UNIVERSITA' POLITECNICA DELLE MARCHE

FACOLTA' DI **INGEGNERIA**

Corso di Laurea triennale **INGEGNERIA ELETTRONICA**

STUDIO, ANALISI E TEST DI UNA RETE NEURALE CONVOLUZIONALE PER IL RICONOSCIMENTO DI HAND POSTURES SU SISTEMI EMBEDDED

STUDY, ANALYSIS AND TESTING OF A CONVOLUTIONAL NEURAL NETWORK FOR HAND POSTURES RECOGNITION ON EMBEDDED SYSTEMS

Relatore:

Prof.ssa Laura Falaschetti

Tesi di Laurea di:

Fabio Tempera

A.A. **2023 /2024**

Indice

1
Introduzione

Il mondo dell'intelligenza artificiale sta attraversando un periodo di crescita senza precedenti, paragonabile ad una vera e propria esplosione. Le sue applicazioni stanno rapidamente permeando ogni aspetto della nostra vita quotidiana, rivoluzionando sostanzialmente tutti i settori. Tra i numerosi ambiti in cui l'IA sta dimostrando il suo potenziale trasformativo, il riconoscimento di immagini si distingue per la sua rapida evoluzione e le sue promettenti prospettive future.

Questa branca dell'IA, che consente alle macchine di "vedere" e interpretare il mondo visivo in modo simile agli esseri umani, sta aprendo la strada a innovazioni rivoluzionarie. Attraverso algoritmi sofisticati e reti neurali profonde, i computer sono ora in grado di identificare oggetti, persone, luoghi e persino emozioni all'interno di immagini e video. Basti pensare alle diagnosi mediche più accurate grazie all'analisi di immagini radiologiche, alle auto a guida autonoma che navigano in sicurezza riconoscendo segnali stradali e pedoni, ai sistemi di sorveglianza intelligenti che individuano comportamenti sospetti, o alle applicazioni di realtà aumentata che arricchiscono la nostra percezione del mondo.

Il riconoscimento di immagini non si limita solo all'identificazione di elementi predefiniti. L'IA sta progredendo verso una comprensione più profonda del contesto visivo, consentendo alle macchine di analizzare le relazioni tra gli oggetti, interpretare scene complesse e persino generare descrizioni accurate di ciò che stanno osservando. Questo apre scenari futuri per i quali stiamo già avendo da tempo un assaggio, come la possibilità per i non vedenti di "vedere" il mondo attraverso descrizioni audio generate in tempo reale, o la creazione di robot in grado di interagire con l'ambiente circostante in modo più naturale e intuitivo.

La continua evoluzione di questi modelli, alimentata da una crescente disponibilità di dati e da una potenza di calcolo sempre maggiore, promette di plasmare il futuro in modi che possiamo solo iniziare a immaginare. Dalla medicina alla robotica, dall'arte all'industria, questa tecnologia sta aprendo nuove frontiere e offrendo soluzioni innovative a sfide complesse, contribuendo a creare un mondo più efficiente, sicuro e accessibile per tutti.

Alla base di ogni modello di intelligenza artificiale per il riconoscimento di immagini c'è un elemento fondamentale, il dataset, un insieme composto da una vasta moltitudine di immagini etichettate, il quale rappresenta ciò con cui l'algoritmo imparerà a riconoscere pattern, forme ed oggetti. La qualità, la quantità e l'adeguatezza di questo dataset sono cruciali per il successo del progetto, influenzando direttamente le prestazioni, l'accuratezza e l'affidabilità del modello finale.

Qui ci focalizzeremo, in particolare, nell'ambito dell'acquisizione e dell'elaborazione dati finalizzata all'implementazione di un modello di intelligenza artificiale in grado di riconoscere gestures eseguite con le mani, che trova applicazione in diverse circostanze della vita quotidiana, come ad esempio nella domotica o nell'assistenza a persone anziane o diversamente abili.

Inoltre, dopo aver creato ed elaborato il nostro dataset personalizzato, ci occuperemo di validarlo, controllando la qualità dei dati e testando la sua efficacia dopo aver terminato la fase di addestramento del modello.

2

Cenni sulle CNN

Recentemente l'utilizzo delle CNN per il riconoscimento di hand postures è ampiamente utilizzato[1, 2, 3]. Tuttavia, per comprendere le CNN, è utile partire dalle basi delle reti neurali artificiali. Una rete neurale artificiale è un modello matematico ispirato alla struttura e al funzionamento del cervello umano. È composta da strati di nodi, chiamati neuroni, che sono connessi tra loro. Ogni connessione ha un peso associato, e i neuroni stessi applicano una funzione di attivazione ai segnali in ingresso per produrre un'uscita. Questo processo di propagazione dei segnali attraverso la rete permette al modello di apprendere dai dati e di fare previsioni o classificazioni.

Le CNN sono una classe speciale di reti neurali progettate specificamente per lavorare con dati che hanno una struttura spaziale, come le immagini. A differenza delle reti neurali tradizionali, che trattano ogni input in modo indipendente, le CNN sfruttano la struttura spaziale dei dati per estrarre caratteristiche locali e costruire rappresentazioni gerarchiche. Questo approccio è particolarmente efficace per il riconoscimento di immagini, ma ha anche trovato applicazione in molti altri campi, come il riconoscimento vocale e l'elaborazione del linguaggio naturale.

2.1 Struttura delle CNN

Una CNN è composta da una serie di strati, ognuno dei quali svolge un ruolo specifico nel processo di apprendimento. Questi strati possono essere suddivisi in tre categorie principali: strati di convoluzione, strati di pooling e strati completamente connessi.

Lo strato di convoluzione è il cuore di una CNN. È qui che avviene la maggior parte dell'elaborazione dei dati. Ma cosa significa esattamente "convoluzione"? In termini semplici, la convoluzione è un'operazione

matematica che combina due funzioni per produrre una terza funzione. Nel contesto delle CNN, una delle funzioni è l'immagine di input, e l'altra è un filtro, o kernel, che viene fatto scorrere sull'immagine.

Il processo di convoluzione permette alla CNN di estrarre caratteristiche locali dall'immagine, come bordi, texture e forme. Queste caratteristiche sono fondamentali per il riconoscimento di oggetti, poiché forniscono informazioni cruciali sulla struttura dell'immagine. Inoltre, poiché il filtro viene applicato a tutte le posizioni dell'immagine, la CNN è in grado di rilevare le stesse caratteristiche indipendentemente dalla loro posizione, rendendo il modello più robusto e generalizzabile.

Dopo lo strato di convoluzione, l'output viene tipicamente passato attraverso uno strato di pooling. Il pooling è un'operazione che riduce la dimensione spaziale dell'immagine, mantenendo le informazioni più rilevanti. Questo processo è noto come downsampling, e aiuta a ridurre la complessità computazionale della rete, oltre a prevenire l'overfitting.

Esistono diversi tipi di pooling, ma il più comune è il max pooling. Nel max pooling, l'immagine viene suddivisa in piccole regioni non sovrapposte, e per ciascuna regione viene selezionato il valore massimo. Ad esempio, se si utilizza un filtro di pooling 2x2, l'immagine viene suddivisa in blocchi di 2x2 pixel, e per ciascun blocco viene selezionato il valore massimo. Il risultato è una nuova immagine con dimensioni ridotte, ma che conserva le caratteristiche più importanti.

Il pooling introduce una forma di invarianza traslazionale, il che significa che la rete è meno sensibile a piccole traslazioni dell'immagine. Questo è particolarmente utile per il riconoscimento di oggetti, poiché gli oggetti possono apparire in diverse posizioni all'interno dell'immagine.

Dopo diversi strati di convoluzione e pooling, l'immagine viene "appiattita" in un vettore unidimensionale e passata attraverso uno o più strati completamente connessi. Questi strati sono simili a quelli delle reti neurali tradizionali, dove ogni neurone è connesso a tutti i neuroni dello strato precedente.

Lo scopo degli strati completamente connessi è quello di combinare le caratteristiche estratte dagli strati precedenti per fare una previsione finale. Ad esempio, in un problema di classificazione delle immagini, l'ultimo strato completamente connesso potrebbe avere un neurone per ogni classe, e l'output della rete sarebbe una distribuzione di probabilità che indica la probabilità che l'immagine appartenga a ciascuna classe.

Gli strati completamente connessi sono responsabili della "decisione" finale della rete, ma è importante notare che la maggior parte dell'apprendimento avviene negli strati di convoluzione. Gli strati completamente connessi agiscono come una sorta di "collo di bottiglia", combinando le informazioni estratte dagli strati precedenti per produrre un output compatto e interpretabile.

Un altro componente cruciale delle CNN, e delle reti neurali in generale, sono le funzioni di attivazione. Queste funzioni introducono non linearità nel modello, permettendo alla rete di apprendere rappresentazioni complesse dei dati. Senza le funzioni di attivazione, la rete sarebbe semplicemente una combinazione lineare degli input, e non sarebbe in grado di catturare le relazioni non lineari presenti nei dati.

La funzione di attivazione più comune nelle CNN è la ReLU (Rectified Linear Unit). La ReLU è una funzione molto semplice: restituisce l'input se è positivo, e zero altrimenti. Nonostante la sua semplicità, la ReLU ha dimostrato di essere estremamente efficace, poiché introduce non linearità senza complicare eccessivamente il calcolo.

Esistono anche altre funzioni di attivazione, come la sigmoid e la tanh, ma la ReLU è di gran lunga la più utilizzata nelle CNN moderne. Una variante della ReLU, chiamata Leaky ReLU, permette un piccolo gradiente anche per i valori negativi, il che può aiutare a prevenire il problema del "dying ReLU", dove i neuroni smettono di apprendere se ricevono sempre input negativi.

2.2 Addestramento di una CNN

L'addestramento di una CNN è un processo iterativo che coinvolge la minimizzazione di una funzione di perdita, che misura quanto l'output della rete si discosta dal valore desiderato. Questo processo è noto come apprendimento supervisionato, poiché la rete viene addestrata su un insieme di dati etichettati, dove ogni immagine è associata a una classe corretta.

Il metodo più comune per addestrare una CNN è l'algoritmo di backpropagation, combinato con l'ottimizzazione del gradiente discendente. Durante l'addestramento, l'output della rete viene confrontato con l'etichetta corretta, e la differenza viene utilizzata per calcolare il gradiente della funzione di perdita rispetto ai pesi della rete. Questi gradienti vengono poi utilizzati per aggiornare i pesi, in modo che la rete migliori le sue previsioni nel tempo.

L'addestramento delle CNN richiede una grande quantità di dati e potenza di calcolo, poiché le reti possono avere milioni di parametri da ottimizzare. Tuttavia, grazie ai progressi nei processori grafici (GPU) e nei processori tensoriali (TPU), è ora possibile addestrare CNN molto profonde e complesse in tempi ragionevoli.

Uno dei principali problemi nell'addestramento delle CNN è l'overfitting, che si verifica quando la rete si adatta troppo bene ai dati di addestramento e non generalizza bene ai dati nuovi. Per prevenire l'overfitting, vengono utilizzate diverse tecniche di regolarizzazione.

Una delle tecniche di regolarizzazione più comuni è il dropout. Il dropout è una tecnica semplice ma efficace che consiste nello "spegnere" casualmente una percentuale di neuroni durante l'addestramento. In altre parole, durante ogni iterazione dell'addestramento, alcuni neuroni vengono esclusi dal processo di apprendimento, e i loro contributi non vengono considerati. Questo costringe la rete a non dipendere troppo da particolari neuroni o connessioni, rendendola più robusta e meno incline all'overfitting. Durante il test, tutti i neuroni sono attivi, ma i pesi vengono scalati per compensare il dropout applicato durante l'addestramento.

Un'altra tecnica di regolarizzazione è la normalizzazione dei batch (batch normalization). Questa tecnica normalizza l'output di ogni strato della rete, in modo che abbia una media di zero e una varianza unitaria. La normalizzazione dei batch aiuta a stabilizzare e accelerare l'addestramento, riducendo la sensibilità della rete alle variazioni nei dati di input e migliorando la generalizzazione.

Oltre a queste tecniche, un'altra strategia comune per prevenire l'overfitting è l'uso di data augmentation. Il data augmentation consiste nel generare nuove immagini di addestramento applicando trasformazioni casuali alle immagini esistenti, come rotazioni, traslazioni, zoom, e riflessioni. Questo aumenta la varietà dei dati di addestramento senza la necessità di raccogliere nuove immagini, migliorando la capacità della rete di generalizzare a nuovi dati.

3

Creazione del dataset

La creazione di un dataset robusto e rappresentativo, cuore pulsante di qualsiasi progetto di Machine Learning, è un'operazione complessa e articolata, che richiede tempo, risorse e una pianificazione accurata. Il primo passo consiste nel definire con precisione l'obiettivo che si intende raggiungere, ovvero cosa vogliamo che il nostro modello debba imparare a riconoscere, e in questo modo potremo indirizzare la raccolta dati nella giusta direzione, assicurandoci che le immagini acquisite siano effettivamente utili per l'addestramento del modello.

Nel contesto del riconoscimento immagini, l'acquisizione dei dati rappresenta un aspetto cruciale per l'addestramento di modelli efficaci. Si possono utilizzare dataset pubblici di immagini, che offrono una vasta gamma di categorie e milioni di esempi annotati. Questi dataset sono un'ottima risorsa per la sperimentazione e lo sviluppo di modelli generici, ma potrebbero non essere sufficienti per applicazioni specifiche.

Per ottenere dati più pertinenti alle proprie necessità, è possibile raccogliere immagini in modo diretto. Questo può avvenire tramite fotocamere o dispositivi di acquisizione dedicati, a seconda del tipo di immagini richieste. La raccolta diretta permette di controllare l'illuminazione, l'inquadratura e altri parametri cruciali per la qualità delle immagini, ma richiede un investimento in termini di tempo e risorse.

Un'altra opzione è quella di acquistare dataset di immagini da fornitori specializzati. Queste aziende offrono collezioni di immagini ad alta risoluzione, annotate e categorizzate, per diversi ambiti di applicazione, come la medicina, l'industria o la sicurezza. Questa soluzione può essere vantaggiosa per progetti che richiedono un elevato grado di accuratezza e una vasta gamma di esempi, ma comporta un costo non trascurabile.

Gli aspetti generali da considerare durante la fase di raccolta dati sono molti, e sono tutti fondamentali affinché la qualità del nostro dataset rispetti in maniera soddisfacente i nostri standard:

- Varietà: il dataset deve essere il più possibile vario e diversificato, includendo immagini che rappresentino tutte le possibili variazioni dell'oggetto o della scena che si desidera riconoscere;

- Quantità: la dimensione del dataset è un fattore cruciale per l'addestramento del modello. In generale, più immagini sono disponibili, migliori saranno le prestazioni dell'algoritmo. Tuttavia, è importante bilanciare la quantità con la qualità, evitando di includere immagini irrilevanti o di bassa risoluzione;

- Qualità: le immagini devono possedere una risoluzione adeguata e un'illuminazione sufficiente. Immagini sfocate, poco nitide o con un basso contrasto possono compromettere l'apprendimento del modello;

- Etichettatura: ogni immagine del dataset deve essere accuratamente etichettata, ovvero associata ad una o più categorie che ne descrivono il contenuto. L'etichettatura può essere manuale, automatica o semi-automatica, a seconda della complessità del progetto e delle risorse disponibili.

3.1 I limiti della strumentazione

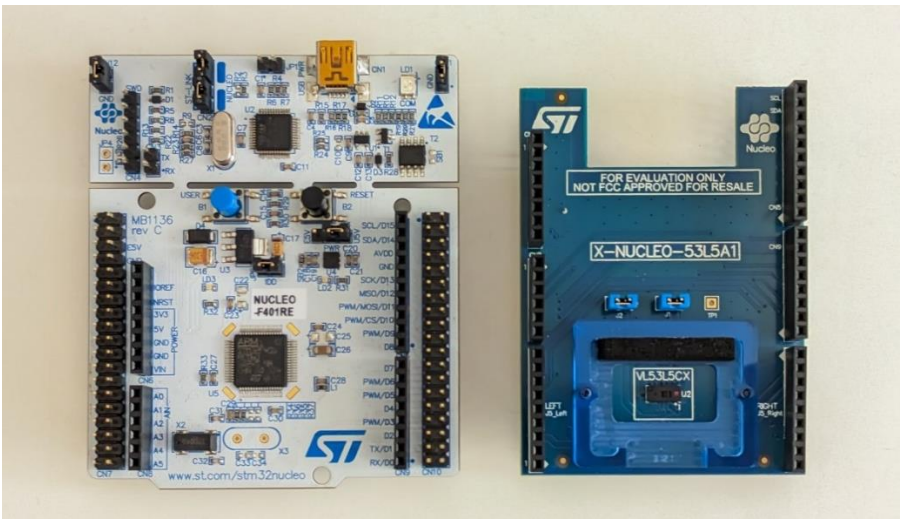
La raccolta dati, soprattutto se condotta manualmente, è un processo delicato influenzato da molteplici fattori. Tra questi, la qualità della strumentazione a disposizione gioca un ruolo fondamentale. Infatti, le caratteristiche e le limitazioni degli strumenti utilizzati impongono dei vincoli che si ripercuotono direttamente sulla tipologia e sulla qualità dei dati acquisiti.

Questo significa che, in fase di progettazione e implementazione del processo di raccolta dati, è imprescindibile una valutazione attenta delle potenzialità e dei limiti della strumentazione. Bisogna considerare aspetti

come la precisione, la sensibilità, la risoluzione e la velocità di acquisizione dello strumento: di conseguenza, è spesso necessario scendere a compromessi, adattando il protocollo ed il tipo di acquisizione alle caratteristiche dello strumento a disposizione.

Nel contesto del nostro progetto, la raccolta dati avverrà tramite l'utilizzo combinato della board NUCLEO-STM32F401 e della expansion board X-NUCLEO-53L5A1, entrambe prodotte da ST Microelectronics. La board di espansione X-NUCLEO-53L5A1 integra un sensore di prossimità Time-of-Flight (ToF) in grado di rilevare la distanza dagli oggetti e di rappresentare l'informazione acquisita sotto forma di una matrice di 8x8 pixel. Questa griglia di dati, tuttavia, presenta una risoluzione limitata, costituendo uno dei principali ostacoli già anticipati.

La bassa risoluzione di 8x8 pixel, infatti, influenza significativamente la complessità e la tipologia di gestures che potranno essere implementate e riconosciute dal sistema. Quelle che richiedono un elevato livello di dettaglio potrebbero risultare difficili da rilevare e interpretare correttamente a causa della scarsa granularità dei dati forniti dal sensore. Di conseguenza, la fase di ideazione e progettazione dovrà tenere conto di questa limitazione, privilegiando gestures semplici, ampie e ben definite che possano essere efficacemente discriminate anche con una risoluzione spaziale ridotta.



Le due board utilizzate per la raccolta dati

Ci appoggeremo al dataset preesistente realizzato da ST specificamente per questa board. Il nostro obiettivo è quello di espanderlo includendo nuove gestures, arricchendo così le funzionalità del sistema. La X-NUCLEO-53L5A1 consente di riconoscere sia gesti statici delle mani, come simboli e numeri, denominati “hand postures”, sia gesti in movimento, come uno swipe o un doppio click, denominati per comodità “gestures”. Noi ci concentreremo sulla prima di queste due categorie, i gesti statici.

L’analisi del lavoro svolto dall’azienda ha confermato le nostre supposizioni, evidenziando in particolare l'importanza della semplicità nella progettazione del nuovo dataset.

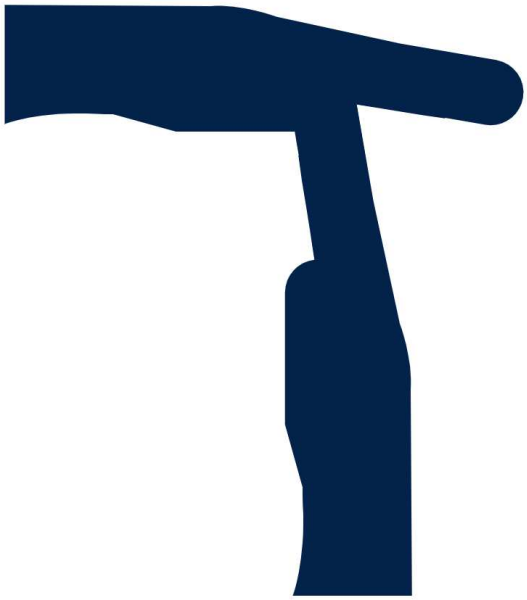
Infatti, emerge chiaramente come l'efficacia del sistema dipenda in larga misura dalla facilità di esecuzione e di riconoscimento delle gestures stesse. Un'eccessiva complessità potrebbe comportare grande incertezza nell'interpretazione da parte del sistema e, di conseguenza, una performance complessivamente insoddisfacente.

Pertanto, nella fase di ideazione e implementazione delle nostre nuove hand postures, ci impegneremo a mantenere un approccio minimalista, privilegiando movimenti chiari, intuitivi e facilmente distinguibili tra loro. Questo ci permetterà di garantire un'esperienza utente ottimale, caratterizzata da immediatezza e precisione nell'interazione con la board.

Il dataset di gesti, scaricato direttamente dal sito ufficiale di ST, fornisce una base solida per il nostro progetto di riconoscimento dei gesti. Questo dataset iniziale include otto distinti simboli, ognuno rappresentato da una specifica configurazione della mano. Tra questi otto simboli, uno rappresenta l'assenza di una mano nell'inquadratura, consentendo all'algoritmo di discernere tra la presenza e l'assenza di input gestuali. Questo simbolo "vuoto" è fondamentale per evitare falsi positivi e garantire un'interpretazione accurata dei gesti.

I restanti sette simboli offrono una gamma di gesti comuni e intuitivi:

- "breaktime", che indica una pausa, rappresentato da due mani che formano una T;



“breaktime”

- "crosshands", in cui le braccia sono incrociate;

“crosshands”

- "dislike", il classico pollice in giù;

“dislike”

- "like", il suo opposto, con il pollice alzato in segno di approvazione;

“like”

- "fist", un pugno chiuso;

“fist”

- "flathand", una mano aperta con il palmo rivolto in avanti;

“flathand”

- "love", rappresentato da due mani che formano un cuore.

“love”

Queste gestures costituiscono un buon fondamento per lo sviluppo del nostro nuovo dataset: la loro analisi ci permette infatti di identificare categorie non ancora contemplate, oltre ad evitare il banale errore di acquisire simboli già presenti, e di incorrere di conseguenza in ripetizioni. Questo garantirà l’originalità ed il valore aggiunto del dataset che andremo a produrre.

Dopo un'approfondita analisi delle diverse gestures possibili e delle loro implicazioni in termini di usabilità, intuitività e chiarezza comunicativa, si è giunti alla selezione di quattro principali, ognuna con un design distintivo per facilitarne il riconoscimento e la memorizzazione da parte dell'utente:

- La prima, rappresentata da una mano vista orizzontalmente, con il palmo rivolto verso il basso e le dita estese, evoca un gesto di stop o di pausa, ma può ovviamente essere implementata anche per altre tipologie di azioni;

- La seconda, caratterizzata da una mano vista verticalmente, con il palmo rivolto lateralmente e le dita estese, richiama un movimento di scorrimento;

- La terza, che consiste nel gesto del numero uno con l'indice alzato, simboleggia la selezione di un singolo elemento, o potrebbe rappresentare uno scorrimento in senso opposto rispetto alla gesture precedente;

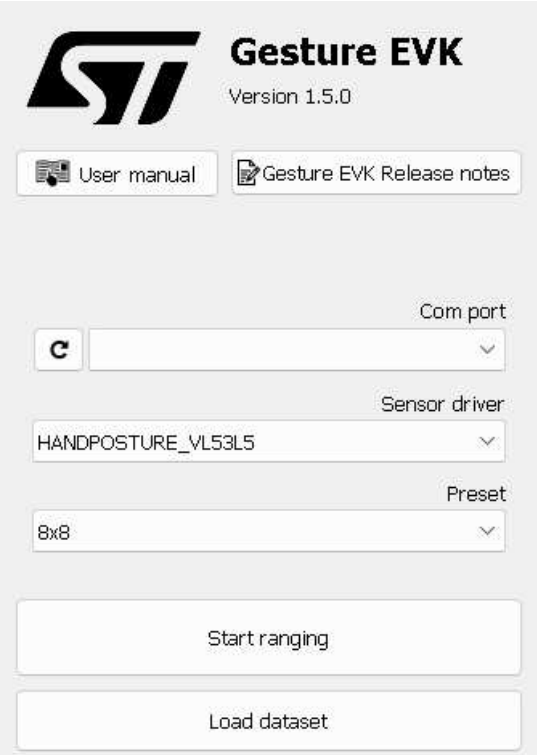
- La quarta, rappresentata da una mano a formare una "C", può essere interpretata come un gesto di "cattura" dello schermo o quello di chiamare qualcuno (utile anche nella traduzione inglese di “call”) sotto forma di aiuto se prendiamo in considerazione un ambiente ospedaliero o di assistenza agli anziani.

3.2 La raccolta dati

Per acquisire le nuove hand postures, abbiamo scelto di utilizzare l'applicazione Gesture Evaluation Kit (Gesture EVK) fornita direttamente da ST Microelectronics e progettata specificamente per la board che abbiamo selezionato per il nostro progetto. Questa applicazione è dotata di un’interfaccia utente estremamente semplice ed intuitiva, che ci ha permesso di velocizzare notevolmente il lavoro.

All’avvio del programma l’utente è accolto da una finestra che permette di selezionare due distinte modalità di acquisizione, ovvero quella delle “hand posture” e quella delle “gestures” in movimento. Oltre alle due modalità di acquisizione in tempo reale, il programma offre anche la possibilità di caricare un "Record di Acquisizioni" tramite file di log precedentemente salvati. Questa funzione consente di rivedere e analizzare i dati raccolti in sessioni precedenti, visualizzando i grafici di rilevamento generati dalla board. I grafici mostrano in modo dettagliato l'andamento dei segnali acquisiti dai sensori, fornendo informazioni preziose per la comprensione e l'interpretazione dei movimenti e delle posture della mano. Questo strumento risulta molto utile nel caso si volessero effettuare confronti tra diverse sessioni di acquisizione, o semplicemente si volessero rivedere e studiare i dati in un momento successivo alla loro acquisizione: metodo indubbiamente più comodo rispetto a quello di visionare manualmente i dati aprendo i file di log in forma tabellare tramite apposite applicazioni.

Prima di poter iniziare l'acquisizione dei dati e immergerci nell'analisi delle posture della mano, è fondamentale configurare correttamente il programma selezionando sia la board di acquisizione che la modalità di rilevamento appropriata. Questo passaggio preliminare assicura che il programma sia in grado di comunicare con l'hardware specifico e di interpretare correttamente i dati ricevuti. Nel nostro caso, andremo a selezionare la board VL53L5 nella modalità “hand posture”.



L’interfaccia utente iniziale dell’applicazione Gesture EVK: possiamo notare la board e la modalità di acquisizione scelte nella sezione “Sensor driver”, l’unica risoluzione disponibile nella sezione “Preset”, e in basso la possibilità di scegliere tra avviare una nuova acquisizione dati (“Start ranging”) o caricare un log precedentemente registrato (“Load dataset”) per lo studio dei valori acquisiti.

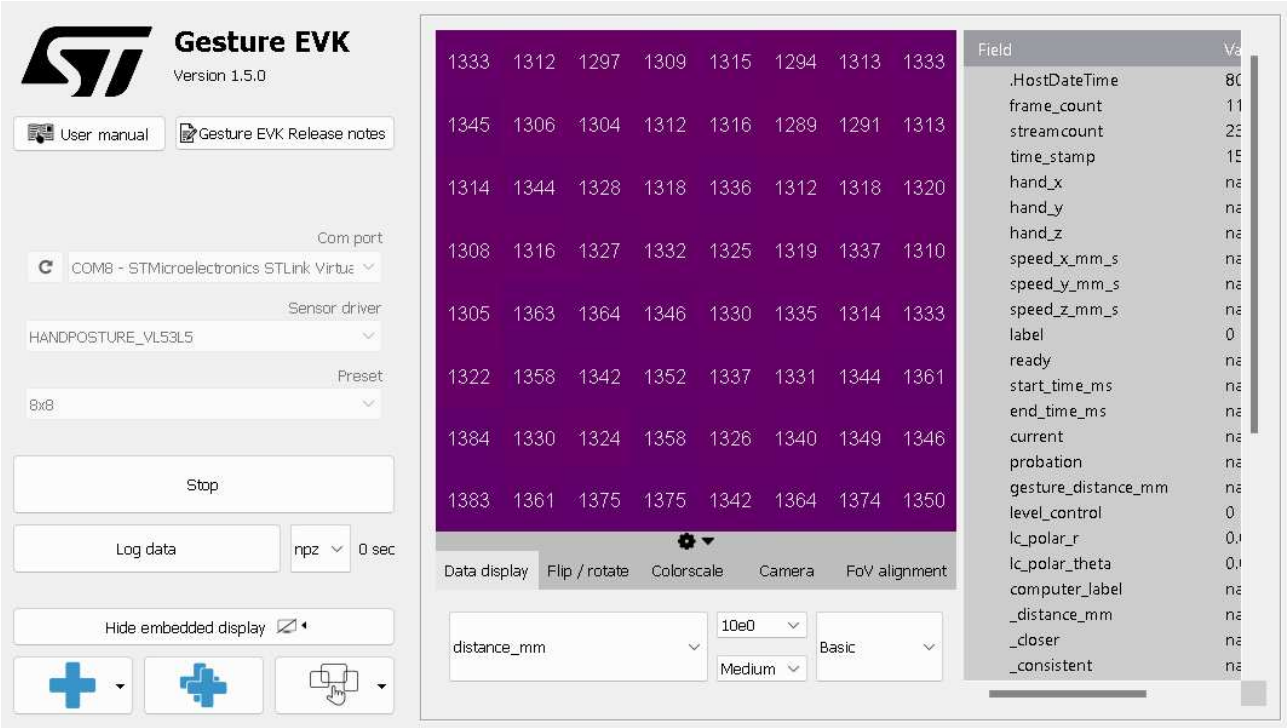
Una volta collegata la board al computer tramite cavo USB, siamo pronti ad avviare il programma principale tramite il pulsante “Start ranging” presente nell’interfaccia iniziale dell’applicazione.

La nuova interfaccia si presenta con la maggior parte dello spazio occupato da una griglia centrale. Questa griglia visualizza i dati provenienti dal sensore di profondità, rappresentando la distanza tra il sensore e l'oggetto di fronte ad esso in millimetri per ogni singolo pixel. In questo modo, si ottiene una rappresentazione visiva approssimata della forma e della posizione dell'oggetto rilevato.

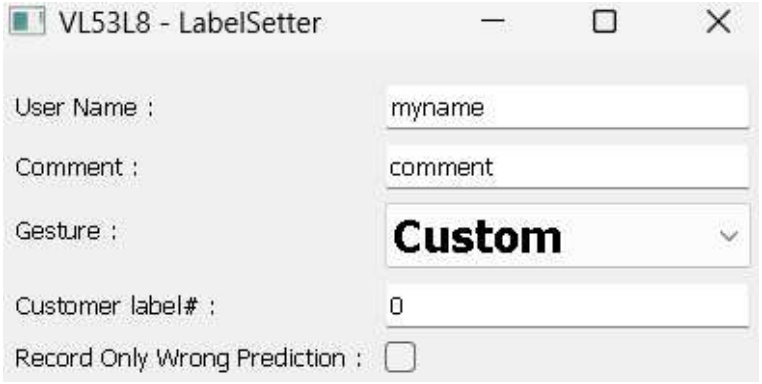
Tra la moltitudine di altre sezioni presenti nell'interfaccia, nel nostro progetto siamo interessati a due di queste nello specifico:

- La prima è quella dedicata all'avvio ed all'interruzione dell'acquisizione dati da parte del sensore, denominata "Log data". Qui è presente la possibilità di salvare il file di log in due formati diversi, ".csv" e ".npz". Nel nostro caso, opteremo per il formato ".npz", essendo il formato compresso degli array numpy, libreria Python molto versatile che con la sua estesa compatibilità permette di semplificare il processo di importazione e manipolazione dati in fase di addestramento di un modello;

- La seconda sezione di interesse è dedicata alla creazione e gestione delle etichette, ed è denominata "Label setter", presente nella tendina degli strumenti aggiuntivi che appare premendo il pulsante in basso a sinistra nell'interfaccia. Questa ci permette di assegnare un nome descrittivo e un codice numerico univoco ad ogni hand posture che vogliamo rilevare. Questa associazione è fondamentale per il processo di apprendimento del modello, in quanto permette di classificare correttamente i diversi simboli durante la fase di addestramento e di interpretare i risultati in modo chiaro e comprensibile. La flessibilità di questa sezione ci consente perciò di definire un vocabolario ben organizzato.



L’interfaccia utente principale dell’applicazione

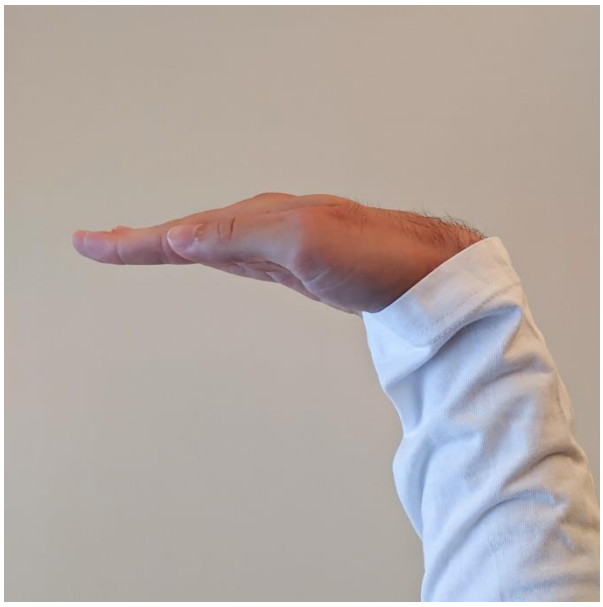


La finestra dedicata alla creazione di una nuova etichetta

Siamo ora pronti ad avviare l'acquisizione delle nostre quattro hand postures: nella sezione “Comment” della sezione appena menzionata abbiamo inserito di volta in volta il nome del gesto che vogliamo registrare, assicurandoci che ogni campione sia correttamente etichettato per facilitare il successivo processo di addestramento del modello. Le quattro etichette che abbiamo scelto sono “uno”, “mano_C”, “mano_verticale” e “mano_orizzontale”, di cui di seguito riporteremo le rispettive immagini per comprendere più facilmente la postura della mano.

Il numero uno

la forma della lettera C



La mano vista orizzontalmente

la mano vista verticalmente

Per garantire l'eterogeneità nei campioni di hand postures raccolti, è stato deciso di effettuare per ogni simbolo diverse acquisizioni tramite l'aiuto di alcuni volontari, ciascuna della durata di circa 10 secondi. Questa durata è stata scelta per catturare una gamma sufficiente di piccole variazioni nella posizione e nell'angolazione della mano, migliorando così la capacità del nostro futuro modello di generalizzare e riconoscere i gesti in diverse condizioni.

L'eterogeneità è infatti fondamentale nel contesto dell'apprendimento automatico, per garantire che i modelli siano in grado di funzionare correttamente su un insieme di dati arricchito dall'unicità di ogni singolo individuo, permettendo di coprire una gamma più ampia di dimensioni e forme, che a loro volta migliorano la robustezza e la portabilità degli stessi.

Una volta terminato il processo di acquisizione dati, per ulteriormente arricchire i campioni di hand postures, verrananno successivamente applicate delle tecniche di data augmentation: queste tecniche consistono nell'aggiungere nuovi dati alla base esistente attraverso la manipolazione degli esempi già raccolti, permettendo così di rendere l'insieme di immagini sulle quali verrà addestrato il modello ancora più vario nel suo complesso.

Di seguito verrà riportata una tabella riassuntiva contenente il confronto tra la consistenza del dataset della ST ed il nostro:

Dataset	**Durata acquisizioni**	**Soggetti partecipanti**
ST	Circa 120 frames	Sconosciuto
Custom	Circa 10 secondi	4

4

Preparazione dei dati

4.1 Le tecniche di elaborazione

Dopo aver raccolto un dataset sufficientemente ampio e diversificato per l'addestramento di un modello di machine learning, soprattutto nel campo della visione artificiale, è fondamentale elaborarlo e ottimizzarlo. Questo processo di raffinamento è cruciale per massimizzare le prestazioni del modello e renderlo più robusto. Una delle tecniche più efficaci in questo ambito è la data augmentation.

Questa tecnica si basa sul principio di generare nuove immagini sintetiche a partire da quelle già presenti nel dataset originale. Applicando una serie di trasformazioni alle immagini esistenti, si ottiene un dataset più ampio e variegato, senza la necessità di raccogliere nuovi dati. Questo approccio presenta numerosi vantaggi, infatti esponendo il modello a una gamma più ampia di variazioni delle immagini, lo aiuta a prevenire la tendenza ad apprendere troppo bene le caratteristiche specifiche del dataset originale, compromettendo la sua capacità di generalizzare a nuovi dati.

Le trasformazioni utilizzate nella data augmentation sono molteplici e possono essere combinate tra loro per creare un'ampia varietà di nuove immagini, amplificando notevolmente l'effetto di aumento del dataset. Tra le più comuni troviamo:

- Rotazione: consiste nel ruotare l'immagine di un certo angolo, ad esempio di 90, 180, 270 gradi, o anche angoli casuali all'interno di un intervallo predefinito. Questa trasformazione aiuta il modello a riconoscere l'oggetto indipendentemente dal suo orientamento nello spazio;

- Riflessione: si può riflettere l'immagine orizzontalmente (lungo l'asse verticale) o verticalmente (lungo l'asse orizzontale), creando una versione "speculare" di quella originale. La riflessione orizzontale è particolarmente

utile per oggetti che possono apparire in entrambe le direzioni, come nel nostro esempio, le mani;

- Zoom: consiste nell'ingrandire o rimpicciolire l'immagine, simulando la variazione di distanza dall'oggetto o un effetto di cropping. Lo zoom può essere applicato in modo uniforme in entrambe le direzioni oppure solo in una direzione specifica. Questo aiuta il modello a gestire le variazioni di scala degli oggetti nelle immagini, imparando a riconoscerli indipendentemente dalla loro dimensione relativa all'immagine;

- Traslazione: si trasla l'immagine orizzontalmente o verticalmente, simulando un cambiamento di inquadratura o un leggero spostamento dell'oggetto all'interno della scena. La traslazione può essere applicata di un numero fisso di pixel oppure di un valore casuale entro un certo intervallo. Questa trasformazione insegna al modello che l'oggetto può essere posizionato in diverse aree dell'immagine e non necessariamente al centro;

- Aggiunta di rumore: consiste nell'introdurre rumore casuale nell'immagine, simulando la presenza di disturbi nella fase di acquisizione, come ad esempio rumore gaussiano, o rumore di Poisson. Questo aiuta il modello a diventare più robusto a imperfezioni e rumore presente in dati reali, imparando a concentrarsi sulle caratteristiche essenziali dell'oggetto e a ignorare i dettagli irrilevanti o i disturbi.

Oltre alla data augmentation, esistono diverse altre tecniche di elaborazione delle immagini che possono essere utilizzate per migliorare la qualità dei dati: una delle più comuni ed importanti è la normalizzazione.

La normalizzazione è un processo che prevede l'adattamento dei valori dei pixel all'interno di un intervallo specifico, solitamente tra 0 e 1. Questa tecnica serve principalmente a standardizzare i dati ed a mitigare la sensibilità di alcuni algoritmi alle variazioni di scala all'interno di un dataset. Si aiuta quindi il modello a semplificare il processo di apprendimento, poiché i valori normalizzati sono più gestibili e possono essere trattati in modo più efficiente da diversi algoritmi.

Oltre alle tecniche già menzionate, per ottimizzare ulteriormente il nostro dataset possiamo adottare tre tipologie di elaborazione specifiche:

- Bilanciamento delle classi: questo processo si applica quando il dataset presenta uno squilibrio tra le diverse classi di dati. Ad esempio, in un dataset di immagini di differenti categorie, potrebbe esserci un numero significativamente maggiore di una rispetto ad un'altra. Questo squilibrio può portare a modelli che prediligono la classe maggioritaria, trascurando la classe minoritaria. Il bilanciamento delle classi mira a riequilibrare la distribuzione delle classi, ad esempio tramite tecniche di oversampling (replicazione degli esempi della classe minoritaria) o undersampling (rimozione di esempi dalla classe maggioritaria);

- Feature extraction: questa tecnica si concentra sull'estrazione di informazioni rilevanti dai dati originali, creandone di nuove che catturano le caratteristiche essenziali del dataset. Ad esempio, da un'immagine possiamo estrarre feature come il colore dominante, la forma, la texture o la posizione di elementi specifici. L'estrazione di feature consente di ridurre la dimensionalità del dataset, semplificando il processo di apprendimento del modello e migliorando le performance. Nel nostro caso specifico utilizzeremo questa tecnica per rimuovere le informazioni catturate dal sensore che non ci interessano e mantenere quelle fondamentali, come la distanza in millimetri dell'oggetto rilevato, unica caratteristica importante per discriminare dove si trova la mano all'interno dell'inquadratura;

- Compressione dei dati: in alcuni casi, il dataset potrebbe essere di dimensioni eccessive, rendendo difficile la sua gestione e l'addestramento di modelli complessi. La compressione dei dati mira a ridurre la sua dimensione senza perdere informazioni cruciali. Esistono diverse tecniche di compressione, come la compressione lossless (che non comporta perdita di informazioni) o la compressione lossy (che comporta una piccola perdita di informazioni ma riduce significativamente la dimensione del dataset).

4.2 I file di configurazione

Una volta organizzata correttamente la cartella contenente il dataset, il passo cruciale prima di avviare il training del modello è quello di modificare alcuni parametri all'interno di due files di configurazione. Questi files sono contenuti nella cartella scaricata dalla repository GitHub[1], che racchiude tutto il necessario per le operazioni che dobbiamo svolgere. I due files in questione sono "handposture_dictionnary.py" e "user_config.yaml".

Nel primo file di configurazione, "handposture_dictionnary.py", è presente, come suggerisce il nome, un dizionario contenente le etichette delle varie hand postures, con i nomi e il corrispettivo numero assegnato. Questo dizionario svolge un ruolo importante nel processo di classificazione dei simboli, fungendo da ponte tra le rappresentazioni numeriche utilizzate dal modello di machine learning e le etichette leggibili dall'uomo.

In questo file dovremo inserire le nostre nuove gestures, alle quali abbiamo assegnato ad ognuna in precedenza il nome ed il numero univoco che la contraddistingue dalle altre. Mantenere la coerenza tra questi due parametri ed il nome delle cartelle presenti nel nostro dataset è fondamentale per prevenire l'insorgere di errori nella fase di training.

Dopo aver inserito quindi le nuove etichette nel file "handposture_dictionnary.py", la sua struttura appare così:

```
hand_posture_dict = {

    ...

    "mano_verticale":1,
    "mano_orizzontale":2,
    "mano_C":3,
    "indice_alzato":4,

    ...

}
```

Dove sono state omesse per semplicità tutte le altre etichette contenute all'interno del file.

Passiamo ora alla configurazione del file più importante per la corretta riuscita del nostro progetto, il file "user_config.yaml". All'interno di questo file sono contenute tutte le impostazioni necessarie per svolgere operazioni con il nostro dataset e con i modelli preaddestrati a nostra disposizione.

Il nostro obiettivo, in particolare, è quello di "aggiornare" uno dei modelli già presenti nella repository della ST per poter riconoscere oltre ai simboli già presenti anche quelli che abbiamo appena creato. Per fare ciò, dobbiamo modificare e personalizzare alcune sezioni del file per adattarle alle nostre esigenze.

Nella prima sezione sono presenti le impostazioni generali:

```yaml
general:
   project_name: handposture
   logs_dir: logs
   saved_models_dir: saved_models
#  model_path: <file-path>
   global_seed: 123
   deterministic_ops: False
   display_figures: True

   gpu_memory_limit: 24
```

I primi tre parametri sono opzionali, e possono essere modificati semplicemente per la comodità di poter scegliere il nome del progetto e la cartella nella quale saranno salvati i nuovi modelli addestrati.

Il parametro “global_seed” serve a generare sequenze casuali utilizzate nella fase di addestramento del modello per compiere una sequenza di scelte, e di default è impostato a 123. Può essere modificato, ma renderebbe l’esperimento difficilmente riproducibile esattamente.

Il parametro “deterministic_ops”, se impostato a “True”, serve a rendere l’esperimento deterministico, ovvero a parità di impostazioni si ottengono sempre gli stessi risultati. La modifica di questo parametro è utile nel caso si volesse riprodurre l’esperimento su dispositivi differenti ottenendo le

stesse caratteristiche in uscita, ma il compromesso da accettare è il maggior tempo di esecuzione.

Il parametro “gpu_memory_limit” serve semplicemente ad impostare un limite superiore espresso in Gigabytes della memoria GPU che può essere utilizzata durante l’esperimento.

Ora, nella riga successiva, tra le varie operazioni a nostra disposizione, ovvero “training”, “evaluation”, “benchmarking” e “deployment”, scegliamo la prima modalità:

```
operation_mode: training
```

Nella successiva sezione è invece necessario dichiarare esplicitamente il nome della cartella del nostro dataset, i nomi delle classi e il percorso completo dove sono presenti le cartelle contenenti i files per l’addestramento. Nell’immagine di esempio sono presenti le gestures preesistenti, che nel nostro caso andremo a sostituire con i nomi delle quattro nuove classi che abbiamo creato, e allo stesso modo andremo a sostituire il percorso di default con quello contenente la nostra cartella principale:

```
dataset:
  dataset_name: ST_handposture_dataset
  class_names: [None, Like, Dislike, FlatHand, Fist, Love, BreakTime, CrossHands]
  training_path: ../datasets/ST_VL53L8CX_handposture_dataset
  validation_path: <validation-set-root-directory>
  validation_split: 0.2
  test_path: <test-set-root-directory>
```

Gli altri parametri vengono lasciati inalterati.

Nelle righe seguenti dedicate al preprocessing possiamo mantenere i dati di default, poiché sono già calibrati correttamente anche per il nostro progetto:

```
preprocessing: # Mandatory
  Max_distance: 400 # Mandatory
  Min_distance: 100 # Mandatory
  Background_distance:  120 # Mandatory
```

La distanza minima e massima dal sensore indicano il range permesso entro il quale considerare i rilevamenti della mano. Tutti i valori più vicini o più lontani vengono automaticamente esclusi in fase di addestramento in quanto non rispettano questa condizione.

La "background_distance" indica la distanza dello sfondo rispetto alla mano. Qualsisi valore che sia superiore del numero indicato rispetto alle rilevazioni all'interno del range consentito viene quindi eliminato perché considerato sfondo.

Nella sezione dedicata alla data augmentation, come accennato in precedenza, ci limiteremo alla semplice riflessione orizzontale, in modo tale da permettere al modello di allenarsi sulle stesse rilevazioni ma svolte sia con la mano destra che con quella sinistra:

```
data_augmentation:
  random_flip:
    mode: horizontal # horizontal, vertical or horizontal_and_vertical
```

Nella sezione di training è conveniente evitare di modificare i parametri in quanto sono già stati predisposti in modo da rispettare tutti i requisiti generalmente necessari e le impostazioni specifiche della board in utilizzo.

Il nome del modello inserito di default è quello che verrà utilizzato nella fase di addestramento per il riconoscimento delle nostre hand postures.

Il parametro "input_shape" rappresenta la dimensione della matrice di dati acquisiti durante la fase di raccolta.

Il numero di "epochs" indica invece il numero di volte che l'intero dataset verrà mostrato al modello durante la fase di training.

I parametri all'interno della sezione "EarlyStopping" servono, nel caso in cui i dataset siano di dimensioni ridotte, ad interrompere la fase di training dopo un numero prefissato di epochs, per evitare che il sistema iteri un numero eccessivo di volte sugli stessi dati, rendendo il processo di addestramento controproducente.

```yaml
training:
  model:
    name: CNN2D_ST_HandPosture
    version: v1
    input_shape: (8, 8, 2)
  frozen_layers: #(0:-1)
  dropout: 0.2
  batch_size: 32
  epochs: 1000
  optimizer:
      # Use Keras Adam optimizer with initial LR set to 0.001
    Adam:
      learning_rate: 0.01
  callbacks:          # Optional section
      # Use Keras ReduceLROnPlateau learning rate scheduler
    ReduceLROnPlateau:
      monitor: val_loss
      factor: 0.1
      patience: 20
      min_lr: 1.0e-04
    EarlyStopping:
      monitor: val_accuracy
      restore_best_weights: true
      patience: 40
```

Dopo aver configurato i due file in questione, è necessario passare all'installazione dei requisiti indispensabili per il corretto funzionamento del codice “stm32ai_main.py”. Questo programma è responsabile dell'avvio e della gestione del processo di addestramento, ed è strettamente legato a una serie di file correlati che ne supportano l'operatività. Per garantire che tutte le librerie e i pacchetti necessari siano presenti e correttamente configurati, utilizzeremo il file "requirements.txt". Questo file di testo, che si trova nella cartella scaricata dal repository GitHub, elenca tutte le dipendenze essenziali per il progetto. Esso include specifiche versioni di librerie Python che devono essere installate per evitare conflitti e assicurare la compatibilità del codice.

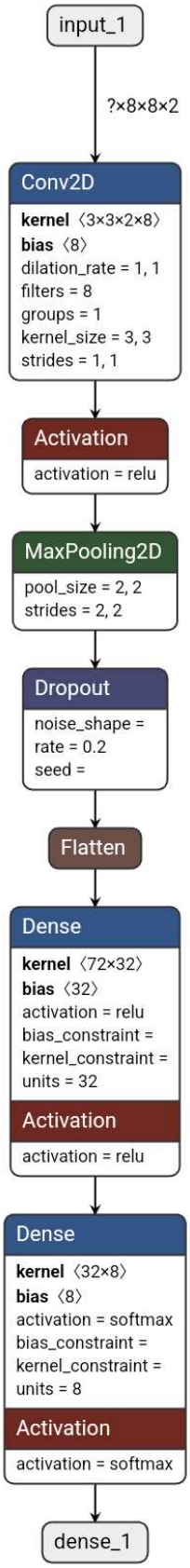
Per procedere con l'installazione, apriamo un terminale nella directory del progetto e utilizziamo il comando pip, un gestore di pacchetti per Python. Eseguendo il comando:

```
pip install -r requirements.txt
```

pip leggerà il file "requirements.txt" e installerà tutte le librerie elencate. Questo processo richiede alcuni minuti, per via dell'elevato numero di librerie presenti all'interno del file di testo. Una volta completata l'installazione, avremo a disposizione un ambiente di sviluppo completamente configurato, pronto per eseguire "stm32ai_main.py" senza incontrare errori legati a dipendenze mancanti o versioni incompatibili.

Al fine del corretto funzionamento di tutti i requisiti necessari, sul sito della ST viene consigliato di utilizzare una versione di Python che sia compresa tra la "3.9.x" e la "3.10.x", in quanto versioni successive entrano in conflitto con alcuni parametri dell'applicazione dedicata al training del modello.

4.3 Architettura del modello utilizzato



Il modello utilizzato per l'addestramento è stato selezionato tra una serie di modelli preaddestrati disponibili nella repository ufficiale della ST: in questo modo possiamo sfruttare le conoscenze già acquisite dal modello su un ampio set di dati, riducendo così il tempo e le risorse necessarie per l'addestramento specifico.

Il modello scelto è basato su una rete neurale convoluzionale (CNN) bidimensionale, una tipologia di architettura particolarmente efficace nell'elaborazione di dati strutturati in griglie 2D, come le immagini. Questa tipologia di rete è nota per la sua capacità di estrarre automaticamente caratteristiche rilevanti dai dati di input, rendendola particolarmente adatta per compiti di visione artificiale e riconoscimento di pattern.

Una caratteristica distintiva del modello scelto è la sua dimensione ridotta, che ammonta a soli 75 KB. Questa compattezza è un vantaggio significativo, soprattutto in applicazioni su dispositivi con risorse limitate, come nel nostro caso.

Grazie a queste caratteristiche, non è stato necessario applicare tecniche di compressione del modello, che spesso vengono utilizzate per ridurre le dimensioni dei modelli più grandi a scapito di una potenziale perdita di accuratezza. La compattezza intrinseca del modello ha permesso di mantenere un buon equilibrio tra prestazioni e efficienza delle risorse.

L’architettura nel dettaglio è mostrata in figura.

Primo blocco: immagine di input

Il processo inizia con l'immagine di input, che ha dimensioni 8x8x2. Questo significa che l'immagine è composta da 8 pixel per lato, con 2 canali. Ogni singolo frame delle acquisizioni viene quindi rappresentato come una matrice bidimensionale, con un ulteriore asse che rappresenta i canali. Questo input viene fornito alla rete neurale per l'elaborazione successiva.

Secondo blocco: primo strato convoluzionale

Il secondo blocco è il primo strato convoluzionale della rete, che applica 8 filtri convoluzionali di dimensione 3x3 all'immagine di input. Questi filtri sono progettati per scorrere sull'immagine un pixel alla volta, come indicato dal parametro "strides" impostato a 1. Ogni filtro convoluzionale agisce come un rilevatore di caratteristiche specifiche, come bordi, angoli o texture, estraendo informazioni rilevanti dall'immagine. L'output di questo strato è un insieme di mappe di caratteristiche, ognuna delle quali rappresenta una diversa interpretazione dell'immagine originale, basata sui filtri applicati.

Terzo blocco: funzione di attivazione ReLU

Il terzo blocco introduce la funzione di attivazione ReLU (Rectified Linear Unit), che viene applicata all'output del primo strato convoluzionale. La ReLU è una funzione non lineare che trasforma ogni valore negativo in zero, mantenendo invariati i valori positivi. Questa operazione introduce non-linearità nel modello, permettendo alla rete di apprendere pattern complessi e di migliorare la capacità di generalizzazione. La ReLU è particolarmente efficace nel prevenire il problema del gradiente che scompare, comune nelle reti neurali profonde.

Quarto blocco: MaxPooling2D

Il quarto blocco è uno strato di MaxPooling2D, che riduce la dimensione spaziale delle mappe di caratteristiche generate dal blocco precedente. Questo strato utilizza una finestra di dimensione 2x2 che si sposta sull'immagine, selezionando il valore massimo all'interno di ogni finestra. Questa operazione non solo riduce la dimensionalità dell'output, ma rende anche le caratteristiche estratte più robuste rispetto a piccole traslazioni

dell'immagine di input. Il MaxPooling aiuta a ridurre il numero di parametri e a prevenire l'overfitting, migliorando l'efficienza computazionale della rete.

Quinto blocco: dropout

Il quinto blocco introduce una tecnica di regolarizzazione chiamata dropout, che aiuta a prevenire l'overfitting durante l'addestramento della rete. In questo caso, il dropout è impostato al 20%, il che significa che, durante ogni iterazione di addestramento, il 20% dei neuroni della rete viene "spento" casualmente. Questo costringe la rete a non dipendere troppo da singoli neuroni e a imparare rappresentazioni più distribuite e robuste delle caratteristiche. Durante il test, tutti i neuroni sono attivi, ma i pesi sono scalati per compensare il dropout applicato durante l'addestramento.

Sesto blocco: flatten

Il sesto blocco è la funzione Flatten, che trasforma l'output tridimensionale degli strati precedenti in un vettore unidimensionale. Questo passaggio è necessario per collegare le mappe di caratteristiche estratte dagli strati convoluzionali agli strati completamente connessi (Dense). Il Flattening permette di passare da una rappresentazione spaziale a una rappresentazione vettoriale, che può essere elaborata dagli strati successivi per la classificazione finale.

Settimo blocco: primo strato completamente connesso (Dense)

Il settimo blocco rappresenta il primo strato completamente connesso della rete, che elabora il vettore di caratteristiche estratto. Questo strato è composto da 32 neuroni, ciascuno dei quali è connesso a tutti i neuroni dell'input appiattito. Il kernel di questo strato è una matrice di pesi che determina l'importanza di ciascuna caratteristica per la classificazione. Anche in questo caso, viene applicata la funzione di attivazione ReLU, che introduce non-linearità e permette alla rete di apprendere relazioni complesse tra le caratteristiche.

Ottavo blocco: secondo strato completamente connesso (Dense) e softmax

L'ottavo blocco è il secondo strato completamente connesso, che ha il compito di produrre la classificazione finale. Questo strato è composto da 8 neuroni, corrispondenti al numero di classi di output. I 32 neuroni del blocco precedente sono connessi a questi 8 neuroni, e l'output di questo strato viene passato attraverso la funzione softmax. La softmax converte i valori di output in una distribuzione di probabilità, assegnando a ciascuna classe una probabilità compresa tra 0 e 1. Questo permette di interpretare l'output della rete come una previsione probabilistica della classe a cui appartiene l'immagine di input.

Nono blocco: output finale

L'ultimo blocco rappresenta l'output finale della rete, che contiene le probabilità per ciascuna delle classi delle posture delle mani. Questo output è il risultato del secondo strato completamente connesso e della funzione softmax, e rappresenta la previsione della rete su quale classe l'immagine di input appartiene.

5

Risultati sperimentali

5.1 Addestramento del modello

Siamo ora pronti ad avviare il programma per l'addestramento del nostro modello. Dalla cartella contenente il file "stm32ai_main.py" apriamo il prompt e digitiamo il seguente comando:

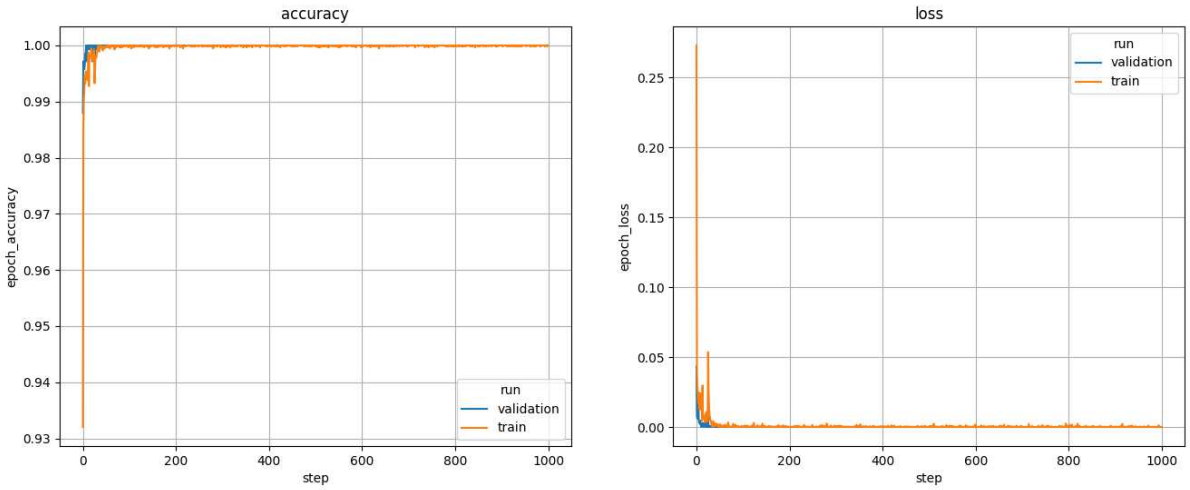
```
python stm32ai_main.py
```

Ora non resta che attendere il normale sviluppo del programma, che stamperà a schermo le varie fasi del processo di training, di cui riporteremo alcuni momenti chiave:

```
Training and Validation dataset using a validation_split = 0.2:
44 .npz files found
Analysing data shapes: 100%|████████████████████████████████████████████████████████████|
44/44 [00:00<00:00, 278.50it/s]
Dataset needs 0.007 GB RAM size (0.014 GB is required during loading)
Loading 44 .npz files
100%|████████████████████████████████████████████████████████████████████████████████████|
44/44 [00:00<00:00, 172.57it/s]
Dropped 1 (over 3348) frames because the hand is outside the Min / Max defined in user_c
onfig.yaml
Dataset stats:
  classes: 4
  training set size: 2677
  validation set size: 670
```

```
Starting training...
Epoch 1/1000
 1/84 [..............................] - ETA: 1:03 - loss: 5.0161 - accuracy: 0.4062WARN
ING:tensorflow:Callback method `on_train_batch_end` is slow compared to the batch time (
batch time: 0.0018s vs `on_train_batch_end` time: 0.0033s). Check your callbacks.
[2024-08-14 11:35:55,842][tensorflow][WARNING] - Callback method `on_train_batch_end` is
 slow compared to the batch time (batch time: 0.0018s vs `on_train_batch_end` time: 0.00
33s). Check your callbacks.
84/84 [==============================] - 1s 5ms/step - loss: 0.2519 - accuracy: 0.9387 -
 val_loss: 0.0052 - val_accuracy: 0.9985 - lr: 0.0100
Epoch 2/1000
84/84 [==============================] - 0s 3ms/step - loss: 0.0150 - accuracy: 0.9948 -
 val_loss: 0.0013 - val_accuracy: 1.0000 - lr: 0.0100
Epoch 3/1000
84/84 [==============================] - 0s 3ms/step - loss: 0.0049 - accuracy: 0.9989 -
 val_loss: 1.0472e-04 - val_accuracy: 1.0000 - lr: 0.0100
Epoch 4/1000
84/84 [==============================] - 0s 3ms/step - loss: 0.0135 - accuracy: 0.9955 -
 val_loss: 0.0074 - val_accuracy: 0.9940 - lr: 0.0100
Epoch 5/1000
84/84 [==============================] - 0s 4ms/step - loss: 0.0080 - accuracy: 0.9970 -
```

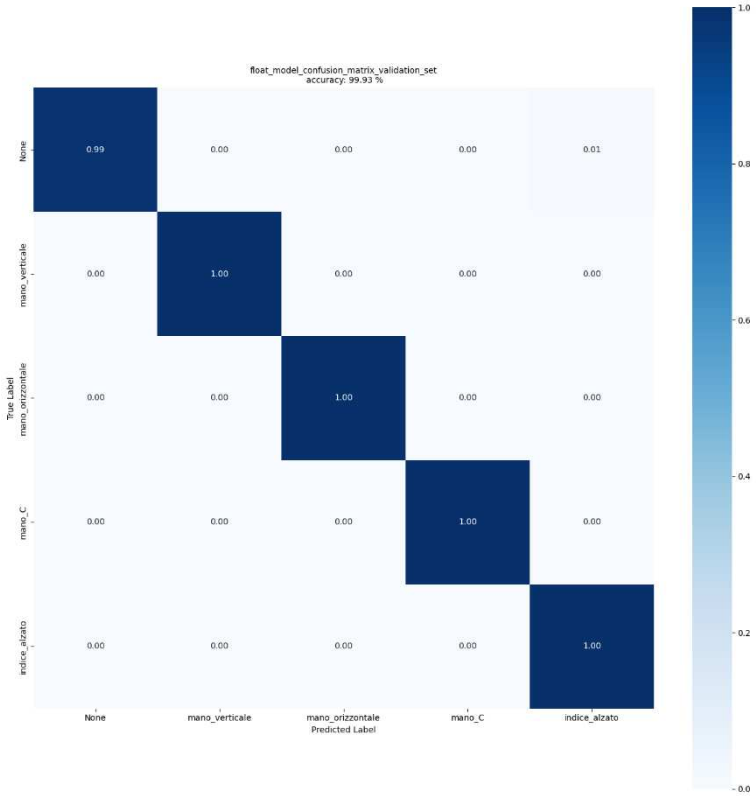
Poiché nel file di configurazione in formato .yaml il valore del parametro “EarlyStopping” è stato impostato a 40 epochs, come già spiegato il sistema ha interrotto la fase di addestramento dopo questo numero di riperizioni. In un secondo momento è stata però soppressa quella sezione di codice ed è stato eseguito un training in formato completo. Di seguito vengono riportati i risultati in formato grafico, prodotti direttamente dal programma dedicato all’addestramento alla fine del suo ciclo di operazioni:



I grafici di accuratezza e perdita della fase di addestramento

In questi grafici sono presenti i parametri denominati rispettivamente “accuracy” e “loss”. Il primo è una misura delle prestazioni del modello, che indica la proporzione di previsioni corrette rispetto al numero totale di previsioni effettuate. Il secondo, invece, misura quanto le previsioni del modello si sono discostate dai valori reali delle etichette durante la fase di addestramento. Si tratta quindi di una funzione che quantifica l’errore del modello.

Il programma ha anche prodotto la seguente immagine, la quale rappresenta una matrice di confusione:



La matrice di confusione della fase di addestramento

La matrice di confusione è uno strumento che viene utilizzato per valutare l'accuratezza e le prestazioni di un algoritmo di classificazione, e permette di visualizzare il confronto fra le previsioni del modello e i valori reali.

Nell'asse delle x sono presenti le "predicted labels", ovvero le etichette predette dal modello, mentre sull'asse delle y sono presenti le "true labels", ovvero le etichette reali presenti nel dataset. Ogni cella della matrice rappresenta in forma normalizzata (da 0 a 1) il numero di esempi che appartengono alla classe reale e sono stati predetti come appartenenti ad una determinata classe.

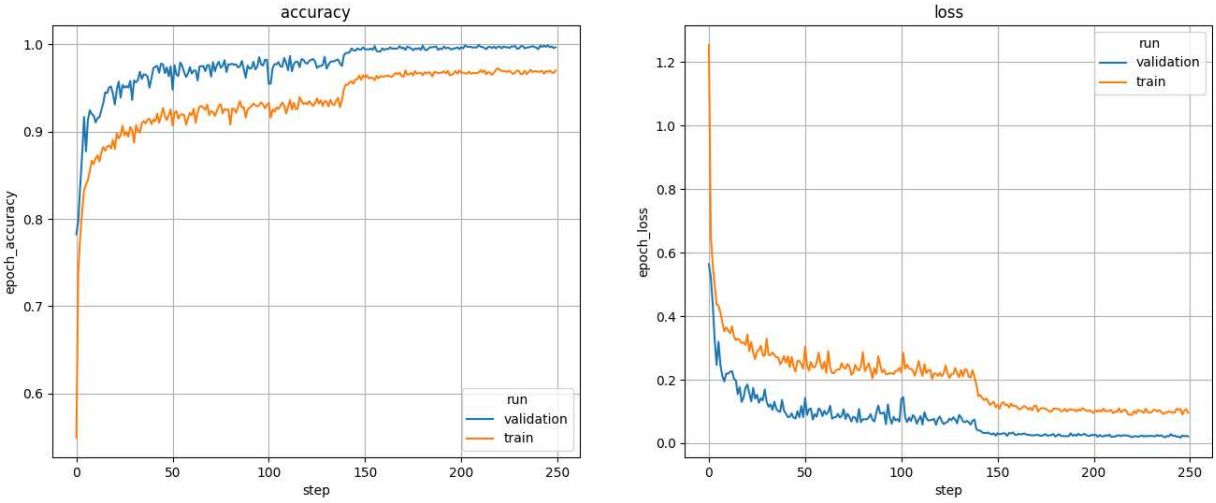
Come possiamo notare dai valori riportati nei due grafici e nella matrice di confusione, l'accuratezza del modello nel prevedere le nostre etichette è straordinariamente elevata, avvicinandosi al 100%. Questo risultato è evidenziato dal fatto che i valori "persi" o errati si collocano attorno alla soglia del millesimo, rendendoli quindi relativamente trascurabili nell'analisi complessiva delle prestazioni. Una tale precisione così elevata è

strettamente collegata al fatto che il modello preaddestrato utilizzato per questo progetto ha già conseguito prestazioni eccellenti, con un'accuratezza prossima al 100%, durante la fase di addestramento principale delle hand postures di base. Questa ha dunque permesso di generalizzare in modo efficace su nuovi dati: di conseguenza, quando il modello viene applicato al nostro specifico set di dati, riesce a mantenere un livello di accuratezza quasi perfetto, dimostrando la robustezza e l'efficacia del trasferimento di conoscenza da un compito generale a uno più specifico.

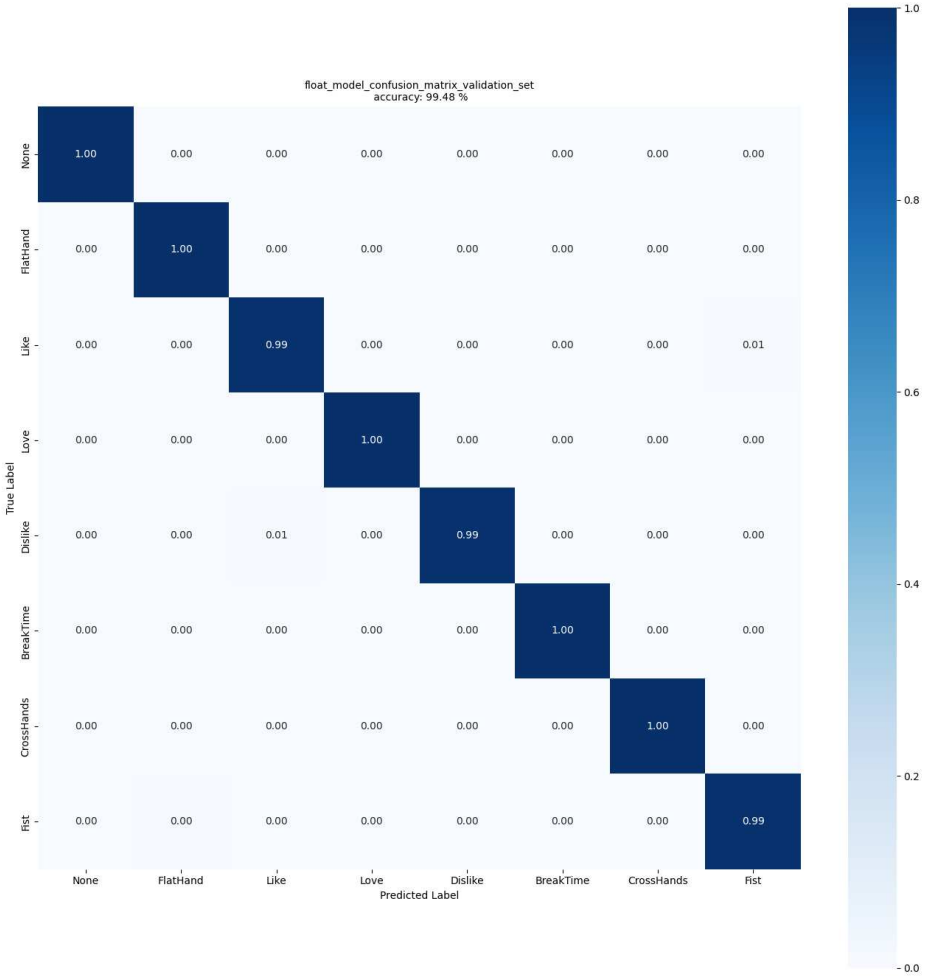
Prima di procedere con l'addestramento del nostro modello sul nuovo set di dati, abbiamo ritenuto fondamentale effettuare una fase di verifica preliminare per garantire l'integrità e il corretto funzionamento del nostro sistema. Questa fase di test è stata condotta utilizzando il dataset originale, una scelta che ci ha permesso di stabilire un punto di riferimento affidabile e di confronto diretto con i risultati noti e pubblicati.

Questa procedura ci ha permesso di confermare che il nostro sistema fosse in grado di leggere e processare correttamente i dati di input, escludendo potenziali errori nella pipeline di acquisizione o preelaborazione dei dati; abbiamo inoltre potuto verificare che tutte le configurazioni e i parametri del nostro modello fossero correttamente impostati, in linea con le specifiche originali.

Considerando che i dettagli della fase di addestramento sono stati ampiamente discussi nella sezione precedente, abbiamo scelto di non ripetere l'intera descrizione del processo, ma ci limiteremo a riportare i soli risultati, i quali sono allineati quasi perfettamente con quelli riportati sul sito ufficiale della ST.



I grafici di accuratezza e perdita della fase di addestramento col dataset originale



La matrice di confusione del dataset originale

5.2 Testing del modello

Dopo aver completato con successo il processo di addestramento del nostro modello, che viene automaticamente salvato dal sistema in formato ".h5", possiamo dunque procedere alla fase di testing. Questa fase è fondamentale per verificare l'efficacia del modello e confermare che le elevate percentuali di accuratezza ottenute durante l'addestramento siano effettivamente rappresentative delle prestazioni reali del modello su dati non visti.

Per iniziare, modifichiamo il file di configurazione, cambiando la modalità operativa da "training" a "evaluation". Questo passaggio permette al modello di essere testato sui dati senza aggiornare i pesi, garantendo così che le prestazioni siano valutate in modo corretto e imparziale. Inoltre, specifichiamo il percorso del modello appena salvato nella sezione "model_path". Questo consente al sistema di caricare il modello addestrato e utilizzarlo per la fase di test.

Una volta che tutte le impostazioni sono state verificate e confermate, siamo pronti per avviare la fase di test. Durante questa fase, il modello viene messo alla prova utilizzando un set di dati che non è stato utilizzato durante l'addestramento, permettendoci di valutare la sua capacità di generalizzare e fare previsioni accurate su nuovi dati. Il processo di test viene monitorato e visualizzato graficamente per facilitare l'interpretazione dei risultati.

```
Training and Validation dataset using a validation_split = 0.2:
114 .npz files found
Analysing data shapes: 100%|
| 114/114 [00:00<00:00, 259.43it/s]
Dataset needs 0.015 GB RAM size (0.029 GB is required during loading)
Loading 114 .npz files
100%|
| 114/114 [00:01<00:00, 94.00it/s]
Dropped 1 (over 7070) frames because they were containing np.nan
Dropped 6 (over 7069) frames because the hand is outside the Min / Max defined in us
er_config.yaml
[INFO] : Evaluating the float model using validation set...
45/45 [==============================] - 1s 3ms/step - loss: 0.0053 - accuracy: 0.99
79
[INFO] : Accuracy of float model = 99.93 %
[INFO] : Loss of float model = 0.005349797196686268
[INFO] : Evaluation complete.
```

Come possiamo osservare nella figura, anche in questa fase il modello mostra una precisione sorprendentemente elevata, tanto che il valore è anche

qui prossimo al 100%. Questo risultato è molto positivo, ma va comunque interpretato con attenzione. Un'accuratezza così alta potrebbe indicare che il modello è particolarmente ben addestrato, ma bisogna escludere la possibilità di overfitting, ovvero che abbia imparato troppo bene i dettagli del set di addestramento senza riuscire a generalizzare altrettanto bene su dati nuovi. Pertanto, sarà opportuno eseguire ulteriori analisi e validazioni, come ad esempio test su altri set di dati o con metriche diverse, per confermare la robustezza e l'affidabilità del modello.

6
Conclusioni

Nel nostro progetto, ci siamo trovati di fronte alla sfida di realizzare un dataset personalizzato, un compito che ha richiesto un approccio meticoloso e una gestione innovativa delle risorse a nostra disposizione. Fin dall'inizio, abbiamo dovuto affrontare la necessità di trovare compromessi per sfruttare al meglio le limitazioni della strumentazione disponibile, il che ha comportato una pianificazione attenta e una strategia ben definita.

Uno dei primi passi fondamentali è stato decidere quali gesti includere nel nostro dataset. Questa scelta non è stata affatto semplice, poiché dovevamo considerare una serie di fattori cruciali. Abbiamo optato per gesti che fossero semplici, intuitivi e facilmente riconoscibili, in modo da garantire che potessero essere replicati con facilità e riconosciuti in una varietà di contesti. La semplicità dei gesti selezionati non solo ha facilitato la raccolta dei dati, ma ha anche assicurato che il modello potesse riconoscere gesti con applicazioni pratiche e immediate, aumentando così la sua utilità in scenari reali.

Il processo di creazione dei gesti è stato iterativo e ha richiesto un’attenta fase di ideazione e revisione. Durante questa fase, abbiamo preso in considerazione diversi aspetti, come la chiarezza del gesto, la sua univocità rispetto ad altri gesti presenti nel dataset, e la sua rilevanza per le applicazioni previste. Ogni gesto è stato attentamente valutato per garantire che fosse distintivo e che potesse essere riconosciuto con precisione dal modello, evitando ambiguità che avrebbero potuto compromettere le prestazioni del sistema.

La raccolta dei dati è stata un'altra sfida significativa. In questa fase, abbiamo dovuto bilanciare la qualità dell'acquisizione con le capacità tecniche della nostra strumentazione. Questo ha comportato l'adattamento dei parametri di acquisizione per ottimizzare la raccolta dei dati senza

sovraccaricare il nostro hardware. Abbiamo dovuto effettuare scelte mirate per garantire che i dati raccolti fossero di alta qualità, pur rimanendo entro i limiti delle nostre risorse tecniche. Questo equilibrio è stato essenziale per garantire che il dataset fosse sufficientemente robusto da permettere al modello di apprendere in modo efficace.

Una volta completata la raccolta dei dati, siamo passati alla fase di addestramento del modello. Qui, l'uso di un dataset personalizzato ha dimostrato i suoi vantaggi in modo evidente. Il modello ha potuto apprendere da un insieme di dati specificamente curato per le nostre applicazioni, il che ha migliorato significativamente la sua capacità di generalizzazione. Questo ha ridotto il rischio di overfitting, un problema comune quando si lavora con dati troppo specifici o non rappresentativi: il risultato è stato un modello più versatile e capace di adattarsi a una varietà di situazioni reali.

L'analisi dei risultati ottenuti durante la fase di testing ha confermato il successo del nostro approccio. Il modello ha dimostrato una notevole capacità di riconoscere i gesti anche in condizioni non ideali, mostrando una robustezza che ha superato le nostre aspettative. Questo successo è stato reso possibile grazie alla cura e all'attenzione dedicate alla selezione e alla raccolta dei dati, che hanno permesso al modello di apprendere in modo efficace e di applicare correttamente ciò che aveva imparato.

Dunque, per concludere, la creazione del nostro dataset personalizzato non solo ha superato le sfide tecniche che abbiamo incontrato, ma ha anche gettato le basi per lo sviluppo di un modello di riconoscimento di hand postures versatile e affidabile. Questo modello è ora in grado di rispondere alle esigenze pratiche e innovative che ci eravamo prefissati all'inizio del progetto, dimostrando che con una pianificazione attenta e una gestione efficace delle risorse, è possibile ottenere risultati eccellenti anche in presenza di limitazioni tecniche.

7

Bibliografia

[1] Tang, A., Lu, K., Wang, Y., Huang, J., & Li, H. (2015). A real-time hand posture recognition system using deep neural networks. *ACM Transactions on Intelligent Systems and Technology (TIST)*, *6*(2), 1-23.

[2] Núñez Fernández, D., & Kwolek, B. (2018). Hand posture recognition using convolutional neural network. In *Progress in Pattern Recognition, Image Analysis, Computer Vision, and Applications: 22nd Iberoamerican Congress, CIARP 2017, Valparaíso, Chile, November 7–10, 2017, Proceedings 22* (pp. 441-449). Springer International Publishing.

[3] Chevtchenko, S. F., Vale, R. F., Macario, V., & Cordeiro, F. R. (2018). A convolutional neural network with feature fusion for real-time hand posture recognition. *Applied Soft Computing*, *73*, 748-766.

- https://www.st.com

- https://github.com/STMicroelectronics/stm32ai-modelzoo

- https://www.ibm.com/topics/convolutional-neural-networks