

# NumWiz: Real-Time Hand Gesture Number Recognition

Sergio Mancini, Enrico Sorbello, and Melo Fuccio

*Università degli Studi di Catania*

18 Luglio 2024

## Abstract

Questo progetto, sviluppato come parte del corso di Machine Learning nel Corso di Laurea Magistrale in Informatica presso l'Università di Catania, implementa tre modelli di machine learning per il riconoscimento in tempo reale dei numeri eseguiti con gesti delle mani. I modelli sono progettati per interpretare accuratamente i gesti delle dita, identificando i numeri da 0 a 10 mostrati dall'utente davanti a una webcam.

## 1 Problema

Il riconoscimento dei gesti numerici delle mani in tempo reale rappresenta una sfida significativa nell'ambito della computer vision e del machine learning. Questo problema si colloca all'intersezione tra il riconoscimento di pattern visivi e l'interpretazione semantica dei gesti.

### 1.1 Sfide Tecniche

Il riconoscimento accurato dei gesti numerici in tempo reale presenta diverse sfide:

1. **Variabilità dei Gesti:** Le persone rappresentano i numeri con le mani in modi leggermente diversi.
2. **Condizioni di Illuminazione:** L'ambiente può influenzare significativamente la qualità dell'input visivo.
3. **Velocità di Elaborazione:** Il sistema deve funzionare in tempo reale per garantire un'interazione fluida.
4. **Robustezza:** Il modello deve funzionare correttamente con mani di diverse dimensioni, colori e orientamenti.

### 1.2 Obiettivi Del Progetto

NumWiz si propone di affrontare queste sfide implementando e confrontando tre diversi approcci basati su reti neurali, ciascuno con le proprie caratteristiche e potenzialità nel riconoscimento accurato e tempestivo dei gesti numerici delle mani. Gli obiettivi di questo progetto sono:

1. Applicare le tecniche di machine learning studiate durante il corso a un problema pratico di computer vision
2. Confrontare l'efficacia di diversi approcci al riconoscimento dei gesti
3. Sviluppare un'applicazione funzionale che può essere utilizzata in scenari reali

## 2 Dataset

### 2.1 Dataset 1: Landmarks

#### 2.1.1 Acquisizione

Il primo dataset è stato acquisito utilizzando uno script personalizzato denominato `create_dataset.py`. Questo script sfrutta la libreria MediaPipe in combinazione con la webcam per catturare immagini delle mani che rappresentano numeri. Per ogni classe (corrispondente a un numero da 0 a 5), lo script acquisisce 300 frame di dimensione 400x400 pixel. Di questi, 150 frame rappresentano la mano destra e 150 la mano sinistra, garantendo una rappresentazione equilibrata di entrambe le mani.

#### 2.1.2 Elaborazione dei Dati

Durante l'acquisizione, lo script elabora ciascun frame per estrarre i landmarks della mano, che sono punti chiave che descrivono la posizione e la forma della mano. Questi landmarks vengono poi salvati in un file CSV insieme alla classe corrispondente. Ogni riga del CSV contiene quindi la classe (il numero rappresentato) e un vettore di landmarks che descrive la configurazione della mano.

#### 2.1.3 Data Augmentation

Per arricchire il dataset e migliorare la robustezza del modello, è stata implementata una tecnica di data augmentation utilizzando lo script `synthetic_data.py`. Questo script legge il file CSV generato precedentemente e, per ogni riga (cioè per ogni esempio), genera 15 nuove varianti applicando rotazioni casuali ai landmarks della mano. Questo processo aumenta significativamente la dimensione del dataset e introduce variabilità che può aiutare il modello a generalizzare meglio.

#### 2.1.4 Suddivisione Del Dataset

Per preparare il dataset all'addestramento e alla valutazione del modello, è stata utilizzata la funzione `train_test_split` della libreria scikit-learn. Questa funzione permette di dividere il dataset in modo casuale in un set di addestramento (training set) e un set di test, garantendo che entrambi i set contengano una rappresentazione bilanciata di tutte le classi.

#### 2.1.5 Esempi Visuali

class	landmarks
0	[789.7343332799564, 106.1569776273783, 790.4025315557443, ...]
2	[-633.569589000745, -314.7191430016492, -669.9106925910853, ...]
3	[215.34454599527012, 962.9875301504285, 171.3307579442555, ...]

## 2.2 Dataset 2: "Our Bare Hands"

### 2.2.1 Acquisizione

Il secondo dataset rappresenta un contributo unico e personale al progetto, sviluppato da noi tre studenti, combina immagini personalizzate (le nostre) con quelle del dataset HAGRID (HAnd Gesture Recognition Image Dataset). L'acquisizione delle immagini personalizzate è avvenuta attraverso due metodi principali:

1. **Registrazione di video:** Abbiamo registrato dei video mentre eseguivamo gesti numerici con le mani. Questi video sono stati poi elaborati utilizzando lo script `createdatasetdelay.py`, che identifica i momenti in cui una mano è visibile e estrae fotogrammi di dimensione 400x400 pixel centrati sulla mano.
2. **Acquisizione diretta:** In alternativa, è stato possibile utilizzare lo stesso approccio del primo dataset, catturando immagini direttamente dalla webcam in tempo reale.

## 2.2.2 Elaborazione e Organizzazione

Le immagini estratte sono state organizzate in cartelle separate per ciascuna classe (numero rappresentato). Questo approccio facilita la gestione del dataset e l'addestramento dei modelli che richiedono una struttura di cartelle specifica.

Per il test set, sono state utilizzate immagini dal dataset HAGRID, che presentano caratteristiche simili alle immagini personalizzate acquisite, garantendo così un test set robusto e diversificato.

## 2.2.3 Dimensioni del Dataset

Il dataset risultante contiene circa 500 immagini per l'addestramento e 500 immagini per il test, fornendo un equilibrio tra i dati di training e di test.

## 2.2.4 Suddivisione Del Dataset

Per la suddivisione del dataset in training set e di test set, sono stati utilizzati due approcci:

1. **Suddivisione manuale:** Le immagini sono state divise manualmente nelle cartelle di train e test, assicurando una distribuzione equilibrata delle classi.
2. **Utilizzo di `split_folders`:** In alternativa, è stata utilizzata la libreria `split_folders` per automatizzare il processo di divisione, garantendo una ripartizione casuale ma bilanciata dei dati.

## 2.2.5 Esempi Visuali



(a) Class "1" of the train set



(b) Class "1" of the test set

Figure 1: Images of class "1"

## 2.2.6 Organizzazione Dei File

La struttura delle cartelle del dataset è organizzata come segue:

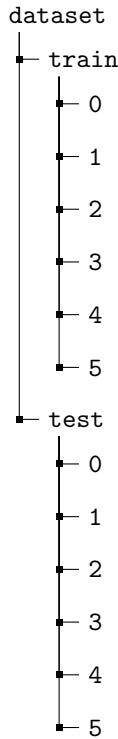


Figure 2: Struttura delle cartelle del dataset

## 2.3 Dataset 3: HAGRID (HAnd Gesture Recognition Image Dataset)

### 2.3.1 Acquisizione

Il terzo dataset utilizzato in questo progetto è basato sul dataset HAGRID (HAnd Gesture Recognition Image Dataset), una risorsa ampiamente riconosciuta nel campo del riconoscimento dei gesti delle mani. Il dataset HAGRID è stato scaricato da Kaggle, da questo [link](#), ed è una porzione dell'enorme dataset originale.

### 2.3.2 Elaborazione dei Dati

Dopo l'acquisizione del dataset HAGRID, è stato utilizzato uno script personalizzato denominato `crop_hagrid.py` per pre-processare le immagini. Questo script offre la flessibilità di selezionare il numero desiderato di immagini da utilizzare, permettendo di adattare la dimensione del dataset alle esigenze specifiche del progetto. Nel nostro caso, sono state selezionate circa 4000 immagini in totale. Lo script si occupa di:

1. Selezionare il numero specificato di immagini dal dataset HAGRID originale.
2. Individuare le mani nelle immagini selezionate.
3. Croppare le immagini attorno alle mani rilevate, generando nuove immagini di dimensione 400x400 pixel.

Questo processo di pre-processing assicura che tutte le immagini siano uniformi in termini di dimensioni e che la mano sia centrata e occupi la maggior parte dell'immagine.

### 2.3.3 Suddivisione Del Dataset

Il dataset risultante è composto da circa 4000 immagini in totale, suddivise come segue:

- Circa 3000 immagini per il training set
- Circa 1000 immagini per il test set

Questa suddivisione fornisce un ampio set di dati per l'addestramento del modello, garantendo al contempo un set di test significativo per valutare le prestazioni.

### 2.3.4 Suddivisione Del Dataset

Per la suddivisione del dataset in training set e di test set, è stato utilizzato l'approccio di `split_folders` per garantire una ripartizione casuale e bilanciata delle cartelle contenenti le immagini del dataset.

### 2.3.5 Esempi Visuali



(a) Class "2" of the train set



(b) Class "2" of the test set

Figure 3: Images of class "2" of Hagrid Dataset

### 2.3.6 Vantaggi del Dataset HAGRID

L'utilizzo del dataset HAGRID offre diversi vantaggi:

1. Alta qualità e diversità delle immagini, che coprono una vasta gamma di gesti e condizioni di illuminazione.
2. Un gran numero di esempi, che permette un addestramento più robusto del modello.
3. Dataset ampiamente riconosciuto nella comunità di ricerca, facilitando il confronto con altri modelli e approcci.

Questo dataset fornisce una base solida per l'addestramento e la valutazione del nostro modello di riconoscimento dei gesti numerici delle mani, offrendo potenzialmente una migliore generalizzazione e robustezza rispetto ai dataset acquisiti localmente.

## 3 Metodi

In questo progetto, abbiamo esplorato tre approcci distinti per il riconoscimento dei gesti numerici delle mani. Ciascun metodo è stato scelto per le sue caratteristiche uniche e per il potenziale di affrontare il problema da diverse prospettive. Di seguito, presentiamo i dettagli tecnici di ciascun metodo, motivando le scelte effettuate.

### 3.1 Metodo 1: Multi-Layer Perceptron (MLP)

#### 3.1.1 Descrizione

Il primo approccio utilizza un semplice Multi-Layer Perceptron (MLP) per classificare i gesti delle mani basandosi sui landmarks estratti. Questo metodo sfrutta la rappresentazione compatta e ricca di informazioni fornita dai landmarks della mano, permettendo una classificazione efficiente e computazionalmente leggera.

#### 3.1.2 Architettura MLP

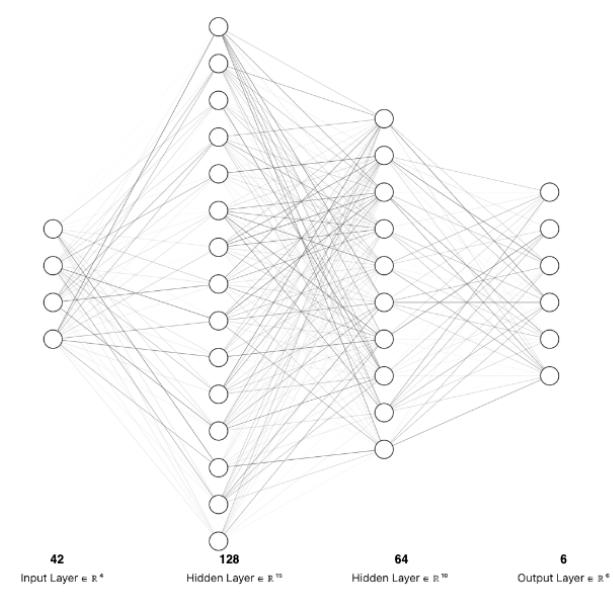


Figure 4: Architettura del Multi-Layer Perceptron per la classificazione dei landmarks della mano

La Figura 4 illustra l'architettura del nostro MLP. Come si può vedere, il modello consiste in tre strati fully connected:

- **Input layer:** 42 neuroni (corrispondenti ai landmarks della mano)
- **Hidden layer 1:** 128 neuroni con attivazione ReLU
- **Hidden layer 2:** 64 neuroni con attivazione ReLU
- **Output layer:** 6 neuroni (corrispondenti alle 6 classi di gesti)

### 3.1.3 Motivazione

La scelta di un MLP per questo task è motivata da:

1. Semplicità e velocità di addestramento
2. Efficacia nel gestire input di dimensioni fisse (i landmarks)
3. Capacità di catturare relazioni non lineari tra i landmarks

## 3.2 Metodo 2 e 3: ResNet18 e ResNet34 con Dataset Personalizzato e Hagrid

### 3.2.1 Descrizione

Il secondo ed il terzo approccio utilizzano modelli ResNet pre-addestrati (ResNet18 e ResNet34) adattati al nostro caso d'uso specifico. La differenza principale tra questi due metodi risiede nel dataset utilizzato per il fine-tuning:

- **Metodo 2:** Utilizza il dataset personalizzato descritto nella sezione Dataset 2.
- **Metodo 3:** Utilizza il dataset HAGRID descritto nella sezione Dataset 3.

### 3.2.2 Architettura

Per entrambi i metodi, abbiamo utilizzato le implementazioni di ResNet18 e ResNet34 fornite da PyTorch, modificando lo strato finale per adattarlo al nostro numero di classi.

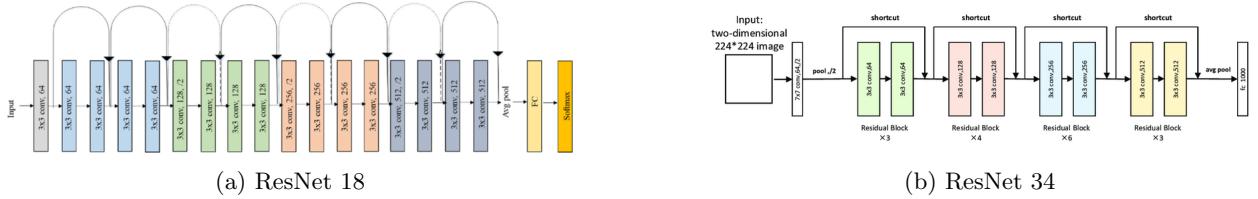


Figure 5: Le architetture generali di ResNet

La Figura 5 mostra le architetture generali di ResNet che abbiamo utilizzato.

### 3.2.3 Motivazione

La scelta di utilizzare le architetture ResNet18 e ResNet34 per entrambi i metodi è motivata da diversi fattori:

1. **Architettura profonda e efficiente:** Le ResNet permettono l'addestramento di reti neurali molto profonde in modo efficace, grazie alle connessioni residuali che mitigano il problema della scomparsa del gradiente.
2. **Trasferimento dell'apprendimento:** Essendo pre-addestrate su vasti dataset come ImageNet, le ResNet consentono di sfruttare l'apprendimento pregresso e adattarlo al nostro compito specifico attraverso il fine-tuning.

- 3. Prestazioni state-of-the-art:** Le ResNet hanno dimostrato prestazioni eccellenti in una vasta gamma di compiti di computer vision, inclusi quelli che richiedono il riconoscimento di dettagli fini e strutture complesse.

### 3.2.4 Differenze tra Metodo 2 e Metodo 3

La principale differenza tra questi due metodi risiede nel dataset utilizzato per il fine-tuning:

- **Metodo 2:** Utilizza un dataset personalizzato, che potrebbe essere più specifico per il nostro caso d'uso particolare ma potenzialmente limitato in termini di varietà e quantità di dati.
- **Metodo 3:** Sfrutta il dataset HAGRID, che offre una maggiore diversità e quantità di dati, potenzialmente migliorando la generalizzazione del modello e permettendo un confronto con altri approcci che utilizzano lo stesso dataset.

## 4 Valutazione

Per valutare in modo completo e obiettivo le prestazioni dei nostri modelli, abbiamo utilizzato diverse metriche di valutazione. Queste metriche ci permettono di analizzare vari aspetti delle prestazioni dei modelli, fornendo una visione completa della loro efficacia. Le metriche principali utilizzate sono:

### 4.1 Accuracy

L'accuracy è la misura più intuitiva e rappresenta la proporzione di previsioni corrette (sia positive che negative) sul numero totale di casi esaminati.

$$\text{Accuracy} = \frac{\text{Numero di previsioni corrette}}{\text{Numero totale di previsioni}} \quad (1)$$

Questa metrica fornisce una visione generale delle prestazioni del modello, ma può essere fuorviante in caso di dataset sbilanciati.

### 4.2 Precision

La precision misura la proporzione di identificazioni positive corrette rispetto al totale delle identificazioni positive effettuate dal modello.

$$\text{Precision} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Positives}} \quad (2)$$

Questa metrica è particolarmente utile quando il costo dei falsi positivi è alto.

### 4.3 Recall

Il recall, noto anche come sensibilità, misura la proporzione di positivi reali che sono stati identificati correttamente.

$$\text{Recall} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Negatives}} \quad (3)$$

Il recall è importante quando il costo dei falsi negativi è alto.

### 4.4 F1-Score

L'F1-Score è la media armonica di precision e recall, fornendo un singolo punteggio che bilancia entrambe le metriche.

$$\text{F1-Score} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}} \quad (4)$$

L'F1-Score è particolarmente utile quando si cerca un equilibrio tra precision e recall, e quando si ha una distribuzione di classi non uniforme.

## 4.5 Implementazione

Per il calcolo di queste metriche, abbiamo utilizzato le funzioni fornite dalla libreria scikit-learn, che garantiscono un'implementazione standardizzata e affidabile. In particolare, abbiamo fatto uso delle funzioni `accuracy_score`, `precision_score`, `recall_score`, e `f1_score`.

## 5 Esperimenti

In questa sezione, presentiamo gli esperimenti condotti per valutare e confrontare le prestazioni dei tre metodi proposti per il riconoscimento dei gesti numerici delle mani. Descriviamo i parametri utilizzati, le prove effettuate e discutiamo i risultati ottenuti.

### 5.1 Setup

Per ciascun metodo, abbiamo condotto una serie di esperimenti per ottimizzare i parametri e valutare le prestazioni. Tutti gli esperimenti sono stati eseguiti su Tesla P100 (Nvidia GPU).

#### 5.1.1 Metodo 1: MLP

Parametri principali:

- Dimensione del batch: 32
- Numero di epoche: 20
- Ottimizzatore: SGD con learning rate 0.001
- Funzione di loss: Cross Entropy Loss

##### 5.1.1.1 Andamento dell'Addestramento

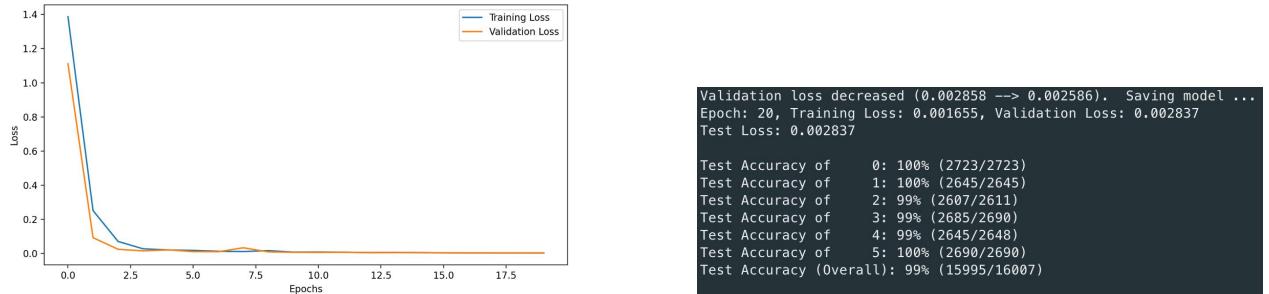


Figure 6: Training e Accuracy MLP

La Figura 6 mostra l'andamento del training per questo primo modello.

#### 5.1.2 Metodo 2 e 3: ResNet18 e ResNet34

- Dimensione del batch: 512 (ResNet18) e 256 (ResNet34)
- Numero di epoche: 100 (Metodo 2) e 10 (Metodo 3)
- Ottimizzatore: Adam con learning rate 0.0001
- Funzione di loss: Cross Entropy Loss

### 5.1.2.1 Andamento dell'Addestramento Metodo 2

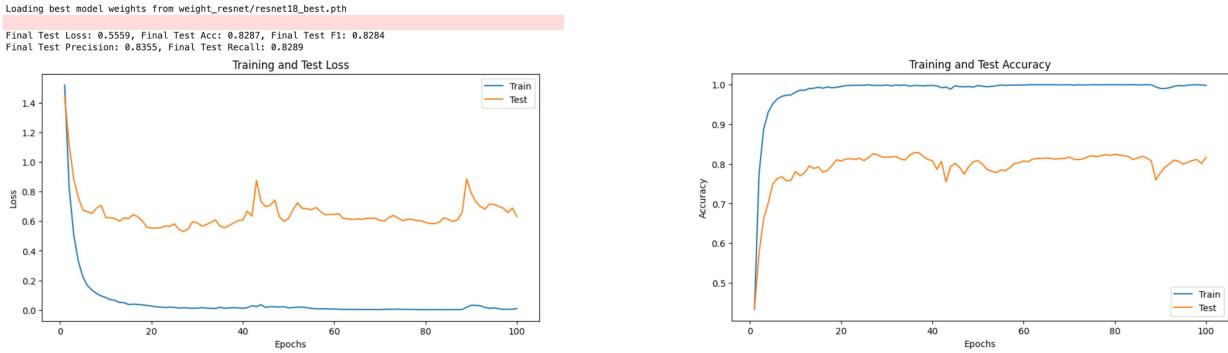


Figure 7: Training e Accuracy Metodo 2 - ResNet18

La Figura 7 nella prima parte mostra l'andamento del training per il modello del Metodo 2, si nota un rapido decremento nelle prime epoche, sia per il training che per il test set. Tuttavia, si osserva un gap persistente tra le due curve, con la loss di training che converge a valori molto bassi (quasi zero) mentre la loss di test si stabilizza intorno a 0.6. Questo suggerisce un certo grado di overfitting.

Nella seconda parte invece, vengono mostrate le curve di accuracy, quella di training raggiunge rapidamente valori prossimi al 100%, mentre l'accuracy di test si stabilizza intorno all'80-82%. Anche questo grafico evidenzia una discrepanza tra performance di training e test, confermando la presenza di overfitting.

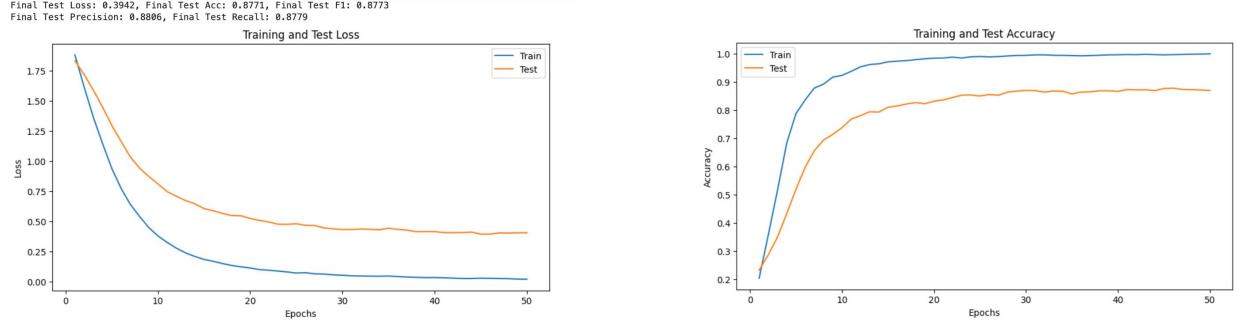


Figure 8: Training e Accuracy Metodo 2 - ResNet18

La Figura 16 nella prima parte mostra l'andamento del training per il modello del Metodo 2, si può notare un decremento rapido della loss nelle prime epoche per entrambi i set. La loss di training converge a valori prossimi allo zero, mentre la loss di test si stabilizza intorno a 0.39. Sebbene sia presente un certo grado di overfitting, evidenziato dal gap tra le curve di training e test in entrambi i grafici, questo risulta meno pronunciato rispetto al modello ResNet18 precedentemente analizzato.

Nella seconda parte invece, vengono mostrate le curve di accuracy, si nota un rapido aumento dell'accuracy nelle prime epoche, con l'accuracy di training che raggiunge rapidamente valori prossimi al 100%, mentre l'accuracy di test si stabilizza intorno all'87-88%.

### 5.1.2.2 Andamento dell'Addestramento Metodo 3

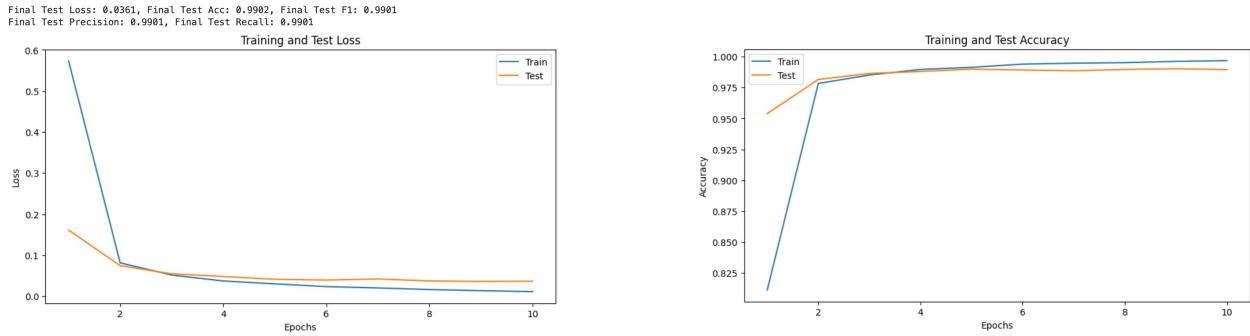


Figure 9: Training e Accuracy Metodo 3 - ResNet18

La Figura 9 nella prima parte mostra l'andamento del training per il modello del Metodo 3, la loss diminuisce velocemente nelle prime epoche e si stabilizza su valori molto bassi, con una differenza minima tra training e test, suggerendo un basso livello di overfitting.

Nella seconda parte invece, vengono mostrate le curve di accuracy che raggiunge rapidamente valori molto alti, superiori al 99% sia per il training che per il test, indicando un'eccellente capacità di classificazione.

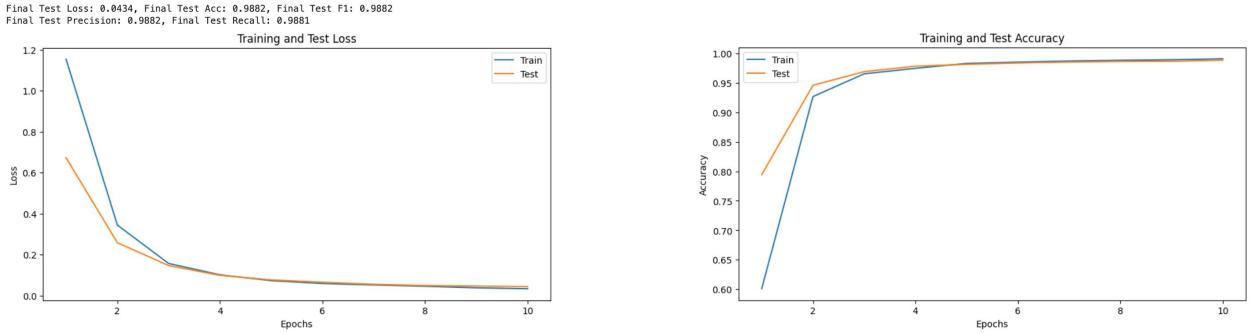


Figure 10: Training e Accuracy Metodo 3 - ResNet34

La Figura 16 nella prima parte mostra l'andamento del training per il modello del Metodo 3, la loss diminuisce rapidamente e si stabilizza su valori molto bassi, con una differenza minima tra training e test, indicando un ottimo fitting del modello senza segni evidenti di overfitting.

Nella seconda parte invece, vengono mostrate le curve di accuracy che raggiunge rapidamente valori estremamente alti, sfiorando il 100% sia per il training che per il test dopo poche epoche.

## 5.2 Metriche Di Valutazione

Questa tabella mostra i vari valori delle metriche e delle curve di training, dei tre metodi proposti:

Table 1: Risultati dei tre modelli

Model	Accuracy	Precision	Recall	F1 Score	Train Loss	Test Loss
MLP	0.99	1.00	1.00	1.00	0.001655	<b>0.002837</b>
ResNet18 (Metodo 2)	0.8396	0.8424	0.8397	0.8387	0.0055	0.5148
ResNet34 (Metodo 2)	0.8786	0.8814	0.8791	0.8737	0.0222	0.3932
ResNet18 (Metodo 3)	<b>0.9910</b>	0.9910	0.9910	0.9910	0.0129	0.0341
ResNet34 (Metodo 3)	0.9893	0.9892	0.9892	0.9892	0.0343	0.0418

### 1. MLP (Multi-Layer Perceptron):

- Mostra prestazioni eccezionali con accuracy, precision, recall e F1 score tutti pari a 1.00 o 0.99.
- Ha le perdite (loss) più basse sia in fase di training (0.001655) che di test (0.002837).
- Sembra essere il modello più performante tra quelli presentati, almeno sui dati forniti.

### 2. ResNet (Metodo 2):

- ResNet34 supera ResNet18 in tutte le metriche di performance.
- Entrambi i modelli mostrano risultati inferiori rispetto all'MLP e al Metodo 3.
- La differenza tra train loss e test loss è notevole, suggerendo un possibile overfitting.

### 3. ResNet (Metodo 3):

- Mostra prestazioni molto elevate, seconde solo all'MLP.
- ResNet18 supera leggermente ResNet34 in questo metodo.
- La differenza tra train loss e test loss è più contenuta rispetto al Metodo 2, indicando un miglior bilanciamento tra fitting e generalizzazione.

## 5.3 Analisi Metriche Di Valutazione

Dall'analisi dei risultati, possiamo trarre le seguenti conclusioni:

### 1. MLP:

### 2. ResNet su Dataset Personalizzato:

### 3. ResNet su Dataset Hagrid:

## 5.4 Analisi Matrici Di Confusione

### 5.4.1 MLP

La Figura 11 mostra il modello del Metodo 1 che presenta un'accuratezza quasi perfetta per tutte le classi, con la maggior parte delle predizioni concentrate sulla diagonale principale.

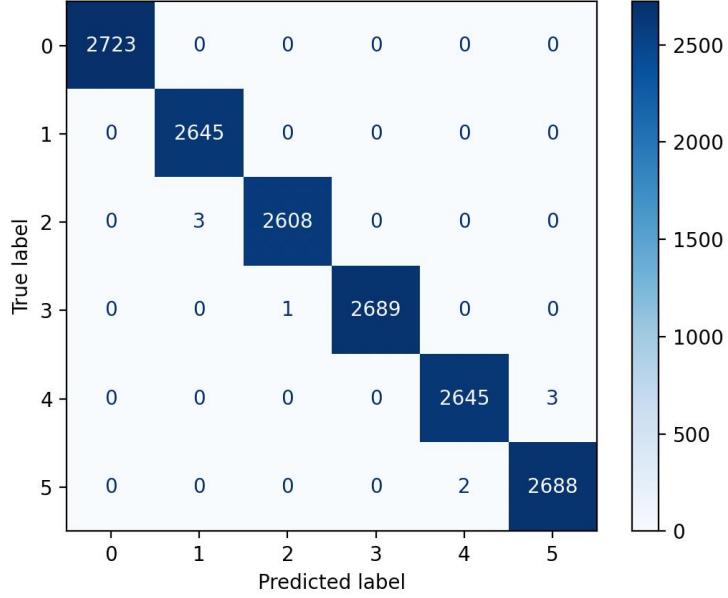


Figure 11: Confusion Matrix - MLP

#### 5.4.2 ResNet18 e ResNet34 del Metodo 2

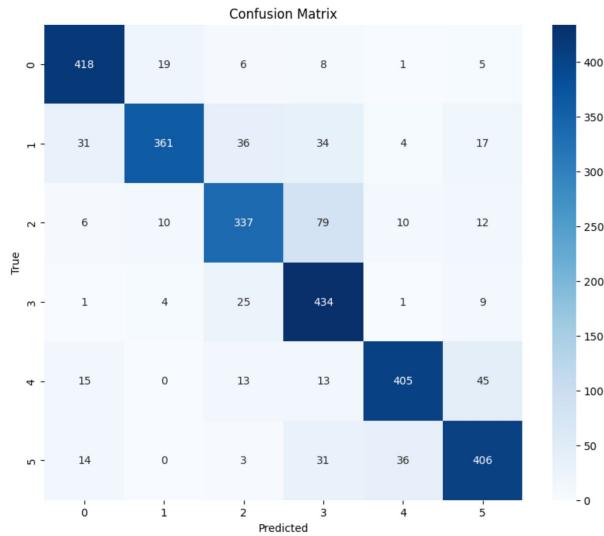


Figure 12: Confusion Matrix - ResNet18

La Figura 12 mostra il modello del Metodo 2 (ResNet18), si osserva una buona capacità di classificazione per tutte le classi, con la maggior parte delle predizioni corrette sulla diagonale principale. Tuttavia, si notano alcune confusione:

- La classe 1 viene spesso confusa con la classe 0 e 3
- C'è una notevole confusione tra le classi 2 e 3
- La classe 4 mostra alcune false classificazioni come classe 5

La Figura 13 mostra il modello del Metodo 2 (ResNet34), anch'esso ha una buona capacità di classificazione molto simile alla ResNet18.

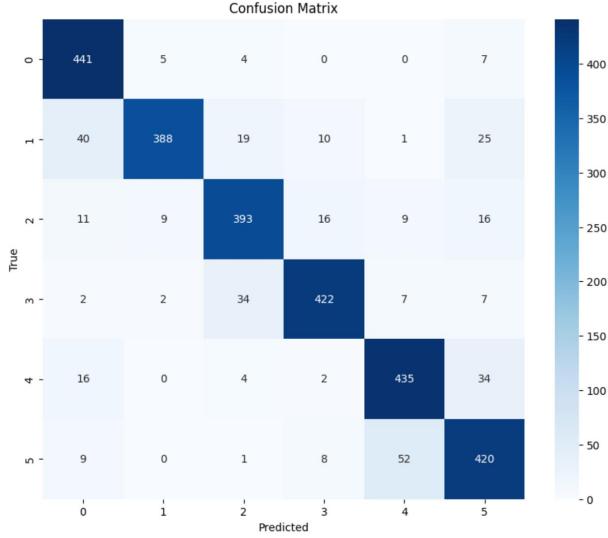


Figure 13: Confusion Matrix - ResNet34

#### 5.4.3 ResNet18 e ResNet34 del Metodo 3

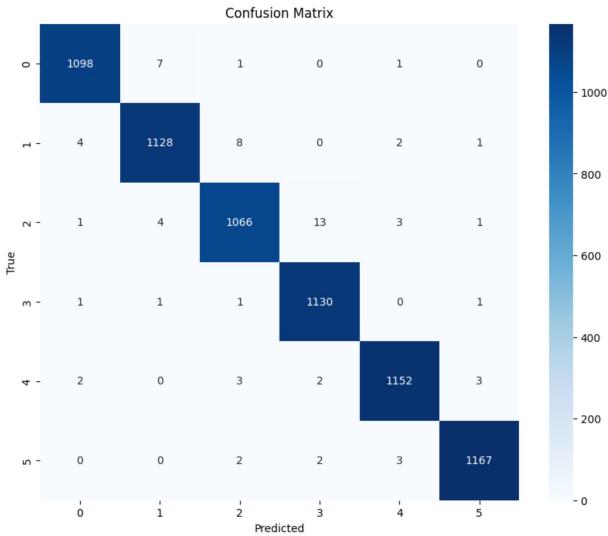


Figure 14: Confusion Matrix - ResNet18

La matrice di confusione della Figura 14 fornisce un’ulteriore conferma delle eccellenti prestazioni del modello. La concentrazione delle predizioni sulla diagonale principale indica un’alta accuratezza nella classificazione per tutte le classi di gesti.

Nella Figura 15 si nota una forte concentrazione delle predizioni sulla diagonale principale indicando un’accuratezza quasi perfetta nella classificazione per tutte le classi di gesti, con pochissimi errori di classificazione.

Questi grafici forniscono una visione più dettagliata e intuitiva delle prestazioni dei nostri modelli, complementando l’analisi numerica presentata nelle sezioni precedenti.

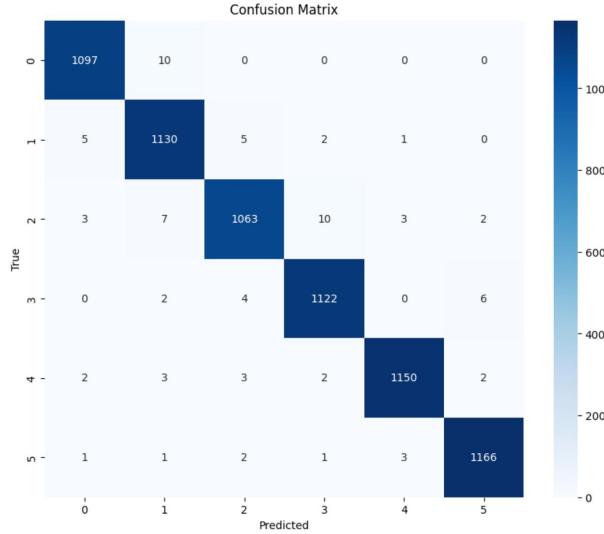


Figure 15: Confusion Matrix - ResNet34

## 6 Demo

Per dimostrare l'efficacia e l'applicabilità pratica dei nostri modelli di riconoscimento dei gesti numerici delle mani, abbiamo sviluppato tre diverse demo. Ciascuna demo illustra l'implementazione e l'utilizzo di uno dei metodi descritti nelle sezioni precedenti.

### 6.1 Demo 1: MLP Con Landmark Delle Mani

Questa demo si concentra sull'utilizzo di un Multi-Layer Perceptron (MLP) per il riconoscimento dei gesti numerici delle mani in tempo reale.

#### 6.1.1 Funzionalità

- Utilizza la libreria MediaPipe per il rilevamento, il tracciamento ed il salvataggio dei landmark delle mani
- Implementa un modello MLP precedentemente addestrato, per poter classificare le pose delle mani
- Elabora il flusso video dalla webcam in tempo reale
- Visualizza una bounding box intorno alla mano rilevata con la classe predetta
- Supporta il riconoscimento di più mani contemporaneamente, sommando i valori predetti per ciascuna mano (per rendere la fase di training computazionalmente più veloce e leggera)

#### 6.1.2 Utilizzo

1. Eseguire lo script `hand_landmarks.py`
2. Posizionare la mano (o le mani) di fronte alla webcam
3. Osservare la predizione in tempo reale sulla schermata

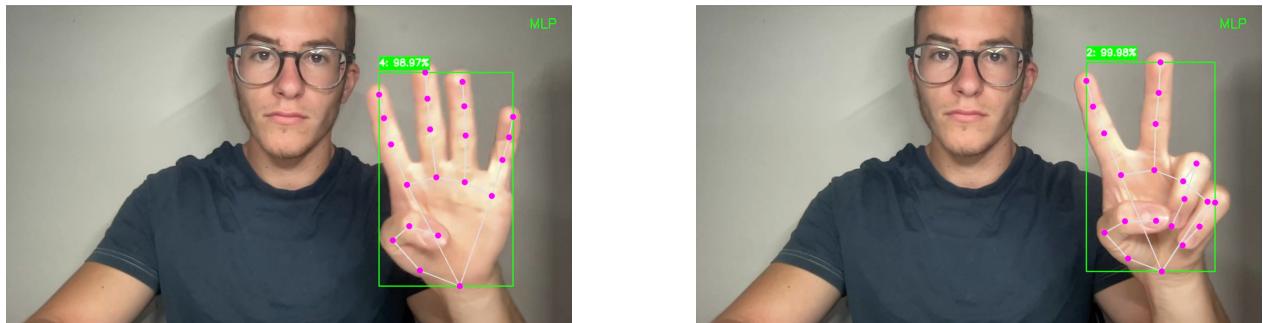


Figure 16: Demo MLP con una mano

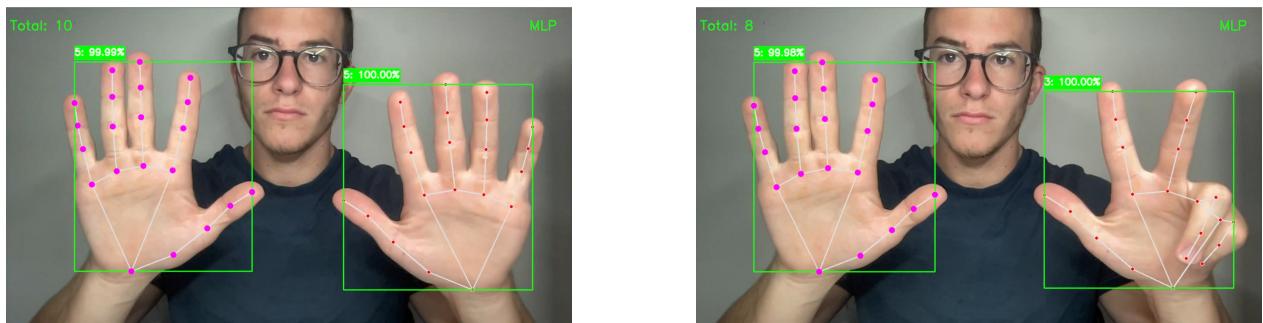


Figure 17: Demo MLP con due mani

## 6.2 Demo 2: ResNet18 e ResNet34 Con Dataset Personalizzato

Questa demo mostra l'applicazione delle reti neurali ResNet18 e ResNet34 addestrate sul nostro dataset personalizzato.

### 6.2.1 Funzionalità

- Utilizza modelli ResNet18 e ResNet34 pre-addestrati
- Carica i pesi ottimizzati da `resnet18_best.pth` e `resnet34_best.pth`
- Elabora immagini statiche per la classificazione dei gesti.

### 6.2.2 Utilizzo

1. Eseguire gli script `hand_resnet18_our.py` o `hand_resnet34_our.py`
2. Fornire un'immagine di input al modello
3. Visualizzare la predizione del modello



Figure 18: Demo ResNet18 su dataset personalizzato

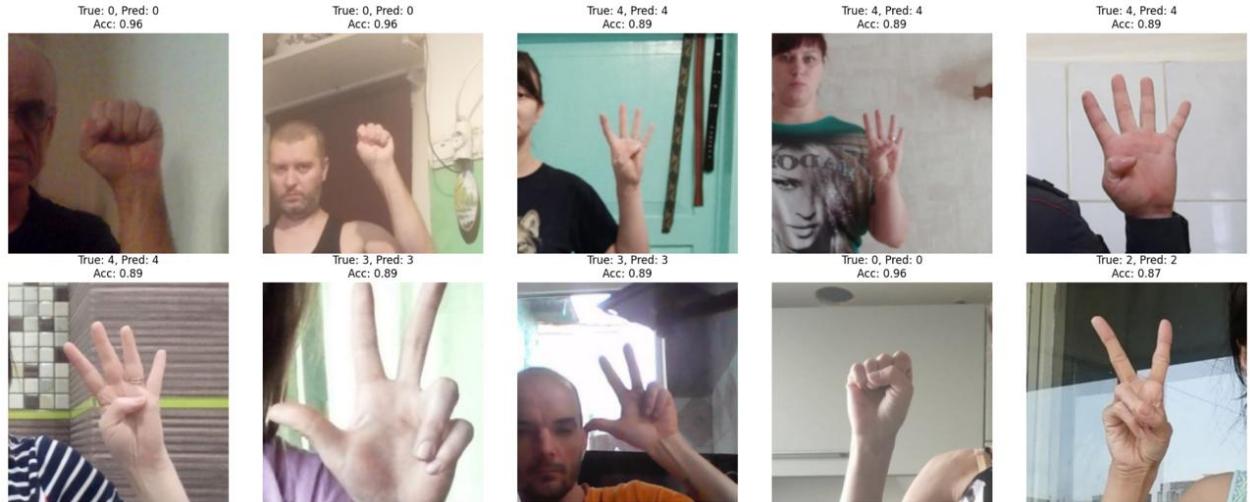


Figure 19: Demo ResNet34 su dataset personalizzato

### 6.2.3 Versione Live

Abbiamo anche sviluppato una versione in real time di questa demo:

- Funziona in modo simile alla Demo 1, ma utilizza i modelli ResNet
- Elabora il flusso video dalla webcam in real time
- Visualizza la predizione sovrapposta all'immagine della webcam.

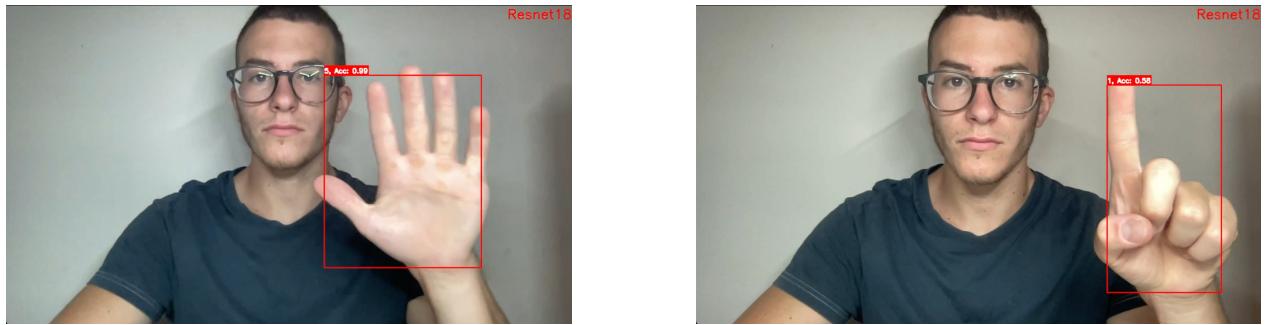


Figure 20: Demo ResNet18 con una mano

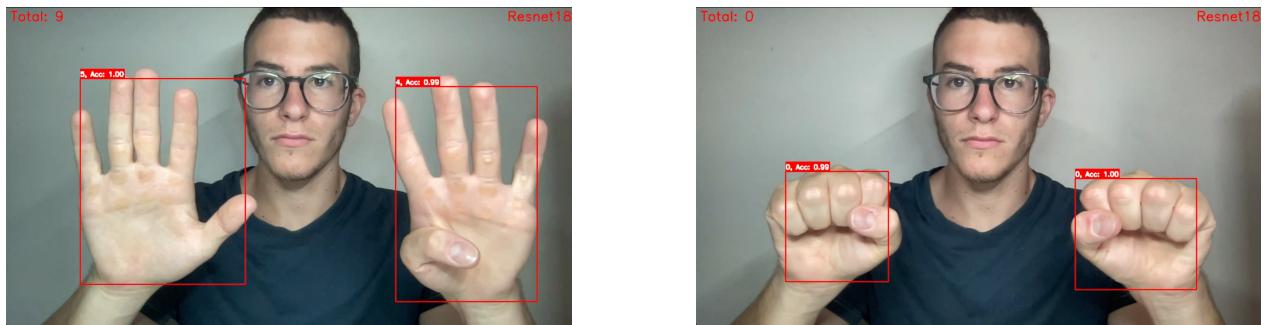


Figure 21: Demo ResNet18 con due mani

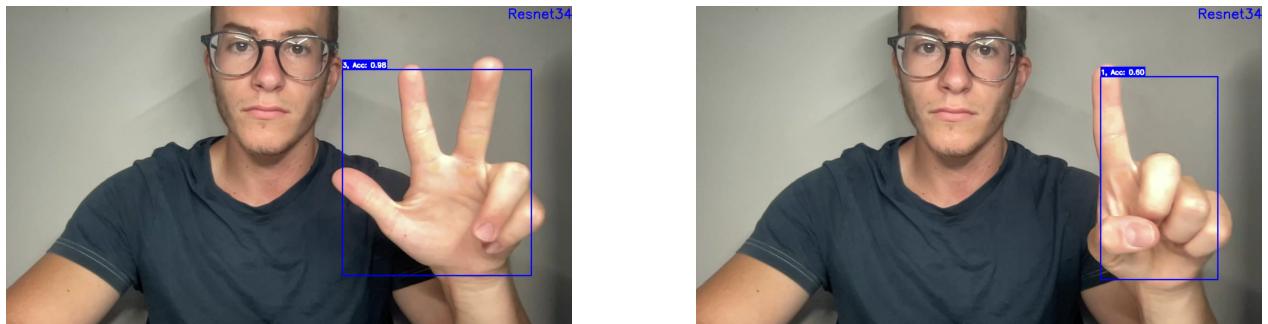


Figure 22: Demo ResNet34 con una mano

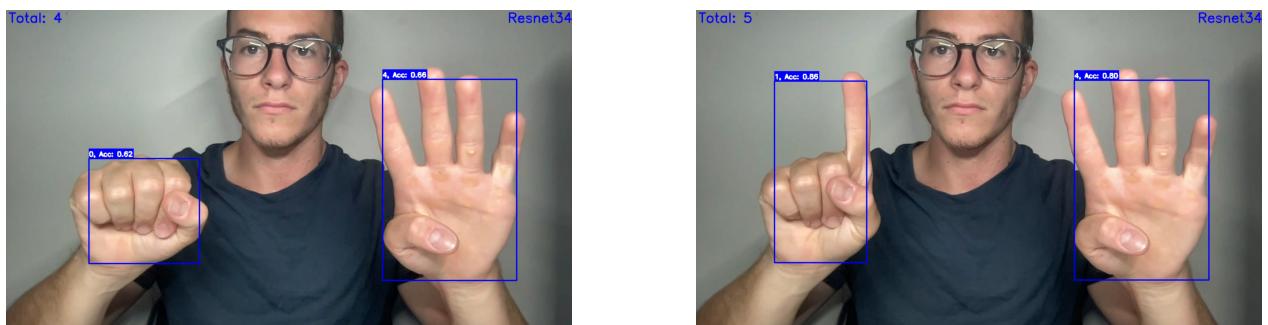


Figure 23: Demo ResNet34 con due mani

### 6.3 Demo 3: ResNet18 e ResNet34 Con Dataset HAGRID

Questa demo è simile alla Demo 2, ma utilizza modelli addestrati sul dataset HAGRID.

### 6.3.1 Funzionalità

- Identiche alla Demo 2, ma con modelli ottimizzati per il dataset HAGRID, cioè `resnet18_hagrid_best.pth` e `resnet18_hagrid_best.pth`

### 6.3.2 Utilizzo

- Eseguire gli script `hand_resnet18_hagrid.py` o `hand_resnet34_hagrid.py`
- Fornire un'immagine di input al modello
- Visualizzare la predizione del modello



Figure 24: Demo ResNet18 su dataset Hagrid

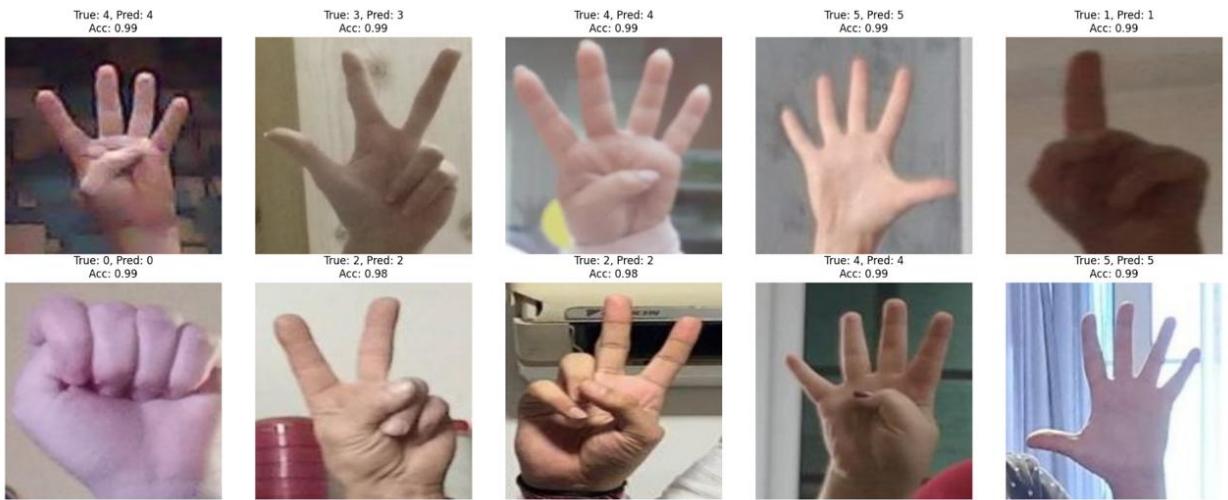


Figure 25: Demo ResNet34 su dataset Hagrid

### 6.3.3 Versione Live

Anche per questa demo è disponibile una versione in real time, identica alla Demo 2

## 7 Codice

### 7.1 MLP

Per la creazione del dataset, nella directory `NumWiz/script_dataset/` troviamo due script fondamentali:

- `create_dataset.py` cattura 150 frame dalla webcam per entrambe le mani, sinistra e destra, e salva i valori dei landmarks ottenuti tramite Mediapipe in un file CSV chiamato `hand.csv`. Questo file contiene due colonne: una per la classe e una per i landmarks. Successivamente, per incrementare il numero di valori dei landmarks e migliorare il dataset si utilizza lo script per fare data augmentation
- `synthetic_data.py` prende il file `hand.csv` e, per ogni riga, esegue 15 rotazioni casuali, aumentando così la varietà dei dati disponibili per l'addestramento.

### 7.2 Training Del Modello e Rilevamento Mani

Nella directory `NumWiz/` troviamo le cartelle `Training/` e `Testing/`, ciascuna contenente script specifici per il training ed il testing del modello

Lo script `train\csv.py` si occupa dell'addestramento di un semplice MLP utilizzando il dataset sintetico precedentemente creato. I dati vengono preparati per il modello convertendo i punti di riferimento da stringhe a liste numeriche e separando le features (X) dalle labels (y). Il dataset viene poi suddiviso in set di training e di test tramite la funzione `train_test_split`.

I dati vengono convertiti in tensori, il formato utilizzato da PyTorch per l'elaborazione, e i DataLoader vengono creati per gestire i batch durante l'addestramento e il test. Viene importato il modello MLP, chiamato **HandModel**, da un file esterno, definendo la funzione di loss (CrossEntropyLoss) e l'optimizer (SGD).

Il modello viene addestrato per un numero specifico di epochhe, durante le quali viene valutato sia sui dati di training che su quelli di test. Le perdite di addestramento e di validazione vengono graficate per monitorare le prestazioni del modello durante l'intero processo.

Lo script `MLP_CCSV.py` definisce un MLP a più strati con tre layer completamente connessi. Nel costruttore (`__init__`), vengono creati tre strati lineari con dimensioni specifiche (spiegato precedentemente).

Il metodo `forward` definisce come i dati attraversano la rete, applicando la funzione di attivazione ReLU ai primi due strati e restituendo l'output dallo strato finale.

### 7.3 Testing dell'MLP

Lo script `hand_landmarks.py` si occupa di rilevare le mani tramite la webcam e classificare i gesti utilizzando il modello addestrato.

La classe `HandDetector` gestisce il rilevamento delle mani e la classificazione dei gesti. Nel costruttore vengono inizializzate le variabili per il rilevamento delle mani con Mediapipe e viene caricato il modello di classificazione delle mani addestrato.

Il metodo `findHands` rileva le mani, disegna i punti di riferimento e classifica i gesti utilizzando il modello caricato, calcolando e visualizzando anche la confidenza della classificazione. Il metodo `findPosition` restituisce le coordinate dei punti di riferimento di una mano specifica in un'immagine. Infine, la funzione `main` avvia la webcam, utilizza la classe `HandDetector` per rilevare e classificare i gesti delle mani in tempo reale, visualizzando i risultati sullo schermo.

## 7.4 ResNet su Dataset Personalizzato e Dataset Hagrid

### 7.4.1 ResNet 18

Nello script della resnet vengono definite delle trasformazioni per la pre-elaborazione delle immagini. La classe `HandDataset` viene definita per caricare le immagini e le relative etichette dai file JSON. Successivamente vengono creati i dataset di training e test utilizzando la classe menzionata precedentemente e verranno creati i `DataLoader` per il training e test, che gestiranno il batching e lo shuffling dei dati.

Dopo aver caricato il modello della ResNet18 preaddestrato (Pytorch) l'ultimo strato della rete (completamente connesso) viene sostituito per adattarsi al numero di classi presenti nel nostro dataset. Viene anche definita la `CrossEntropyLoss` come funzione di loss e l'ottimizzatore `Adam`.

L'addestramento del modello viene controllato definendo un ciclo di addestramento di 100 epoche e se l'accuracy sul set di test migliora, i pesi verranno poi salvati.

Infine viene caricato il modello con le migliori prestazioni ottenute durante l'addestramento.

### 7.4.2 ResNet 34

Lo script implementa una ResNet34 per il riconoscimento di immagini. Definisce trasformazioni per il pre-processing, una classe `HandDataset` per caricare dati e etichette, e crea `DataLoader` per training e test. Il modello ResNet34 preaddestrato viene modificato nell'ultimo strato per adattarsi al numero di classi del nostro dataset. Si utilizzano `CrossEntropyLoss` e ottimizzatore `Adam`. L'addestramento avviene attraverso una funzione dedicata, su 50 epoche.

Una funzione di valutazione misura le prestazioni sul set di test (accuracy, F1 score, precisione, richiamo). I migliori pesi vengono salvati. Infine, si carica il modello ottimale e si visualizzano i risultati finali, includendo grafici di loss, accuracy e F1 score durante il training, oltre alla matrice di confusione e esempi di predizioni su immagini di test.

## 8 Conclusione

In conclusione, questo progetto si è focalizzato sullo sviluppo di tre modelli di intelligenza artificiale per il riconoscimento dei gesti numerici eseguiti con le mani, utilizzando tecniche di supervised learning e computer vision. Sono stati impiegati diversi dataset, tra cui il dataset ampiamente utilizzato Hagrid e un dataset creato internamente da noi che ha arricchito la diversità e la specificità delle immagini utilizzate per l'addestramento.

Il processo di sviluppo ha incluso l'utilizzo di MediaPipe per il rilevamento dei punti di riferimento delle mani e la creazione di un dataset annotato attraverso l'esperienza di registrazione video e l'annotazione manuale. Per migliorare la capacità predittiva del modello, è stata applicata la data augmentation, aumentando significativamente la variabilità dei dati di training.

Sono stati eseguiti esperimenti con due tipologie di modelli: un MLP (Multi-Layer Perceptron) e una CNN (Convolutional Neural Network). Il MLP è stato addestrato su vettori di landmarks estratti dalle immagini delle mani, mentre la CNN è stata addestrata utilizzando le immagini che abbiamo raccolto.

Durante il progetto, abbiamo imparato l'importanza della diversità dei dati e dell'annotazione accurata per migliorare l'affidabilità del modello. Inoltre, abbiamo apprezzato il ruolo cruciale della data augmentation nel migliorare la generalizzazione del modello, rendendolo più robusto alle variazioni nelle condizioni di acquisizione delle immagini.

Per migliorare il metodo proposto, potremmo considerare alcune azioni:

1. **Espansione dei dataset:** Integrare ulteriori dataset per includere una maggiore variabilità nelle pose delle mani e nelle condizioni ambientali.
2. **Ottimizzazione dei modelli:** Esplorare architetture più complesse per la CNN e migliorare la profondità e la complessità del MLP per ottenere migliori performance.
3. **Esplorazione di altre tecniche di augmentation:** Esplorare altre tecniche di augmentation oltre alla rotazione, come il trasferimento di dominio o l'introduzione di rumore.