video presi dal web, ma anche con l'aiuto di alcuni volontari. Inoltre, alcuni video sono stati moltiplicati per raggiungere un numero adeguato di video (75 video per ogni segno) su cui effettuare l'allenamento della rete neurale.

Le parole disponibili al momento sono: ciao, grazie, prego, mangiare, bere, amico; è stata realizzata anche una posa nulla (null) nel caso non si eseguano segni.

Gli **script** creati sono:

- collect_dataset.py
- actionDetection_helper.py

Le **librerie** utilizzate sono:

- mediapipe (utilizzata dallo script actionDetection_helper.py);
- open-cv Python.

collect_dataset.py

Di seguito è spiegato il funzionamento dello script:

```
segni = np.array(['amico', 'mangiare', 'bere', 'grazie', 'prego', 'ciao', 'null'])
n_video = 5
frame_video = 30
```

I segni, il numero dei video e dei frame vengono definiti negli array all'inizio dello script.

```
def create_folders():
    os.chdir(str(os.getcwd()))
    if not os.path.exists('DataSet'):
        os.mkdir('DataSet')
    if not os.path.exists('keypointsDataset'):
        os.mkdir('keypointsDataset')
```

Il primo metodo che viene eseguito dallo script è create_folders() il quale crea la cartella "DataSet" in cui saranno conservati i video registrati e la cartella "keypointsDataset" in cui saranno conservati i keypoints estratti dai video.

```
def video_position():
    cap = cv2.VideoCapture(0)
    with ddc.mp_holistic.Holistic(min_detection_confidence=0.5, min_tracking_confidence=0.5) as holistic:
        while cap.isOpened():
            ret, frame = cap.read()
            image, results = ddc.mediapipe_detection(frame, holistic)
            ddc.draw_styled_landmarks(image, results)
            image = cv2.flip(image, 1)
            cv2.imshow('OpenCV Feed', image)
            if cv2.waitKey(10) & 0xFF == ord('q'):
                break
    cap.release()
    cv2.destroyAllWindows()
```

Il metodo video_position() permette all'utente di posizionarsi in modo tale che mediapipe riesca a riconoscere tutti i punti nel modo corretto.

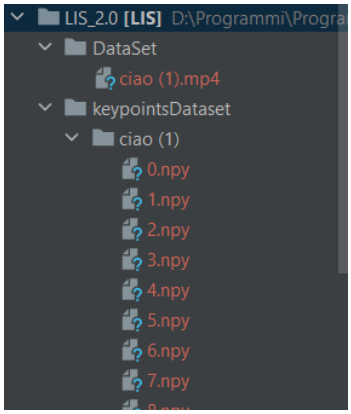
```python
def register_video():
    cap = cv2.VideoCapture(0)
    fourcc = cv2.VideoWriter_fourcc('m', 'p', '4', 'v')
    for segno in segni:
        ready = False
        while ready == False:
            print(segno)
            ret, frame = cap.read()
            frame = cv2.flip(frame, 1)
            cv2.putText(frame, 'PREPARATI PER {} E PREMI Q'.format(segno), (120, 200), cv2.FONT_HERSHEY_SIMPLEX, 1, (0, 255, 0), 4, cv2.LINE_AA)
            cv2.imshow('OpenCV Feed', frame)
            if cv2.waitKey(10) & 0xFF == ord('q'):
                ready = True
        for video in range(n_video):
            filename = str(segno) + " " + "(" + str(video) + ")" + '.mp4'
            videopath = os.path.join('DataSet', filename)
            out = cv2.VideoWriter(videopath, fourcc, 30, (640, 480))
            for frame_num in range(frame_video+1):                #perde un frame nel salvataggio e quindi ne registro 31
                ret, frame = cap.read()
                frame = cv2.flip(frame, 1)
                if frame_num == 0:
                    cv2.putText(frame, 'STARTING COLLECTION', (120, 200),
                                cv2.FONT_HERSHEY_SIMPLEX, 1, (0, 255, 0), 4, cv2.LINE_AA)
                    cv2.putText(frame, 'Collecting frames for {} Video Number {}'.format(segno, video),
                                (15, 12),
                                cv2.FONT_HERSHEY_SIMPLEX, 0.75, (0, 0, 255), 1, cv2.LINE_AA)
                    cv2.imshow('OpenCV Feed', frame)
                    cv2.waitKey(2000)
                else:
                    cv2.imshow('OpenCV Feed', frame)
                    out.write(frame)

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

Il metodo register_video() permette la registrazione dei video in modo sequenziale segnalando all'utente il segno che deve replicare. I video registrati sono composti da 30 frame.

```python
def extract_keypoints_dataset():
    videoDataSetPath = str(os.getcwd()) + "\DataSet"
    videoList = os.listdir(videoDataSetPath)
    for video in videoList:
        videoPath = os.path.join("DataSet", str(video))
        cap = cv2.VideoCapture(videoPath)
        print(video)
        videokeypath = os.path.join('keypointsDataset', str(video))
        os.mkdir(videokeypath)
        i_keypoints = 0
        with ddc.mp_holistic.Holistic(min_detection_confidence=0.5, min_tracking_confidence=0.5) as holistic:
            for frame_num in range(frame_video):
                ret, frame = cap.read()
                image, results = ddc.mediapipe_detection(frame, holistic)
                ddc.draw_styled_landmarks(image, results)
                cv2.imshow('OpenCV Feed', frame)
                keypoints = ddc.extract_keypoints(results)
                keypointspath = os.path.join(videokeypath, str(i_keypoints))
                np.save(keypointspath, keypoints)
                i_keypoints = i_keypoints+1
                if cv2.waitKey(10) & 0xFF == ord('q'):
                    break
        cap.release()
        cv2.destroyAllWindows()
```

Il metodo extract_keypoints_dataset() è l'ultimo eseguito. Si occupa di estrarre da ogni frame dei video registrati, presenti nella cartella "DataSet", i keypoints riconosciuti da mediapipe. I keypoints sono organizzati in cartelle con lo stesso nome del video a cui appartengono.

Le cartelle del dataset sono strutturate come segue:



actionDetection_helper.py

Questo script aiuta nella registrazione del dataset e nel riconoscimento dei segni.

```
def mediapipe_detection(image, model):
    results = model.process(image)  # Make prediction
    return image, results
```

Il metodo mediapipe_detection() analizza la posizione di faccia, postura, mano destra, mano sinistra del frame passato (image) secondo il modello olistico e ne restituisce il risultato.

```
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_landmarks else np.zeros(33*4)
    face = np.array([[res.x, res.y, res.z] for res in results.face_landmarks.landmark]).flatten() if results.face_landmarks else np.zeros(468*3)
    lh = np.array([[res.x, res.y, res.z] for res in results.left_hand_landmarks.landmark]).flatten() if results.left_hand_landmarks else np.zeros(21*3)
    rh = np.array([[res.x, res.y, res.z] for res in results.right_hand_landmarks.landmark]).flatten() if results.right_hand_landmarks else np.zeros(21*3)
    return np.concatenate([pose, face, lh, rh])
```

Il metodo extract_keypoints() analizza i risultati ottenuti dal metodo mediapipe_detection() e li elabora in modo tale da poterli conservare in un array.

```python
def draw_styled_landmarks(image, results):
    # Draw face connections
    mp_drawing.draw_landmarks(image, results.face_landmarks, mp_holistic.FACEMESH_CONTOURS,
                              mp_drawing.DrawingSpec(color=(80,110,10), thickness=1, circle_radius=1),
                              mp_drawing.DrawingSpec(color=(80,256,121), thickness=1, circle_radius=1)
                              )
    # Draw pose connections
    mp_drawing.draw_landmarks(image, results.pose_landmarks, mp_holistic.POSE_CONNECTIONS,
                              mp_drawing.DrawingSpec(color=(80,22,10), thickness=2, circle_radius=4),
                              mp_drawing.DrawingSpec(color=(80,44,121), thickness=2, circle_radius=2)
                              )
    # Draw left hand connections
    mp_drawing.draw_landmarks(image, results.left_hand_landmarks, mp_holistic.HAND_CONNECTIONS,
                              mp_drawing.DrawingSpec(color=(121,22,76), thickness=2, circle_radius=4),
                              mp_drawing.DrawingSpec(color=(121,44,250), thickness=2, circle_radius=2)
                              )
    # Draw right hand connections
    mp_drawing.draw_landmarks(image, results.right_hand_landmarks, mp_holistic.HAND_CONNECTIONS,
                              mp_drawing.DrawingSpec(color=(245,117,66), thickness=2, circle_radius=4),
                              mp_drawing.DrawingSpec(color=(245,66,230), thickness=2, circle_radius=2)
                              )
```

Il metodo draw_styled_landmarks() si occupa di visualizzare a schermo i keypoints con colori diversi in base alla faccia, postura, mano destra, mano sinistra.

Modello neurale

Lo script create_model.py si occupa della creazione del modello neurale con l'ausilio della libreria **Tensorflow**.

create_model.py

I metodi principali sono due:

- define_label();
- create_model().

Di seguito la spiegazione dei metodi:

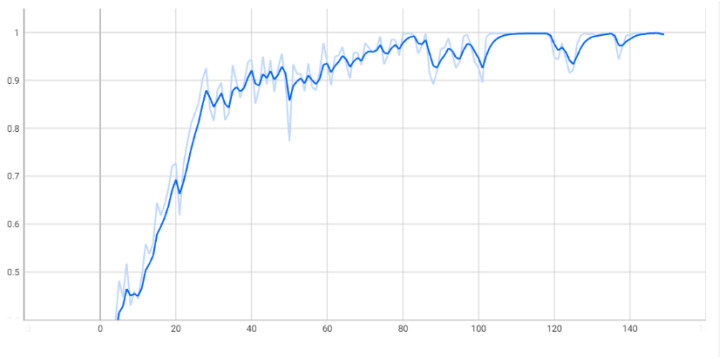
```
def define_label():
    segni.sort()                        #ordine l'array per rispettare l'ordine alfabetico del dataset
    label_map = {label:num for num, label in enumerate(segni)}
    pp.pprint(label_map)
    keypointsDataSetPath = str(os.getcwd()) + "\keypointsDataset"
    keypointsList = os.listdir(keypointsDataSetPath)
    sequences, labels = [], []
    for keypointFolder in keypointsList:
        window = []
        for frame_num in range(frame_video):
            keypoint = np.load(os.path.join(keypointsDataSetPath, keypointFolder, "{}.npy".format(frame_num)))
            window.append(keypoint)
        sequences.append(window)
    for segno in segni:
        for i in range(n_video):
            labels.append(label_map[segno])
    X = np.array(sequences)
    y = to_categorical(labels).astype(int)  #to_categorical converte un vettore di interi in una matrice binaria
    global x_train, x_test, y_train, y_test
    x_train, x_test, y_train, y_test = train_test_split(X, y, test_size=0.05)
```

Il metodo define_label() si occupa del preprocessing dei dati: assegna la label adatta ad ogni insieme di keypoints, che fanno parte dello stesso video. Successivamente divide i dati elaborati in dati di training e dati di testing.

```python
def create_model():
    model = Sequential()
    model.add(LSTM(256, return_sequences=True, activation='tanh', input_shape=(30, 1662)))
    model.add(LSTM(64, return_sequences=False, activation='tanh'))
    model.add(Dense(64, activation='relu'))
    model.add(Dense(32, activation='relu'))
    model.add(Dense(segni.shape[0], activation='softmax')) #3 neural units
    model.compile(optimizer='Adam', loss='categorical_crossentropy', metrics=['categorical_accuracy'])
    model.fit(x_train, y_train, epochs=150, callbacks=[tb_callback])
    pp.pprint(model.summary())
    return model
```

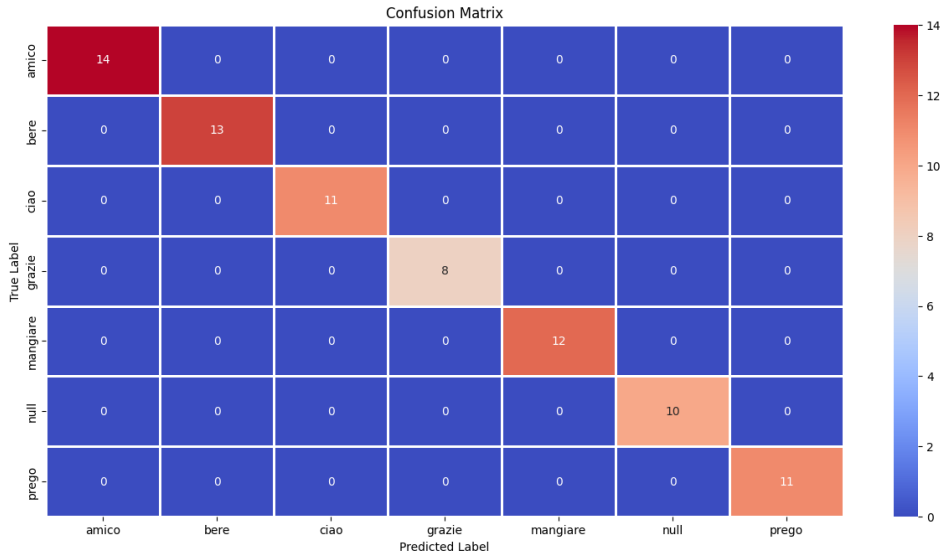
Il metodo create_model() si occupa della creazione del modello neurale.
Il modello in questione è un modello sequenziale composto da 5 livelli, di cui due di tipo **LSTM** (Long Short-Term Memory) utili per l'action detection e tre di tipo **Dense**.

Dopo diversi tentativi, quella nell'immagine sopra riportata è la configurazione più performante.
Il modello creato presenta le seguenti **statistiche**:
Epoch Categorical Accuracy: rappresenta la precisione per ogni categoria (nel nostro caso per ogni segno) in base all'allenamento per ogni epoca.
La percentuale di precisione finale, quindi, è di **0.99** circa.



Epoch Loss: rappresenta l'imprecisione del modello neurale nel riconoscimento.
La percentuale di perdita finale è di **0.01**.

Matrice di confusione:



Per creare la matrice è stato preso in considerazione il **15%** del dataset, cioè 79 set di keypoints relativi ai video.

Rappresentazione tabellare della matrice di confusione:

Classe	**Precision**	**Recall**	**f1-score**	**Support**
Amico	1.00	1.00	1.00	14
Bere	1.00	1.00	1.00	13
Ciao	1.00	1.00	1.00	11
Grazie	1.00	1.00	1.00	8
Mangiare	1.00	1.00	1.00	12
Null	1.00	1.00	1.00	10
Prego	1.00	1.00	1.00	11

Riepilogo della tabella sopraindicata:

	Precision	**Recall**	**f1-score**	**Support**
Accuracy			1.00	79
Macro AVG	1.00	1.00	1.00	79
Weighted AVG	1.00	1.00	1.00	79

I risultati ottenuti potrebbero derivare dalla moltiplicazione di alcuni video del dataset.

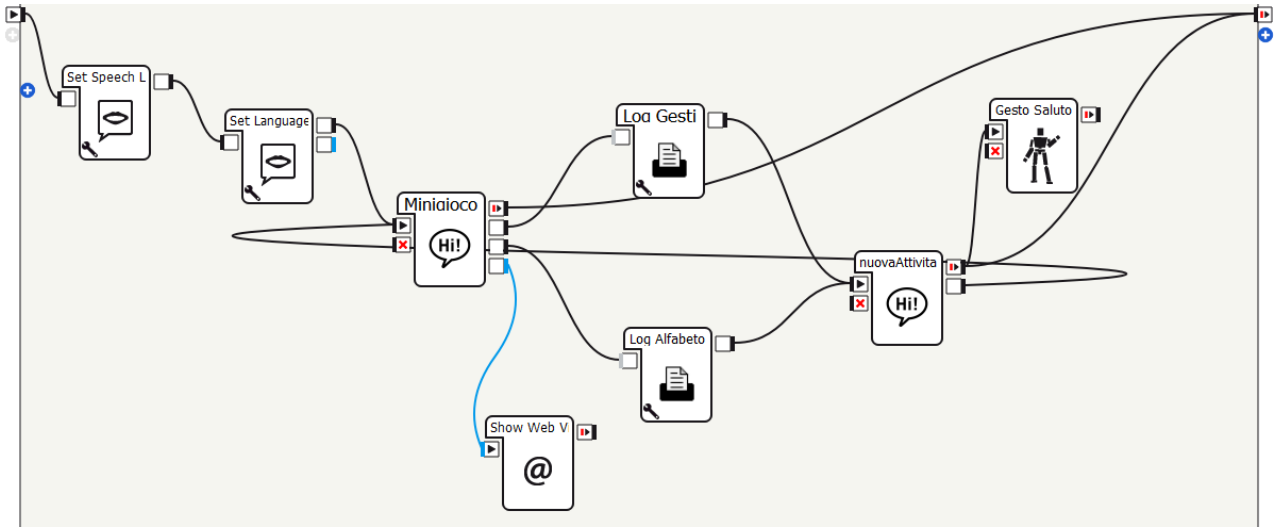
Realizzazione prodotto finale

Per l'applicazione del modello precedentemente creato, inizialmente si è pensato all'utilizzo del robot umanoide Pepper. Successivamente a causa di diversi fattori, il gruppo ha optato per la realizzazione di una Web App.

Pepper

Per l'integrazione del modello neurale in Pepper è stato utilizzato Choregraphe con l'ausilio di script Python.

Di seguito il progetto di Choregraphe:



Il progetto riportato presenta un minigioco interattivo tra Pepper e l'utente. Inizialmente Pepper, tramite il dialog "Minigioco", comunica con l'utente mostrando contemporaneamente sul suo tablet integrato i minigiochi disponibili.
I minigiochi pensati si comportano nel seguente modo: mostra la gif del segno all'utente il quale dovrà replicarla. Dopo un feedback positivo da Pepper l'utente potrà proseguire con il segno successivo o abbandonare l'attività e tornare al menu.

Il progetto con Pepper ha incontrato problemi durante lo sviluppo a causa di un conflitto di versioni tra le librerie Mediapipe e Naoqi: Mediapipe non supporta versioni di Python inferiori a 3.0 e Naoqi non supporta versioni di Python superiori alla 2.7.
Quindi, per integrare gli script di riconoscimento si è pensato di connettere Pepper ad uno script Python 2.7 secondo il paradigma Client-Server passando la stream video con la libreria Naoqi; successivamente si sarebbe utilizzato **SocketIO** per collegare lo script Python 2.7 allo script di riconoscimento avente la versione di Python 3.10.

Di seguito lo script della connessione in Python 2.7:

```
@sio.event
def connect():
    print("connected")

@sio.event
def my_message(video):
    sio.emit(str(video))

@sio.event
def disconnect():
    print("disconnected")

def video():
    videoDevice = ALProxy('ALVideoDevice')
    AL_kTopCamera = vision_definitions.kTopCamera
    AL_kQVGA = vision_definitions.kQVGA
    AL_kBGRColorSpace = vision_definitions.kRGBColorSpace
    captureDevice = videoDevice.subscribeCamera("test", AL_kTopCamera, AL_kQVGA, AL_kBGRColorSpace, 10)
    captureDevice = cv2.VideoCapture(0)
    return captureDevice



if __name__ == "__main__":
    yBroker = ALBroker("myBroker", "0.0.0.0", 80, "192.168.191.245", 9559)
    sio.connect('http://127.0.0.1:3000/')
    video = video()
    my_message(video)
```

Il progetto con Pepper non è stato portato a termine a causa della complessità del problema, ma soprattutto nella difficoltà del testing in quanto era necessaria la presenza costante di Pepper.

Tutti i file relativi allo sviluppo, sebbene incompleto, riguardante Pepper sono disponibili insieme al resto dei file di progetto nella cartella “Pepper”.

Web App

Come soluzione alternativa al progetto con Pepper si è pensato di realizzare una web app con l'utilizzo di **Flask**.
Il **funzionamento** del **minigioco** e dell'interazione con l'utente è rimasto pressoché **invariato**: verrà mostrata una gif di un segno che l'utente dovrà memorizzare e replicare nella pagina successiva, con la possibilità di ripetere l'attività dopo un feedback positivo. L'utente potrà tornare al menu in qualsiasi momento.
L'esperienza è accompagnata da un sintetizzatore vocale offerto dall'interfaccia SpeechSynthesisUtterance della Web Speech API che comunicherà all'utente alcune informazioni presenti su schermo.

Lo script che si occupa della web app è app.py.

Di seguito le route che compongono la web app:

```python
@app.route('/index')
@app.route('/')
def index():
    session["counter"] = 0
    return Response(stream_with_context(render_template('index.html')))


# Pagina in cui viene mostrata la gif del segno da memorizzare
Vito
@app.route('/gif_segni')
def gif_segni():
    global path_gifname, name_gif
    path_gifname, name_gif = randgif()
    session["isRecognized"] = False
    return stream_template("gif_segni.html", sign_gif=path_gifname, name_gif=name_gif)


# Pagina minigioco
Vito
@app.route('/minigioco_segni')
def minigioco_segni():
    return stream_template("minigioco_segni.html", name_gif=name_gif)


# In questo url viene eseguita solo la cam
Vito
@app.route('/video_feed')
def video_feed():
    return Response(open_camera(), mimetype='multipart/x-mixed-replace; boundary=frame')


# In questo url viene eseguita la detection del segno
Vito
@app.route('/detect_segno')
def return_detect_segno():
    return stream_with_context(detect_segni())
```

- **index**: carica il menu principale della web app dalle quale puoi selezionare un'attività. Al momento ne è disponibile solo una;
- **gif_segni**: mostra la gif che l'utente dovrà memorizzare;
- **minigiochi_segni**: mostra la video stream fornita dalla route video_feed e i risultati forniti dalla route detect_segno;
- **video_feed**: restituisce la video stream fornita dal metodo open_camera();
- **detect_segno**: restituisce i risultati del riconoscimento dei segni eseguito dal metodo detect_segni().

Di seguito i metodi del minigioco:

```
def open_camera():
    while camera.isOpened():
        ret, frame = camera.read()
        frame = cv2.flip(frame, 1)
        ret, buffer = cv2.imencode('.jpg', frame)
        frame = buffer.tobytes()
        yield (b'--frame\r\n'
               b'Content-Type: image/jpeg\r\n\r\n' + frame + b'\r\n')
```

Il metodo open_camera() utilizza la libreria opencv per Python per acquisire la video stream.

```
def detect_segni():
    model = tf.keras.models.load_model("model_segni.h5")
    segni.sort()
    sequence = []
    last = ''
    with ddc.mp_holistic.Holistic(min_detection_confidence=0.5, min_tracking_confidence=0.5) as holistic:
        while not session["isRecognized"]:

            ret, frame = camera.read()
            frame = cv2.flip(frame, 1)
            image, results = ddc.mediapipe_detection(frame, holistic)

            keypoints = ddc.extract_keypoints(results)
            sequence.append(keypoints)
            sequence = sequence[-30:]

            if len(sequence) == 30:
                res = model.predict(np.expand_dims(sequence, axis=0))[0]
                detected = segni[np.argmax(res)]
                print("detected: " + detected)
                print("gif: " + name_gif)
                if not session["isRecognized"]:
                    if detected != last:
                        last = detected
                        yield "Riprova! "
                    if name_gif == detected:
                        session["isRecognized"] = True
                        session["counter"] = session.get("counter") + 1
                        yield "Corretto! "
```

Il metodo detect_segni() si occupa del riconoscimento dei segni con l'utilizzo del modello neurale precedentemente creato e fornisce all'HTML un feedback.

Conclusioni e possibili sviluppi futuri

Sebbene il **modello neurale** sia stato creato sulla base un dataset non professionale e non molto vario, ha comunque mostrato delle performance soddisfacenti.
In futuro si progetta di ampliare il dataset con nuovi segni e di qualità maggiore, in modo tale da poter offrire un'esperienza più precisa e varia.

Al momento la **web app** presenta un solo minigioco funzionante basato sul riconoscimento dei segni.
In futuro si potrebbero realizzare altri minigiochi, con dinamiche differenti, ma utilizzando lo stesso dataset e modello neurale. In più aggiungendo al dataset i segni dell'alfabeto LIS, si potranno realizzare attività basate su di esso. Grazie all'utilizzo di Flask, l'implementazione di nuove attività non comporterebbe l'impiego di molto tempo.