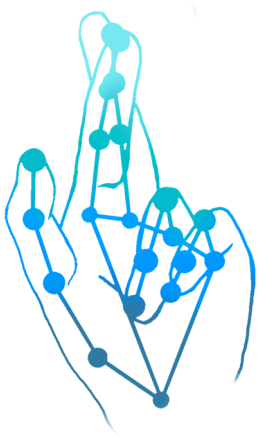
RAFT



Real-Time Automated FingerSpelling Translator

Testo

Alla fine del secondo quadrimestre abbiamo svolto un corso su come usare un raspberry PI , soffermandoci in particolare sulla parte web.

Ci è stato poi chiesto di pensare ad un progetto sulle nozioni base che ci erano state insegnate ed abbiamo deciso di realizzare un sito che, grazie ad una videocamera, possa riconoscere l’alfabeto dattilologico e tradurlo in testo.

Analisi del contesto

A livello globale si stima che le persone sordomute siano circa 70 milioni. Molti di loro affrontano sfide significative nella comunicazione quotidiana, specialmente con coloro che non conoscono la lingua dei segni. Come ben sappiamo, per di più, in alcuni paesi poche scuole offrono corsi di lingua dei segni, e la disponibilità di interpreti qualificati è spesso insufficiente.

Abbiamo pensato che un’app che utilizza l'intelligenza artificiale (IA) per tradurre l'alfabeto dattilologico (finger spelling) potrebbe offrire una soluzione innovativa per superare alcune delle barriere di comunicazione affrontate dalle persone non parlanti.

I vantaggi sarebbero molti, ad esempio l’app sarebbe accessibile a chiunque abbia uno smartphone ed il suo utilizzo potrebbe essere un aiuto alle persone udenti per imparare l'alfabeto dattilologico e, quindi, a migliorare la loro capacità di comunicare con le persone non parlanti.

Ipotesi

- Il progetto necessitava di una webcam con una mediocra risoluzione per distinguere i dettagli nella mano.
- Avevamo bisogno di una macchina abbastanza potente per il training dell'intelligenza.
- Il raspberry doveva avere abbastanza memoria per supportare l'intelligenza.

Obiettivi

Quando abbiamo avuto l'idea siamo partiti in grande pensando addirittura ad un'applicazione mobile, ma poi, a causa di tempo e risorse, siamo passati a degli obiettivi minori.

Cosa doveva includere il progetto finale:

- Un sito web hostato su un raspberry in localHost user-friendly, per poter semplificare l'utilizzo dell'intelligenza artificiale e da usare come mezzo di distribuzione di quest'ultima. Questo sarebbe stato accessibile da qualsiasi altro dispositivo all'interno dalla stessa rete del raspberry.
- Un'intelligenza artificiale che riconoscesse i 26 simboli dell'alfabeto dattilologico.

Implementazioni del FrontEnd

- Un interfaccia user-friendly.
- Streaming della telecamera collegata al raspberry (opzionalmente far anche vedere il disegno dei punti della mano rilevati in real-time).
- Casella di testo in cui veniva creata una stringa di senso compiuto in base alle lettere tradotte dall'intelligenza. In questa casella sarebbe anche stato possibile scrivere da tastiera, cancellare ecc…
- Tasto per scaricare in un file .txt il testo scritto nella casella.
- Un tasto per copiare negli appunti il testo scritto nella casella.
- Tre radioButton che permettono di scegliere la modalità di funzionamento dell'AI. Le tre modalità sono rispettivamente: utilizzare la telecamera collegata al raspberry, utilizzare la telecamera del dispositivo da cui si accede al sito e caricare un file .png da far tradurre all'AI.

Implementazioni del backEnd

- Un codice che riuscisse a hostare il sito in localhost.
- Un codice che fosse in grado di streammare la videocamera sul sito.
- Un codice che fosse in grado di inviare i risultati estrapolati dall'intelligenza al sito in realtime.

Implementazioni dell’intelligenza artificiale

- Un'intelligenza che riconoscesse tutte le 26 lettere dell’alfabeto.
- Un codice di allenamento per il nostro DataSet.
- 3 segni speciali per poter cancellare, fare uno spazio ed un segno “null” che serve puramente al motore e non ha effetti sul FrontEnd.

Scaletta degli obiettivi da seguire durante lo sviluppo

- Trovare il codice di un AI open da poter utilizzare.
- Trovare diversi dataSet contenenti tutte le lettere necessarie, compresi i segni speciali.

Questi ultimi svolti ancora prima di iniziare il progetto, per decretare se la sua realizzazione sarebbe stata fattibile o meno.

I prossimi obbiettivi sono all’incirca in ordine di come dovrebbero essere realizzati, anche se alcuni verranno svolti in parallelo:

- Configurare e far funzionare la telecamera sul raspberry.
- Configurare l’AI e farla andare sul raspberry con un dataSet qualunque.
- Mokup del sito web.
- Realizzare il FrontEnd.
- Streaming della telecamera sul sito.
- Creare la casella di testo del sito.
- Creare la modalità di poter scaricare in un file .txt
- Creare modalità per copiare negli appunti.
- Far andare l’AI sul raspberry ed avere un output dovuto al DataSet standard.
- Creare il BackEnd per la comunicazione tra AI e sito web.
- Creare il codice d’allenamento per l’AI ed allenarla col nostro DataSet.
- Customizzare l’AI col dataSet dell’alfabeto dattilologico.
- Customizzare l’AI per dare all’output il formato previsto sul terminale.
- Streaming dell’AI sul sito al posto della semplice cam facendo vedere i punti rilevati in real time (opzionale).
- Scrivere nella casella di testo l’output corretto.

Target

Questo servizio è di mirabile attenzione per tutti coloro che hanno una condizione, fisica o neurologica che sia, che non le rende in grado di parlare, e che si ritrovano a dover comunicare con persone che non conoscono l'alfabeto dattilologico. Quest'applicazione semplificherebbe di molto le piccole conversazioni quotidiane in quanto nella nostra idea basterebbe inquadrare la persona che usa la lingua dei segni per avere una traduzione automatica istantanea del loro significato.

Strategia di soluzione

Divisione del lavoro

- *Installazione, funzionamento, customizzazione (training) dell'AI*: Melissa.
- *FrontEnd*: Martina, Salomè.
- *Grafica*: Martina.
- *BackEnd:* Melissa, Martina.
- *Installazione cam*: Melissa, Martina, Salomè.
- *Stesura della documentazione*: Melissa, Martina.

Hardware

Raspberry PI 3 - Model 3

Di seguito le specifiche del raspberry utilizzato:

- Quad Core 1.2GHz Broadcom BCM2837 64bit CPU
- 1GB RAM
- BCM43438 wireless LAN and Bluetooth Low Energy (BLE) on board
- 100 Base Ethernet
- 40-pin extended GPIO
- 4 USB 2 ports
- 4 Pole stereo output and composite video port
- Full size HDMI®
- CSI camera port for connecting a Raspberry Pi camera
- DSI display port for connecting a Raspberry Pi touchscreen display
- Micro SD port for loading your operating system and storing data
- Upgraded switched Micro USB power source up to 2.5A

USB cam

Specifiche della USB cam utilizzata:

- Modello: XPRESS CAM 300 (usb 2.0)
- Marca: NGS
- Risoluzione di acquisizione video: 5 Mpx
- Risoluzione dell'immagine: 8Mpx

Software

Linguaggio

- *HTML:* abbiamo utilizzato il linguaggio di markup HTML per la realizzazione del sito web
- *CSS*: utilizzato per le grafiche e lo stile del sito
- *Javascrip*t: utilizzato per l'invio di richieste al backend e funzioni dinamiche del sito
- *Python*: contiene l'API dell'intelligenza e le funzioni per l'hosting del sito

Servizi utilizzati

Flask

- framework web scritto in python utilizzato per lo sviluppo dell'applicazione web. Usato per creare un server web leggero e per esporre l'API dell'intelligenza ad altri dispositivi e applicazioni che comunicano con il raspberry. In particolare abbiamo utilizzato la libreria Socket.IO per permettere la comunicazione in tempo reale tra client e server.

MediaPipe

- Libreria utilizzata per l'implementazione di algoritmi di visione artificiale in progetti come il nostro. Permette l'analisi e l'interpretazione di immagini e video in tempo reale. Può riconoscere diversi soggetti, dai volti, al corpo, fino alle mani.

OpenCv (Open Source Computer Vision Library)

- Libreria scritta in C++ per la visione artificiale,l'elaborazione delle immagini, riconoscimento di oggetti e persone. L'abbiamo utilizzata insieme alla libreria MediaPipe.

Motion

- Software di monitoraggio video che rileva il movimento attraverso l’analisi dei flussi video della telecamera collegata. Lo abbiamo inizialmente utilizzato per streammare le catture della telecamera sul sito web, purtroppo quando ci siamo resi conto che l’utilizzo della telecamera di motion andava in contrasto con l’utilizzo della telecamera dell’intelligenza, abbiamo abbandonato l’idea di questa libreria.

Google Colab

- Google Colab è un ambiente di sviluppo basato su Jupyter Notebooks, viene utilizzato per il machine learning. Quest'ultimo comprende una serie di librerie, tra cui TensorFlow.

TensorFlow

- Libreria Open.source sviluppata da Google per l’apprendimento automatico ed il deep learning. Permette di costruire ed addestrare rete neurali, supporta operazioni complesse di machine learning e offre un’ampia gamma di strumenti per la visualizzazione del debugging dei modelli.

Caratteristiche dell’intelligenza

L’intelligenza è in grado di prendere in input diversi formati tra cui:

- Immagini
- Video
- Trasmissione video real-time

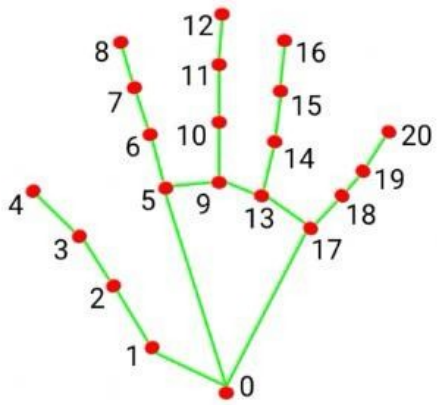
L’intelligenza da in output diversi valori tra cui:

- Categoria del gesto riconosciuto (ovvero, che gesto è stato fatto)
- Se è presente la mano destra o la mano sinistra
- Le coordinate dei punti riconosciuti della mano in base all’immagine analizzata
- Le coordinate dei punti riconosciuti della mano in base alla realtà

L’intelligenza presentava una serie di configurazioni disponibili, di seguito riportate le fondamentali che ci sono servite per la realizzazione del progetto:

- *running_mode*: quale tipo di input avere
- *num_hands* : numero massimo di mani da analizzare in un immagine (selezionabili una o due)
- *min_hand_detection_confidence*: il minimo punteggio di precisione che un gesto deve ottenere per poter essere classificato come tale
- *min_hand_presence_confidence*: minimo punteggio di precisione necessario per accertare che sia presente una mano nell’immagine

È possibile customizzare l’intelligenza ed allenarla su un dataSet personale per farle riconoscere gesti differenti, ma il modello per il riconoscimento dei punti di riferimento della mano e il modello per il riconoscimento del palmo rimarrà sempre il medesimo.



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

Il modello di rilevamento del palmo, localizza la regione della mano nell’intera immagine di output, dopodiché il modello di rilevamento dei punti di riferimento localizza tutte le 21 coordinate mostrate in figura. Solo a questo punto entra in gioco la customizzazione, dove l’intelligenza si basa sulla posizione dei punti di riferimento rilevati e cerca di associarli ad uno dei gesti che abbiamo allenato a riconoscere col nostro dataSet.

L’intelligenza è quindi in grado di distinguere:

- Se ha rilevato un gesto, che gesto ha rilevato
- Se ha rilevato una mano, ma nessuno dei gesti insegnatogli
- Se non ha rilevato una mano

Neuroni di input:

- Gli input si basano sui punti di riferimento della mano catturata nell’immagine. Ogni punto di riferimento ha coordinate (x, y, z). Essendo che il modello utilizza 21 punti di riferimento della mano, avrà 21*3 neuroni di input, ovvero 63

Neuroni di output:

- I neuroni di output corrispondono al numero dei gesti che il modello può riconoscere (nel nostro caso 29). Per ogni classe viene associata una percentuale di probabilità che il gesto riconosciuto sia quello appartenente a quella classe. Ovviamente nelle classi è anche presente la classe “null”, ovvero “nessun gesto riconosciuto”

Implementazione

Configurazioni

Raspberry PI 3 VS Raspberry PI 4

Quando abbiamo iniziato il progetto, non avevamo un raspberry a disposizione; ci eravamo sempre esercitati con i dispositivi che ci erano stati forniti dalla scuola, ma con il bisogno di un lavoro anche oltre alle ore scolastiche, abbiamo dovuto procurarcene uno.

La parte di configurazione l’abbiamo svolta su un raspberry PI 4, ma l’unico dispositivo che potevamo tenere anche nelle ora extrascolastiche era un raspberry PI 3; ciò ci ha causato diverse difficoltà.

Nonostante una schedina SD da 8GB fosse sufficiente per far funzionare il sistema operativo, abbiamo optato per una memoria più ampia, da 32GB, questo perché abbiamo immaginato che il motore dell’intelligenza artificiale avrebbe occupato molto spazio.

Nella nostra memoria da 32GB era già stato installato il sistema operativo e configurato il protocollo SSH per la comunicazione da remoto, ma abbiamo scoperto che un sistema operativo installato utilizzando un raspberry PI 4, non fosse compatibile con un raspberry PI 3.

Assodato ciò abbiamo resettato la schedina e ripetuto le configurazioni. Da quel momento il raspberry ha funzionato tranquillamente sia usato in singolo, che tramite SSH.

Questo non vuol dire che per implementare il progetto bisogni necessariamente usare un raspberry PI 3, quest’ultimo tende a generare meno calore, riducendo la necessità di soluzioni di raffreddamento aggiuntive e alcuni software o librerie potrebbero essere più stabili o meglio supportati , ma allo stesso tempo un raspberry Pi 4 ha un processore più potente e più RAM e porte USB 3.0 che offrono una velocità di trasferimento dati molto superiore rispetto alle porte USB 2.0 del Raspberry Pi 3.

La scelta dipende sia dalle disponibilità che da quali vantaggi si preferisce avere.

RPIcam VS USB cam

Inizialmente avevamo deciso di utilizzare la RPIcam per rilevare i punti della mano, poiché il Raspberry Pi disponeva di una porta dedicata per la CSI camera. Tuttavia, a causa di vari problemi legati al fatto che la camera non venisse rilevata dall'intelligenza artificiale e quindi non riuscisse a individuare i punti necessari, siamo passati a una telecamera USB.

L'intelligenza artificiale che avevamo trovato era perfettamente in grado di leggere anche questo tipo di camera, quindi abbiamo proseguito con quest'ultima. Per poter usare al meglio questa camera abbiamo installato Cheese che ci ha permesso di testarne il funzionamento.

FrontEnd

Mockup

ll Mockup è stato inizialmente realizzato con un' interfaccia molto semplice perchè doveva rispecchiare una possibile implementazione futura come applicazione e perciò mantenere il medesimo stile. E' stato pensato per avere una piccola animazione iniziale che presentava brevemente il nostro logo e l'acronimo. Dopodichè la pagina si sarebbe aperta sull'effettiva applicazione. Essa avrebbe mostrato lo stream della fotocamera ed a lato una casella dove saremmo andati ad inserire il testo rilevato dall'AI. Un menu a discesa avrebbe permesso la scelta tra i tre tipi di input che si volevano inserire. Infine un bottone avrebbe permesso di scaricare il contenuto della casella di testo.

Ambiente

I codici python, sia sulle macchine, che su rasbian, sono stati fatti runnare all'interno di un ambiente virtuale apposito, per non andare ad intaccare il python di sistema.

Per quanto riguarda l'intelligenza ed il sito, abbiamo lavorato sull'ide di VisualStudioCode per le stesure di codice più lunghe (essendo questo software molto più comodo e veloce da usare). Andavamo poi a passare i codici sul raspberry per fare i test, e nel caso ci fosse bisogno di piccole correzioni usavamo l'IDE geany direttamente installato su rasbian.

Invece per quanto riguarda il training abbiamo provato a far runnare il codice su differenti macchine: la prima versione è stata scritta in python utilizzando sempre VisualStudioCode, ed è stata fatta partire l'elaborazione su un computer portatile, di seguito le sue specifiche:

- ***Microprocessore**: Intel® Core™ i7-1195G7*
  - *Frequenza massima: fino a 5,0 GHz con Intel® Turbo Boost*
  - *Cache: 12 MB L3*
  - *Core: 4 core, 8 thread*
- ***Chipset**: Intel® Integrated SoC*
- ***Memoria**:16 GB di RAM DDR4-3200 MHz (2 x 8 GB)*
- ***Scheda Grafica**:Intel® Iris® X^e*
- ***Unità Disco Rigido**:1 TB PCIe® NVMe™ M.2 SSD*
- ***Sistema Operativo**: Windows 11 Home*

Dopo 6 ore ed un elevatissimo consumo di batteria, a fine allenamento abbiamo notato che il codice avesse un errore, infatti il codice non era riuscito a generare l'output del formato corretto. Dati i lunghi tempi di esecuzione, abbiamo provato a far eseguire il codice (in teoria corretto) su un fisso, di seguito le sue caratteristiche:

- *AMD Ryzen 7 5800X*
- *ASUS ROG Strix B550-F Gaming WiFi II*
- *Thermalright Peerless Assassin 120 SE*
- *Corsair Vengeance LPX 16GB DDR4 3600*
- *Crucial P3 1TB M.2 PCIe Gen3 NVMe SSD*
- *Mars Gaming MC-2000 Case*
- *CORSAIR CX650 ATX 650W PSU*
- *Sapphire Pulse AMD Radeon RX 7600 XT 16GB GDDR6*
- *ARCTIC P12 Slim PWM PST 120 mm Ventola*

Questa volta il tempo impiegato è diminuito di circa due ore, ma alla fine sono stati generati comunque degli errori.

Non potevamo permetterci di aspettare ore per ogni esecuzione, alchè abbiamo deciso far runnare il codice su google colab. Google colab è un ambiente online da poter usare come IDE. Il codice ha accesso ad una serie di librerie google (molte ottime per il machine learning) ed in più, per l'esecuzione, viene prestata la potenza di calcolo dei server google. Il training ha infatti avuto bisogno di appena due minuti per essere completato, abbiamo così potuto fare molti test.

L'unico problema fu che fummo costretti a dimezzare il nostro dataSet, avendo una grande perdita di precisione. La versione gratuita di google colab non supportava il peso di tutte le immagini inizialmente necessarie.

Come avevamo creato un codice per la numerazione delle immagini nelle cartelle (vedi il punto base dati) avremmo potuto creare un codice che andasse a selezionare in automatico tot immagini randomiche da ogni classe. Abbiamo però deciso, di svolgere il processo manualmente, per un motivo principale: le immagini nel dataset erano molto diverse tra di loro, c'erano gruppi di immagini al buio, gruppi di immagini alla luce, gruppi con mani di pelle chiara e gruppi di immagini con pelle scura ecc… una macchina non avrebbe mai potuto capire questa differenza, avevamo bisogno di una mente umana che scegliesse un numero equo di immagini di ogni tipo.

Index e style

Il sito è strutturato su uno scheletro HTML composto da un main, un header e un altro main. Questo perché, per come è stato concepito, era necessario avere un main su cui applicare un effetto "vedo-non vedo" al primo collegamento con il nome del progetto e il logo. Successivamente, nel secondo main, si trova la pagina effettiva. In questa seconda pagina è presente un header per visualizzare le informazioni che saranno visibili in ogni futura pagina del sito, e un main con lo spazio dedicato allo streaming della videocamera e una casella di testo dove verrà visualizzato il testo rilevato dall'intelligenza artificiale. Inoltre, sono stati inseriti una serie di div per creare un menu che permetterà in futuro di visualizzare diverse pagine.

Per quanto riguarda lo stile della pagina, sono stati applicati font aggiuntivi e sono state aggiunte delle piccole media-query per rendere la pagina minimamente responsive su PC. I colori sono stati scelti in base alla regola 60-30-10 sui toni del grigio/azzurro e abbiamo mantenuto uno stile volutamente tondeggiante con linee semplici.

BackEnd (hosting del sito)

JavaScript

Nello script sono state inserite tre funzioni principali. La prima funzione viene utilizzata al primo avvio del sito per effettuare lo scambio tra le due schermate, rendendo visibili o invisibili gli header e i main a seconda dello stato in cui si trovano. Un'altra funzione è stata aggiunta per permettere il download di un file .txt contenente le frasi rilevate dall'intelligenza artificiale. Lo script apre una connessione alla route indicata e richiede le informazioni, che poi salva nel file testo.txt . Infine, l'ultima funzione si occupa di aggiornare la casella di testo, effettuando una richiesta HTTP ogni N. secondi a un server Python per ottenere la stringa generata e visualizzarla sul sito.

Python e API

Le funzioni principali dello script python si dividono in due: hosting del sito e algoritmo dell’intelligenza. Le due funzioni sono divise in due thread separati, essendo che il sito deve essere hostato in modo costante, ma anche l’intelligenza deve rilevare ogni singolo frame, hanno quindi bisogno di lavorare in parallelo.

I thread condividono la memoria, infatti la stringa calcolata dal thread intelligenza viene passata al thread hosting per poterla poi stampare sul sito web.

Per l'hosting del sito, abbiamo creato due route: la route base e un'altra rinominata 'dht' per la trasmissione della stringa correttamente strutturata. Infine, abbiamo aggiunto una route standard per servire direttamente al JavaScript richiedente i file dalla cartella 'static'.

Intelligenza artificiale

Funzionamento dell’intelligenza base

Il codice dell’intelligenza, presente come API nel python del sito web svolge le seguenti funzioni:

- Importa le librerie necessarie di mediaPipe per il riconoscimento gesti
- Cattura i frame dalla videocamera
- Scarica il modello (ovvero il file .task) per il riconoscimento e lo inizializza
- Prepara i dati in input andando a convertire il frame del video o l’immagine in un array Numpy grazie alla libreria esterna di OpenCV
- Calcola i risultati

- Scrive l’output a video e disegna i punti di riferimento trovati

Allenamento dell’intelligenza

Il codice dell’allenamento viene scritto in Python e fatto runnare sull’IDE di google colab, svolge le seguenti funzioni:

- Installa le librerie necessarie al training e le importa
- Si collega ad un link google drive in cui è stata caricata la cartella .zip del dataSet e la scarica
- Verifiche che il dataSet sia organizzato nel modo corretto e trova tutte le classi
- Traccia alcune immagini di esempio per ogni gesto
- Carica tutte le immagini del dataSet
- Allena il modello tramite funzioni specifiche
- Calcola la precisione che si è raggiunta e le perdite
- Esporta il modello nel formato specifico per le intelligenze di mediaPipe, ovvero .task

Ecco un esempio di output che viene dato durante il training:

```
Epoch 1/10
770/770 [==============================] - 4s 4ms/step - loss: 2.2920 - categorical_accuracy: 0.2890 - val_loss: 0.6036 - val_categorical_accuracy: 0.7098
Epoch 2/10
770/770 [==============================] - 5s 6ms/step - loss: 1.2396 - categorical_accuracy: 0.6000 - val_loss: 0.4230 - val_categorical_accuracy: 0.7824
Epoch 3/10
770/770 [==============================] - 3s 4ms/step - loss: 0.9647 - categorical_accuracy: 0.6643 - val_loss: 0.3810 - val_categorical_accuracy: 0.7876
Epoch 4/10
770/770 [==============================] - 3s 4ms/step - loss: 0.8397 - categorical_accuracy: 0.7000 - val_loss: 0.3358 - val_categorical_accuracy: 0.8083
Epoch 5/10
770/770 [==============================] - 3s 4ms/step - loss: 0.7558 - categorical_accuracy: 0.7286 - val_loss: 0.3154 - val_categorical_accuracy: 0.8290
Epoch 6/10
770/770 [==============================] - 4s 5ms/step - loss: 0.7092 - categorical_accuracy: 0.7468 - val_loss: 0.2921 - val_categorical_accuracy: 0.8549
Epoch 7/10
770/770 [==============================] - 3s 4ms/step - loss: 0.6739 - categorical_accuracy: 0.7552 - val_loss: 0.2896 - val_categorical_accuracy: 0.8497
Epoch 8/10
770/770 [==============================] - 4s 5ms/step - loss: 0.6434 - categorical_accuracy: 0.7643 - val_loss: 0.2733 - val_categorical_accuracy: 0.8601
Epoch 9/10
770/770 [==============================] - 3s 4ms/step - loss: 0.6167 - categorical_accuracy: 0.7747 - val_loss: 0.2662 - val_categorical_accuracy: 0.8653
Epoch 10/10
770/770 [==============================] - 3s 4ms/step - loss: 0.6108 - categorical_accuracy: 0.7747 - val_loss: 0.2620 - val_categorical_accuracy: 0.8653
Modello allenato
Inizio valutazione del modello
193/193 [==============================] - 1s 2ms/step - loss: 0.2214 - categorical_accuracy: 0.8394
Test loss: 0.22136135399341583, Test accuracy: 0.8393782377243042
```

Abbiamo stabilito 10 epoch totali per l’allenamento, il completamento di ogni epoch rappresenta il passaggio di tutto il dataSet all’interno dell’algoritmo, questo vuol dire che noi facciamo passare il dataSet per 10 volte. Ogni epoch viene suddivisa in tante piccole parti, nel nostro caso 770. Il numero di fianco a “loss” sta ad indicare quanti sbagli sono stati fatti, e quindi quanta perdita c’è stata in quella epoch, se l’allenamento va a buon fine, la perdita deve diminuire più cicli vengono fatti. Il numero vicino ad “accuracy” sta invece ad indicare quanto la nostra intelligenza è stata precisa, questo numero dovrebbe salire al salire delle epoch. Più epoch vengono fatte e più la nostra intelligenza sarà precisa, ma sempre basandoci sul dataSet fornitogli. In sintesi, potrebbe anche avere una precisione del 100% riconoscendo solamente le immagini del dataSet, ma se quest’ultimo ha troppe poche immagini o ha una poche varietà, il risultato finale sarà comunque abbastanza scarso.

Customizzazione

Abbiamo leggermente semplificato il codice originale andando a togliere tutti le opzioni di configurazione da all'invio dell'intelligenza ed andando a settarle noi di default dentro al codice, in modo tale da togliere pesantezza a quest'ultimo. In ogni caso le configurazioni sarebbero state standard e l'utente non avrebbe potuto sceglierle.

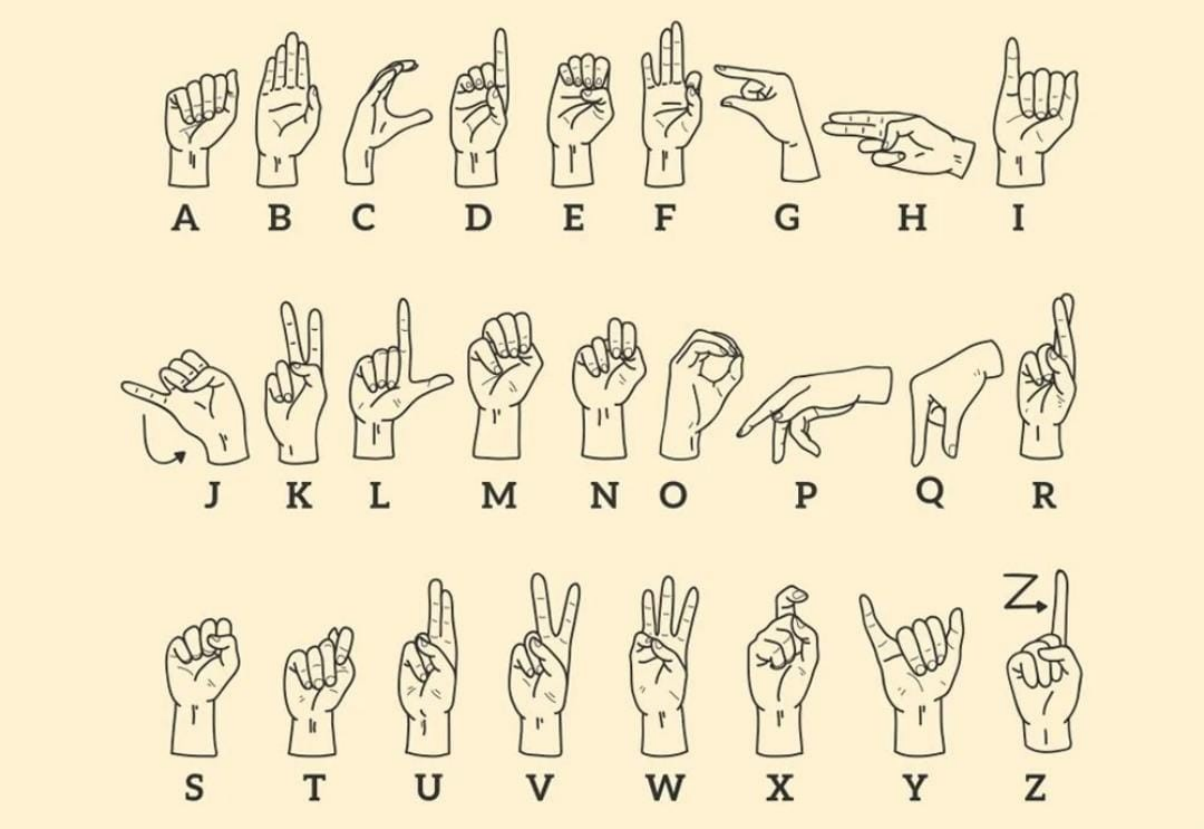
Una funzione principale che abbiamo aggiunto al codice è quella per modificare il formato dell'output: l'intelligenza, ad ogni ciclo, andava ad inserire in una variabile, la lettera rilevata. Ovviamente, andando ad analizzare ogni frame, avevamo un output per ogni frame, quindi anche più di uno al secondo.

Abbiamo dovuto implementare una funzione che andasse a creare una stringa di senso compiuto, ove se viene mantenuto lo stesso gesto costantemente, la lettera viene scritta una volta sola. La funzione implementa anche la logica dello space e del delete. Per quanto riguarda le doppie, basta lasciare anche un solo secondo di "vuoto" e ripetere il gesto precedente, in questo modo verranno stampate due lettere uguali di fila nella stringa.

Base dati

DataSet utilizzati

Per allenare l'intelligenza avevamo bisogno di migliaia di immagini di mani che facessero i segni dell'alfabeto dattilologico.



Prima ancora di iniziare il progetto abbiamo trovato un dataSet iniziale da 2.5GB. Aveva 24 classi, una per ogni lettera dell'alfabeto (meno la J e la Z, che bisognerebbe rappresentarle tramite un semplice movimento). Ogni classe aveva al suo interno circa 1500 immagini.

Come spiegato in precedenza, abbiamo dovuto diminuire la quantità di immagini per ogni classe e dopo il primo allenamento, essendo già la qualità delle foto abbastanza bassa, ci siamo resi conto che il rilevamento risultava molto impreciso.

Abbiamo quindi deciso di cambiare dataSet, mantenendo lo stesso numero di immagini, ma avendo una qualità migliore. Grazie al secondo dataSet trovato, dal peso di 5GB con 5mila immagini per classe (link in fondo), abbiamo sia potuto implementare le lettere che prima erano state escluse (effettivamente il movimento necessario per produrle era minimo, quindi erano facilmente comprensibili anche mantenendo la mano ferma), in più sono stati aggiunti due gesti (del e space) per poter implementare la funzionalità dello spazio e di delete.

Dopo aver trovato le immagini (questo procedimento lo abbiamo svolto con entrambi i dataSet per poter allenare l'intelligenza) è stato necessario organizzare le cartelle in modo intelligente e compatibile con il codice di allenamento (derivato soprattutto dalle funzionalità dell'intelligenza).

A differenza di molti training, non abbiamo diviso il dataSet in un "train" ed un "test", ma semplicemente avevamo 29 cartelle, una per classe, con dentro le rispettive immagini.

Come detto in precedenza, la scrematura del dataSet è stata fatta a mano, mentre per l'organizzazione della numerazione abbiamo usato un codice python che andasse a rinominare ogni immagine con un numero che partisse da 1 ed arrivasse fino all'ultima immagine.

Distribuzione

Nei target sopra elencati, abbiamo inizialmente pensato che le persone interessate a questa applicazione fossero coloro che, in un breve periodo, hanno a che fare con persone non parlanti e che quindi non conoscono l'alfabeto dattilologico. Per la distribuzione, abbiamo ritenuto che le priorità per far conoscere questo prodotto fossero due:

1. Che fosse facile da trovare, quindi richiedere un IP pubblico e spostarsi dalla rete locale per mettere il sito online o convertire il sito in una vera e propria applicazione scaricabile su telefono. Siamo partiti da qui per l'idea dell'acronimo RAFT, che è la spiegazione più semplice di ciò che fa l'applicazione, "real-time automated fingerspelling translator", per facilitarne la ricerca in rete.

2. Il secondo proposito è quello di far conoscere il servizio innanzitutto alle persone non parlanti, poiché sono loro a trovarsi più frequentemente di fronte a difficoltà di comunicazione rispetto alle persone comuni. Per perseguire questo obiettivo, si potrebbero contattare centri specializzati e far conoscere il servizio a loro.

Valutazione

Di seguito spiegato come abbiamo proceduto nelle varie fasi del progetto:

- La parte di frontEnd viene prima sviluppata sui nostri dispositivi e controllato il corretto funzionamento senza hostare il sito, solo al corretto funzionamento si passerà il codice sul raspberry.
- Viene installata e testata l'intelligenza senza andare a modificare il codice, solo al corretto funzionamento si inizieranno le customizzazioni.
- Viene creato un backend, e quindi un hosting, da far partire insieme all'intelligenza usando un sito completamente vuoto, senza parte grafica.
- viene customizzata l'intelligenza senza collegarla al backend, al corretto funzionamento verrà poi unita a quest'ultimo.
- La funzione per il corretto formato dell'output viene testata con delle stringhe inserite da terminale, al corretto funzionamento potrà allora prende in input gli output dell'intelligenza.
- Quando il backEnd ed il frontEnd funzioneranno correttamente, allora si potrà andare ad unirli direttamente sul raspberry.
- Ultimo testing per la correzione di eventuali piccoli bug.

Come testare l'intelligenza:

- L'intelligenza è in grado di riconoscere tutti i gesti inseriti nelle classi
- Provare a ruotare la mano (nei limiti del possibile, perchè se si cambia di troppo l'angolazione, si potrebbe anche cambiare lettera)
- Avere illuminazioni differenti
- Avere sfondi differenti
- Avere colori differenti

Come testare il frontEnd:

- Controllare il sito su più dispostivi per verificare se è responsive
- Verificare che tutte le funzioni fondamentali vengano svolte correttamente (ad esempio il salvataggio della stringa in un file)

Come testare il backEnd:

- Hostare il sito diverse volte e controllare il corretto funzionamento
- Verificare che le funzioni fondamentali vengano svolte correttamente (ad esempio l'invio dei dati da motore a sito)

Sviluppi

Sviluppi sito:

- Oltre ad utilizzare la telecamera USB, si potrebbe implementare la scelta di utilizzare anche la telecamera del dispostivo collegato in quel momento al sito oppure di poter caricare un immagine, essendo che la nostra intelligenza sarebbe facilmente in grado di riconoscere anche quei formati di input. Nel sito abbiamo già implementato la parte grafica che si occupa di svolgere queste opzioni, collegata ad una funzione backend, che però al momento è vuota.
- Oltre al pulsante per scaricare il file .txt con il testo generato, si potrebbe implementare un pulsante che permette velocemente di copiare negli appunti quello che viene visualizzato nella casella di testo, così da facilitare l'azione all'utente.
- Il sito non è responsive visualizzato da mobile, un'implementazione futura sarebbe renderlo tale anche da cellulare.

Sviluppo AI:

- Per quanto riguarda l'allenamento, si potrebbe allargare di molto il dataSet per aumentare il livello di precisione di riconoscimento inserendo anche diverse gradazioni di pelle. Il dataSet sarebbe già pronto, dovremmo però avere un dispositivo con abbastanza potenza di calcolo.
- La nostra intelligenza si basa completamente sul machine learning, va quindi ad analizzare nel dettaglio ogni immagine per poter riconoscere il gesto: un'intelligenza basata più su calcoli e formule matematiche sarebbe nettamente più veloce.
- Implementare e allenare l'intelligenza anche su gesti più complessi dei caratteri singoli, arrivare fino a riconoscere delle parole.
- Dopo aver raggiunto l'obiettivo del riconoscimento parole, appoggiarsi ad un'altra intelligenza in grado di formulare delle probabili frasi a partire da un insieme di parole riconosciute.

Sviluppi hardware:

- Runnare il codice dell'intelligenza su un raspberry pi 3 rallenta di molto l'esecuzione di quest'ultima. Un dispositivo più potente potrebbe mantenere meglio l'intelligenza, non dovendo aspettare sempre circa 5 secondi per il riconoscimento di ogni gesto.

Sviluppi Back-End:

- Purtroppo, per una questione di tempo, non siamo riusciti ad implementare lo streaming della telecamera sul sito. All'inizio avevamo stremmato la telecamera, senza far partire l'intelligenza, con Motion. Quando però abbiamo fatto partire i due processi in sincrono, ci siamo resi conto che andavano in contrasto e che la telecamera poteva essere gestita o da Motion o dall'intelligenza. La soluzione sarebbe quindi stata quella di streammare direttamente l'output video dell'intelligenza, ma ci siamo concentrati sulla risoluzione di problematiche più importanti.

Materiale utilizzato

Primo dataSet utilizzato (bassa qualità, 24 classi presenti, 2.5GB)

- https://www.kaggle.com/datasets/mrgeislinger/asl-rgb-depth-fingerspelling-spelling-it-out

Secondo dataSet utilizzato (media qualità, 28 classi presenti, 5GB)

- https://www.kaggle.com/datasets/debashishsau/aslamerican-sign-language-aplhabet-dataset

Documentazione dell'intelligenza di mediaPipe

- https://ai.google.dev/edge/mediapipe/solutions/vision/gesture_recognizer

Tutorial generali per pyhton

- https://www.w3schools.com/python/default.asp

Risoluzione errori

- https://stackoverflow.com/

IDE per il training dell'intelligenza

- https://colab.research.google.com/

Configurazione del raspberry e della camera

- https://www.raspberrypi.com/documentation/

Fonte di informazioni statistiche e demografiche globali sulla popolazione sorda e muta

- https://www.bing.com/search?pc=OA1&q=World%20Federation%20of%20the%20Deaf%20statistics