

Identificazione del linguaggio dei segni tramite l'uso di immagini.

Luigi Priano (1000002081), Gianluca Giardina (1000015165),
Giuseppe Napoli (1000012802)

Professori: Giovanni Maria Farinella, Rosario Leonardi.

Dipartimento: D.M.I. Informatica LM-18 A.A. 2023/2024

Data documento: 20/06/2024.

Indice

1 Introduzione al progetto	4
1.1 Obiettivi e sviluppi futuri	4
1.2 Alfabeti: ASL e BSL	4
1.2.1 ASL (American Sign Language)	4
1.2.2 BSL (British Sign Language)	5
1.2.3 Differenze tra i due alfabeti	5
2 Dataset	6
2.1 Fasi di acquisizione dei dati	6
2.1.1 Strumenti utilizzati	6
2.1.2 Prima fase di acquisizione	6
2.1.3 Seconda fase di acquisizione	7
2.1.4 Terza fase di acquisizione	8
2.2 Etichettatura dei dati	9
2.3 Dati all'interno del dataset	10
3 Addestramenti e risultati	11
3.1 Tecniche utilizzate per la costruzione dei modelli	11
3.1.1 Output del Modello Base	11
3.1.2 Global Average Pooling	11
3.1.3 Dense Layer con ReLU	11
3.1.4 Dropout Layer	12
3.1.5 Dense Layer con Softmax	12
3.1.6 Ottimizzatore Adam	12
3.2 Data Augmentation	13
3.3 DenseNet121	14
3.3.1 Implementazione del Modello	14
3.3.2 Risultati	15
3.3.3 Discussioni	17
3.4 VGG16	19
3.4.1 Implementazione del Modello	19
3.4.2 Addestramento	19
3.4.3 Valutazione del Modello	19
3.4.4 Risultati	19

3.4.5	Miglioramenti applicati	20
3.4.6	Risultati finali	21
3.5	MobileNetV2	22
3.5.1	Implementazione del modello	22
3.5.2	Fine-tuning	22
3.5.3	Risultati	23
3.5.4	Discussione	27
3.6	Altre reti utilizzate	27
4	Classificazione di immagini esterne	30
4.1	Introduzione al classificatore	30
4.2	Classificazione alfabeto ASL	31
4.3	Classificazione alfabeto BSL	32
4.4	Classificazione con entrambi gli alfabeti	34
4.5	Classificazione con soggetti non presenti nel dataset	34
4.5.1	Alfabeto BSL	34
4.5.2	Alfabeto ASL	35
4.6	Classificazione con DenseNet121	36
5	Classificazione con dataset variato	38
5.1	Addestramento del dataset	39
5.2	Risultati dell'addestramento	39
5.3	Risultati della classificazione	40
5.4	Discussione	42
6	Conclusione	42
6.1	Sviluppi futuri	43
6.2	Struttura GitHub	43
7	Riferimenti	45

1 Introduzione al progetto

Il linguaggio dei segni è una forma di comunicazione visiva molto ricca e complessa, usata principalmente da persone sordi. Questo progetto vuole creare un sistema che possa interpretare il linguaggio dei segni utilizzando modelli avanzati di apprendimento automatico. Questo perché, nonostante la sua importanza, esistono ancora notevoli barriere comunicative tra le persone sordi e quelle udenti. La tecnologia può aiutare a ridurre queste barriere, migliorando l'accessibilità e l'inclusione sociale. Utilizzando tecniche di machine learning, come le reti neurali convoluzionali (CNN), questo progetto si propone di creare un modello in grado di riconoscere e interpretare accuratamente i gesti del linguaggio dei segni.

1.1 Obiettivi e sviluppi futuri

L'obiettivo del progetto è il riconoscimento automatico delle diverse varianti regionali del linguaggio dei segni. Esistono numerose varianti, come la lingua dei segni americana (ASL), la lingua dei segni britannica (BSL) e molte altre, ciascuna con le proprie regole grammaticali e sintattiche. Riconoscere automaticamente la variante del linguaggio dei segni utilizzato permetterebbe una traduzione più precisa e contestualizzata, migliorando ulteriormente l'efficacia del sistema in contesti culturalmente diversi. Il sistema sviluppato potrebbe trovare applicazione nel campo della computer vision, permettendo la traduzione del linguaggio dei segni da video preregistrati o trasmissioni in diretta. Questo consentirebbe, ad esempio, di sottotitolare automaticamente programmi televisivi, eventi live o contenuti online, migliorando l'accessibilità per le persone sordi. Inoltre, la capacità di tradurre il linguaggio dei segni in tempo reale potrebbe rivoluzionare il modo in cui le persone sordi interagiscono con gli ambienti digitali, rendendo più accessibili le piattaforme di videochiamata, i webinar e altre forme di comunicazione visiva.

1.2 Alfabeti: ASL e BSL

La decisione di focalizzarsi sull'American Sign Language (ASL) e sul British Sign Language (BSL) all'interno di questo progetto è stata guidata dalla netta differenza esistente tra i due alfabeti. Questa scelta mira a esplorare e comprendere le peculiarità uniche di queste due varianti regionali del linguaggio dei segni, fornendo una soluzione di riconoscimento che tenga conto delle loro differenze distintive.

1.2.1 ASL (American Sign Language)

L'American Sign Language (ASL) è una lingua visiva-gestuale ampiamente utilizzata negli Stati Uniti d'America e in alcune parti del Canada. Un aspetto peculiare dell'ASL è l'uso predominante di una sola mano per eseguire i segni, dove la mano dominante è solitamente quella utilizzata per esprimere le parole, mentre l'altra può essere utilizzata per fornire supporto o per modificare il significato dei segni.

1.2.2 BSL (British Sign Language)

Il British Sign Language (BSL) è la lingua dei segni predominante nel Regno Unito e viene utilizzata da un'ampia parte della comunità sorda britannica. Una caratteristica distintiva del BSL è l'uso predominante di entrambe le mani nell'esecuzione dei segni.

1.2.3 Differenze tra i due alfabeti

La differenza principale tra l'American Sign Language (ASL) e il British Sign Language (BSL) è presente nella struttura dei segni. Come precedentemente descritto il linguaggio ASL utilizza principalmente una mano per eseguire i segni, con la mano dominante che esegue la maggior parte dei gesti. L'altra mano può essere utilizzata per supportare il gesto o per fornire ulteriori informazioni. Mentre il linguaggio BSL coinvolge spesso entrambe le mani nell'esecuzione dei segni. Questo significa che molti segni del BSL richiedono l'uso simultaneo di entrambe le mani, consentendo una maggiore varietà di movimenti e espressioni. Sono presenti altre differenze nell'uso dei linguaggi che derivano dalle convenzioni culturali e dalle differenti sintassi delle parole e delle frasi. Di seguito è riportata la parola "HELLO" nei due linguaggi dei segni, così da poter apprezzare le differenze nella struttura dei gesti.

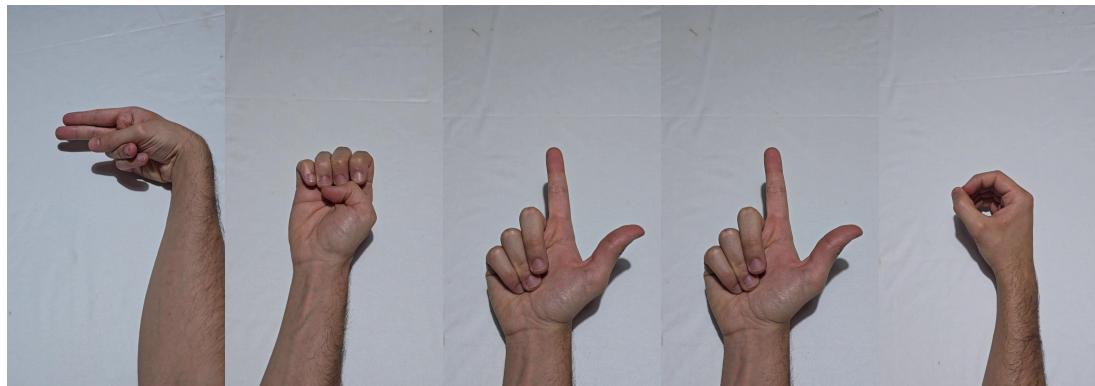


Fig. 1: "Hello" nel linguaggio ASL

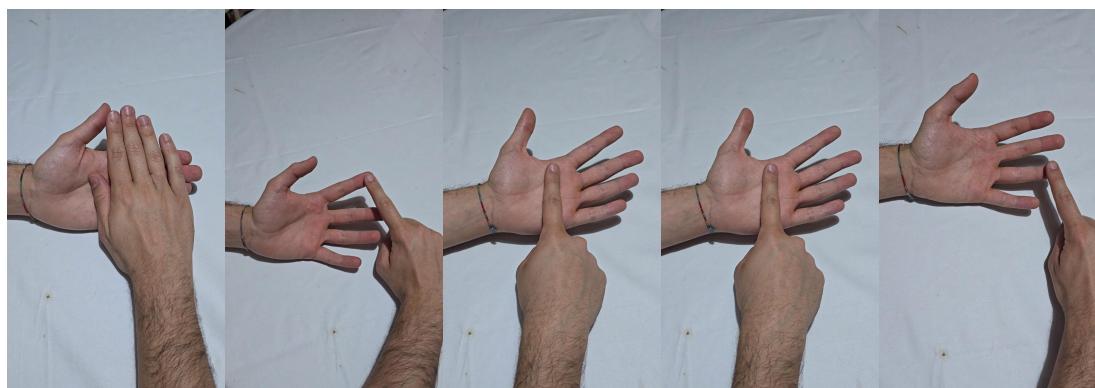


Fig. 2: "Hello" nel linguaggio BSL

Nella Figura 1 è evidente l'impiego di una sola mano, mentre nella Figura 2 sono coinvolte entrambe le mani. Da questo esempio è possibile notare come i gesti presenti nelle figure siano completamente differenti. È importante sottolineare che le uniche due lettere escluse dai due alfabeti sono la J e la Z, poiché esse richiedono movimento anziché una posizione statica come le restanti lettere. Infine, per quanto riguarda la lettera H dell'alfabeto BSL, si è concordato con il docente che, nonostante la rappresentazione originale sia dinamica, essa sarebbe stata comunque inclusa, seppur convertita in una rappresentazione statica, come è possibile vedere nella prima immagine della Figura 2.

2 Dataset

Per sviluppare un sistema di riconoscimento del linguaggio dei segni efficace, è essenziale disporre di un dataset di alta qualità che contenga immagini rappresentative dei gesti utilizzati nei linguaggi dei segni. L'acquisizione dei dati per questo progetto ha seguito un processo meticoloso per garantire che il dataset fosse completo e diversificato, includendo sia il linguaggio dei segni americano (ASL) sia quello britannico (BSL).

2.1 Fasi di acquisizione dei dati

L'acquisizione dei dati è avvenuta tre volte a causa di scelte errate nelle prime fasi del processo.

2.1.1 Strumenti utilizzati

Per catturare i gesti in modo preciso, sono state utilizzate videocamere ad alta risoluzione posizionate in modo tale da riprendere chiaramente le mani. Le riprese sono state effettuate in un ambiente ben illuminato per minimizzare le ombre e migliorare la qualità delle immagini. È stato inoltre utilizzato uno sfondo uniforme per facilitare l'estrazione dei gesti durante la fase di preprocessing dei dati. Lo strumento di ripresa utilizzato per le immagini è:

- Samsung S24 Ultra SM-S928B/DS, One UI 6.1, Android 14.

2.1.2 Prima fase di acquisizione

Durante la fase iniziale di acquisizione dei dati, per ogni lettera degli alfabeti ASL e BSL, i partecipanti sono stati fotografati mentre eseguivano ciascun segno. Per ciascun segno, sono state scattate tre foto (risoluzione 1920x1080) da diverse angolazioni per garantire una rappresentazione variegata e accurata dei gesti. Questo metodo ha offerto un vantaggio significativo in termini di velocità, permettendo di raccogliere rapidamente un gran numero di immagini. Tuttavia, ha anche presentato una sfida: il numero relativamente basso di immagini per ogni segno potrebbe portare a un underfitting durante l'addestramento del modello, poiché il dataset potrebbe non essere sufficientemente variegato per catturare tutte le variazioni naturali nei gesti. Come è possibile osservare nella Figura 3, è presente la rappresentazione della lettera A dell'alfabeto BSL interpretata da tutti e tre i partecipanti del progetto.

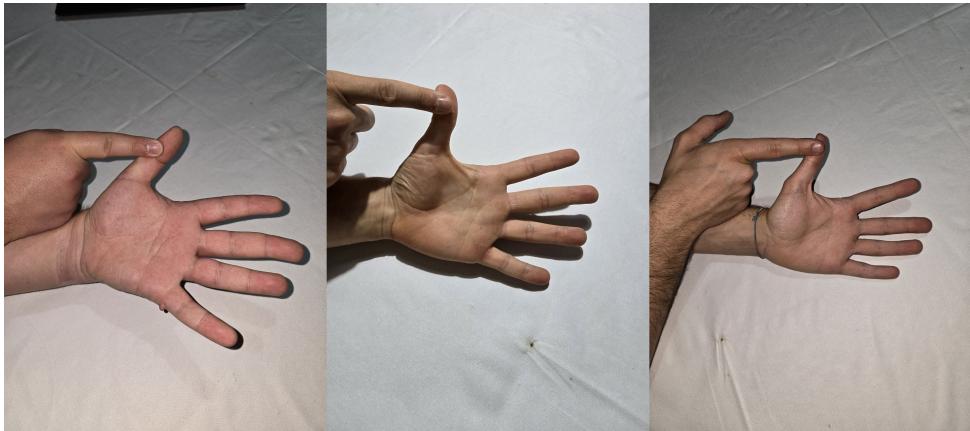


Fig. 3: Rappresentazione lettera A dell’alfabeto BSL nel seguente ordine: Giuseppe Napoli, Gianluca Giardina, Luigi Priano

La suddivisione dei dati è stata effettuata assegnando due partecipanti al training e uno al test. In numeri, ciò si traduce in 288 elementi per il training e 144 per il testing, con una distribuzione del 66.6% per il training e del 33.3% per il testing. Il principale vantaggio di questa metodologia è la velocità di acquisizione del dataset. Tuttavia, ci sono alcuni svantaggi potenziali, tra cui l’overfitting che potrebbe verificarsi a causa della quantità limitata di dati. In questo caso, il modello potrebbe ottenere alte prestazioni sui dati di training, ma risultati scadenti sui dati di test. Il secondo svantaggio è l’underfitting che potrebbe derivare dall’uso di un modello troppo semplice che non riesce a catturare le relazioni presenti nei dati. Questo si traduce in prestazioni insoddisfacenti sia sui dati di training che su quelli di test.

2.1.3 Seconda fase di acquisizione

Nella seconda fase di acquisizione dei dati, è stato deciso di registrare video (Risoluzione 1920x1080 60 FPS) di un singolo partecipante del gruppo mentre eseguiva ogni carattere dell’alfabeto dei segni. Successivamente, i video sono stati suddivisi in frame, ottenendo circa 900 frame per carattere. Durante la registrazione il partecipante traslava ed effettuava lo scaling del carattere dell’alfabeto. Il vantaggio principale è stata la possibilità di avere un elevato numero di campioni, questo perché la suddivisione dei video in frame permette di ottenere un grande numero di campioni per ogni carattere, aumentando la quantità di dati disponibili per l’addestramento del modello. Gli svantaggi sono molteplici come l’assenza di rotazione del carattere, la presenza di frame duplicati dovuti all’alto frame rate dei video, questi non aggiungono informazioni utili e potrebbero rallentare l’addestramento senza migliorare le prestazioni del modello. Lo svantaggio più grande era dovuto alla traslazione del carattere che rendeva complessa l’etichettatura poiché necessitava di etichettare frame per frame in base alla posizione delle mani, questo causa rallentamenti e confusione durante l’addestramento. Di seguito è presente la Figura 4 con le differenti posizioni presenti all’interno dei video.

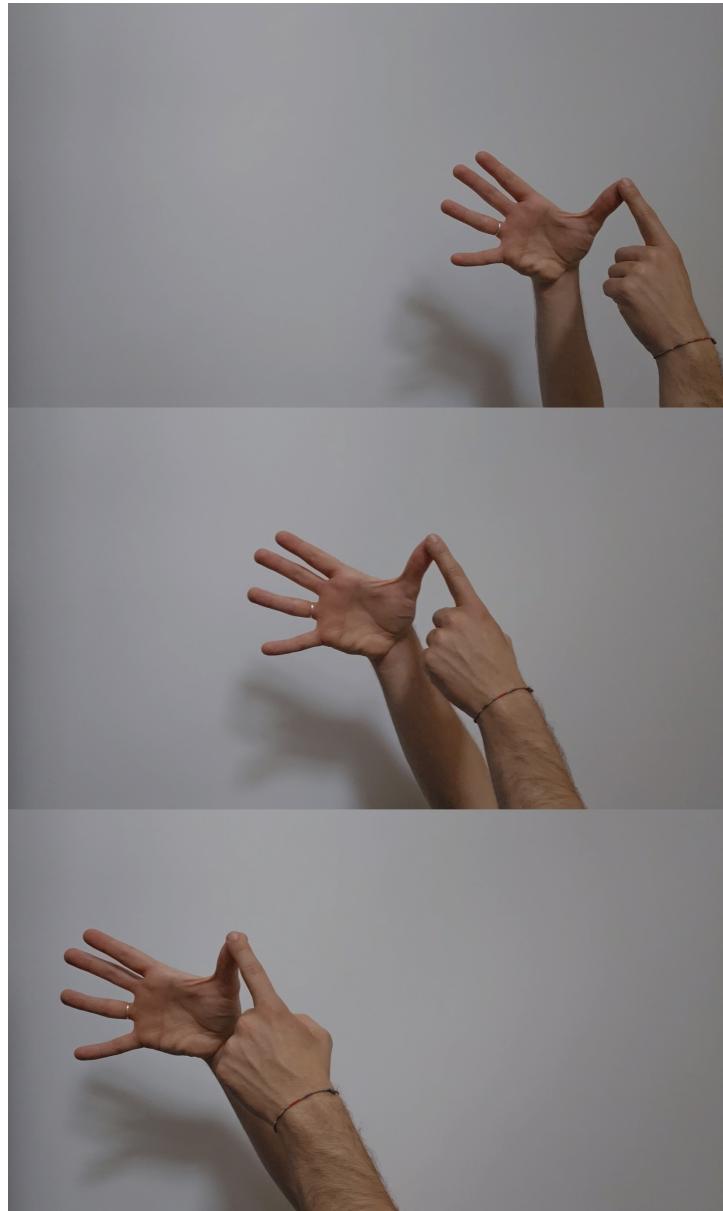


Fig. 4: Posizione del carattere traslata nei diversi frame

Come è facile notare dalla figura soprastante, il gesto eseguito con le mani è presente in diverse posizioni all'interno dello stesso video. Questo comporta, come già detto, una etichettatura speciale per ogni singolo frame, rendendo l'acquisizione del dataset un processo lento e meticoloso.

2.1.4 Terza fase di acquisizione

Dopo aver parlato col professore della seconda fase di acquisizione, abbiamo deciso di evitare la traslazione del carattere, mantenendo solo rotazione e scaling. Inoltre, abbiamo scelto di mantenere uno sfondo bianco uniforme per tutti i video. Il nostro obiettivo era di utilizzare un unico elemento come modello per i caratteri, mantenendo il soggetto sempre al centro del video per garantire una coerenza visiva e una facile identificazione ed etichettatura. Ogni carattere è stato diviso in tre video distinti, ciascuno della durata di 20 secondi, per evitare frame simili e per avere una prospettiva diversa per ogni carattere.

Questo approccio ci ha permesso di raccogliere circa 120 frame per ogni carattere, con una frequenza di 2 frame per secondo. La suddivisione dei frame è stata effettuata come segue: 80 frame per il training, 14 frame per la validation e 26 frame per il test. In particolare, la suddivisione del terzo video è stata eseguita nel rapporto di 35% per il validation e 65% per il test. I principali pregi di questo metodo di acquisizione sono un numero adeguato di immagini per evitare problemi di overfitting e underfitting. Inoltre, l'uso di rotazione e scaling senza traslazione contribuisce a mantenere la coerenza spaziale dei caratteri, migliorando la robustezza del modello e rendendo più facile l'etichettatura. Gli svantaggi invece sono molteplici come l'acquisizione dei dati relativamente lenta, la suddivisione in frame del video computazionalmente pesante e il peso finale del dataset ha un peso considerevole. Di seguito, nella Figura 5, sono presenti degli esempi casuali dei frame acquisiti durante questa fase.

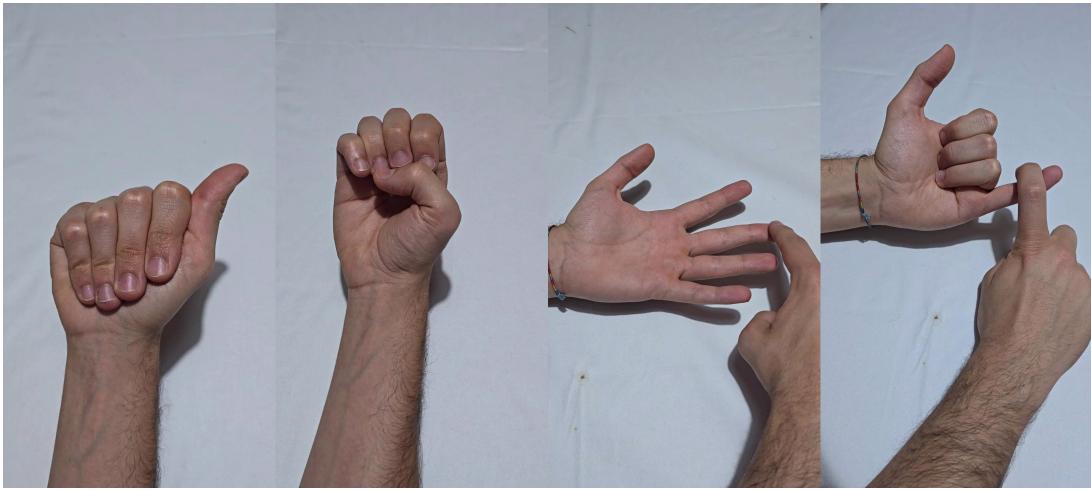


Fig. 5: Rappresentazione dei seguenti caratteri: A ASL, E ASL, I BSL, S BSL

2.2 Etichettatura dei dati

Per l'etichettatura del dataset, è stata utilizzata la piattaforma Roboflow. Questo strumento ha permesso di gestire e annotare le immagini in modo efficiente e preciso. Il processo di etichettatura è stato eseguito in base all'ordine in cui le immagini venivano caricate sulla piattaforma. In particolare, una volta caricata un'immagine su Roboflow, l'interfaccia della piattaforma ha fornito gli strumenti necessari per aggiungere etichette e annotazioni. Queste annotazioni includevano bounding box, punti di interesse, segmentazione e altre forme di markup necessarie per il progetto. Inoltre, Roboflow ha permesso di selezionare le percentuali di suddivisione del dataset in train, validation e test per ogni classe. Questa funzionalità ha garantito una distribuzione equilibrata e rappresentativa dei dati durante le fasi di addestramento e validazione dei modelli. Il flusso di lavoro è stato il seguente:

- 1. Suddivisione dei video in frame tramite script in Python:** I video sono stati suddivisi in singoli frame, come specificato nella terza fase di acquisizione, utilizzando uno script in Python. Questo processo ha permesso di ottenere una serie di immagini statiche da ogni video.

2. **Suddivisione delle immagini in cartelle e caricamento su Roboflow:** Le immagini ottenute dai frame sono state organizzate in cartelle, queste ultime sono state poi rinominate seguendo la convenzione "Carattere_ASL" o "Carattere_BSL" per indicare se il carattere rappresentato apparteneva alla lingua dei segni americana (ASL) o britannica (BSL). Successivamente, queste cartelle sono state caricate su Roboflow.
3. **Selezione delle percentuali di train/valid/test:** Utilizzando le funzionalità di Roboflow, sono state selezionate le percentuali di suddivisione del dataset tra training, validation e test per ciascuna classe. Questo ha garantito una distribuzione bilanciata e rappresentativa dei dati per le diverse fasi di sviluppo dei modelli.
4. **Verifica dell'inserimento e dell'etichettatura del dataset:** Dopo la suddivisione, è stata effettuata una verifica completa per assicurarsi che tutte le immagini fossero state correttamente caricate e annotate. Questo passaggio ha garantito l'accuratezza e la completezza del dataset.
5. **Esportazione del dataset tramite l'uso di API:** Una volta completata l'etichettatura e la verifica, il dataset è stato esportato utilizzando le API di Roboflow. Questo ha permesso di ottenere il dataset in vari formati compatibili con i modelli di machine learning e altre applicazioni di analisi dei dati. Il codice per l'acquisizione è il seguente:

```
!pip install roboflow
from roboflow import Roboflow
rf = Roboflow(api_key="IwbjvxFzcbBgxgAJv6M2")
project = rf.workspace("aslvsbsl").project("aslvsbsl")
version = project.version(2)
dataset = version.download("folder")
```

2.3 Dati all'interno del dataset

Il dataset è stato organizzato in tre cartelle principali: train, valid e test, ciascuna contenente 48 classi denominate "Carattere_ASL" o "Carattere_BSL". All'interno di ogni cartella di train sono presenti 80 immagini per classe, mentre nelle cartelle di valid e test sono presenti rispettivamente 14 e 26 immagini per classe. Le immagini all'interno delle cartelle di train sono numerate da 1 a 80, mentre quelle nelle cartelle di valid e test sono state selezionate casualmente da un intervallo che va da 81 a 120. Roboflow ha aggiunto un'etichettatura nel nome delle immagini, ad esempio

"rf.169eff9ec4c6617175ee970cdffdb214.jpg", presente subito dopo il nome dell'immagine. In totale, il dataset comprende 5760 immagini.

3 Addestramenti e risultati

3.1 Tecniche utilizzate per la costruzione dei modelli

Nella costruzione dei modelli di reti neurali convoluzionali, sono state utilizzate diverse tecniche per adattare modelli pre-addestrati a compiti specifici di classificazione delle immagini. Di seguito viene fornita una spiegazione delle funzioni e delle tecniche utilizzate:

3.1.1 Output del Modello Base

```
x = base_model.output
```

Il primo passo è ottenere l'output del modello base pre-addestrato. Questo output rappresenta le caratteristiche estratte dalle immagini di input attraverso i vari layer convoluzionali del modello pre-addestrato. Utilizzare un modello pre-addestrato consente di sfruttare le conoscenze acquisite da grandi dataset generici come ImageNet.

3.1.2 Global Average Pooling

```
x = GlobalAveragePooling2D()(x)
```

Il `GlobalAveragePooling2D` è una tecnica utilizzata per ridurre la dimensionalità dei dati senza perdere informazioni spaziali importanti. Questo layer calcola la media di ogni mappa di caratteristiche 2D, producendo un vettore di caratteristiche 1D. Questa tecnica riduce significativamente il numero di parametri e il rischio di overfitting, migliorando la generalizzazione del modello.

3.1.3 Dense Layer con ReLU

```
x = Dense(512, activation='relu')(x)
```

Un `Dense` layer, o fully connected layer, è un layer in cui ogni neurone è connesso a tutti i neuroni del layer precedente. Il layer con 512 unità introduce non linearità nel modello tramite la funzione di attivazione ReLU (Rectified Linear Unit). La funzione ReLU è definita come:

$$\text{ReLU}(x) = \max(0, x)$$

Questa funzione aiuta il modello ad apprendere rappresentazioni complesse dei dati, attivando solo i neuroni con valori positivi e mantenendo i valori negativi a zero.

3.1.4 Dropout Layer

```
x = Dropout(0.5)(x)
```

Il **Dropout** layer è una tecnica di regolarizzazione utilizzata per prevenire l'overfitting. Durante l'addestramento, spegne casualmente il 50% dei neuroni nel layer, costringendo la rete a non dipendere troppo da alcun neurone specifico e migliorando la generalizzazione del modello.

3.1.5 Dense Layer con Softmax

```
predictions = Dense(num_classes, activation='softmax')(x)
```

L'ultimo layer è un **Dense** layer con una funzione di attivazione **softmax**. Questo layer produce un'uscita di probabilità per ciascuna delle classi nel dataset, permettendo al modello di effettuare una classificazione.

3.1.6 Ottimizzatore Adam

```
model.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=learning_rate),  
loss='categorical_crossentropy', metrics=['accuracy'])
```

L'ottimizzatore **Adam** (Adaptive Moment Estimation) è utilizzato per aggiornare i pesi della rete neurale durante l'addestramento. Adam combina i vantaggi di due altri metodi di ottimizzazione: AdaGrad e RMSProp. Adam utilizza stime sia della media dei gradienti (momenti) sia della varianza dei gradienti per adattare i tassi di apprendimento per ogni peso individuale del modello. Questo lo rende particolarmente efficace per problemi con grandi quantità di dati e/o parametri.

3.2 Data Augmentation

La data augmentation è una tecnica utilizzata nell'addestramento delle reti neurali per aumentare artificialmente la quantità di dati di training disponibili. Questo è particolarmente utile per migliorare la capacità del modello di generalizzare e di prevenire l'overfitting. Nel codice fornito, vengono eseguite diverse operazioni di data augmentation sul set di dati di training mediante la classe `ImageDataGenerator` di Keras. Più nello specifico sono presenti le seguenti operazioni di data augmentation:

- Rescale: i valori dei pixel vengono trasformati dall'intervallo 0-255 all'intervallo 0-1;
- Rotation_range: le immagini vengono ruotate casualmente entro un intervallo di 20 gradi, aiutando il modello a essere robusto alle variazioni angolari;
- Width_shift_range: le immagini vengono spostate casualmente orizzontalmente e verticalmente fino al 20% delle dimensioni originali, rendendo il modello meno sensibile agli spostamenti;
- Shear_range: viene applicata una trasformazione di shear (taglio) fino al 20%;
- Zoom_range: le immagini vengono ingrandite casualmente entro un intervallo del 20%;
- Horizontal_flip: le immagini vengono capovolte orizzontalmente;
- Fill_mode: i pixel mancanti dopo le trasformazioni vengono riempiti utilizzando il valore del pixel più vicino "nearest".

```
train_datagen = ImageDataGenerator(  
    rescale=1.0/255.0,  
    rotation_range=20,  
    width_shift_range=0.2,  
    height_shift_range=0.2,  
    shear_range=0.2,  
    zoom_range=0.2,  
    horizontal_flip=True,  
    fill_mode='nearest'  
)  
valid_datagen = ImageDataGenerator(rescale=1.0/255.0)  
test_datagen = ImageDataGenerator(rescale=1.0/255.0)
```

3.3 DenseNet121

DenseNet121 è un tipo di rete neurale convoluzionale caratterizzata da connessioni dense tra i layer. In una DenseNet, ogni layer riceve in input tutte le mappe di attivazione di tutti i layer precedenti. Questo approccio aiuta a mitigare il problema del *vanishing gradient* e a migliorare il flusso di informazioni e i gradienti attraverso la rete.

La struttura di DenseNet121 è particolarmente efficace per la classificazione delle immagini, grazie alla sua capacità di riutilizzare le caratteristiche estratte nei vari layer della rete, migliorando così l'efficienza dei parametri e riducendo il rischio di overfitting.

DenseNet121 è pre-addestrata su ImageNet, un ampio dataset di immagini comunemente utilizzato per il pre-addestramento dei modelli di deep learning.

3.3.1 Implementazione del Modello

Il modello DenseNet121 viene caricato con i pesi pre-addestrati su ImageNet, escludendo l'ultimo strato di classificazione della rete originale. Questo permette di aggiungere strati finali per adattare il modello a un diverso compito di classificazione. Inoltre, vengono specificate le dimensioni delle immagini in ingresso. Successivamente, il modello di base viene congelato in modo tale che i suoi pesi non vengano aggiornati durante l'addestramento.

```
base_model = DenseNet121(weights='imagenet', include_top=False,
                           input_shape=(img_height, img_width, 3))
base_model.trainable = False
```

La creazione del modello sequenziale include i seguenti strati: GlobalAveragePooling2D, Dense con 512 neuroni e una funzione di attivazione ReLU, BatchNormalization, Dropout impostato a 0.5, e un ulteriore Dense che aggiunge uno strato denso con un numero di neuroni pari al numero delle classi usando la funzione di attivazione softmax.

```
model = Sequential([
    base_model,
    GlobalAveragePooling2D(),
    Dense(512, activation='relu'),
    BatchNormalization(),
    Dropout(0.5),
    Dense(num_classes, activation='softmax')
])
```

Il modello viene quindi compilato usando l'ottimizzatore Adam, specificando la perdita come categorical_crossentropy e l'accuratezza come metrica di valutazione.

```
model.compile(optimizer='adam',
              loss='categorical_crossentropy',
              metrics=['accuracy'])
```

3.3.2 Risultati

I risultati dell’addestramento mostrano variazioni significative in base a diversi fattori come il learning rate, il numero di epoche, e l’assenza di data augmentation. Di seguito sono elencati i risultati ottenuti:

- Nel primo tentativo, con un learning rate di 0.001 e 15 epoche, l’addestramento ha raggiunto un’accuratezza del 83.41%. I risultati mostrano una convergenza veloce ma una potenziale sottostima della capacità del modello di generalizzare, come si evince dall’andamento dei grafici di accuratezza e loss.

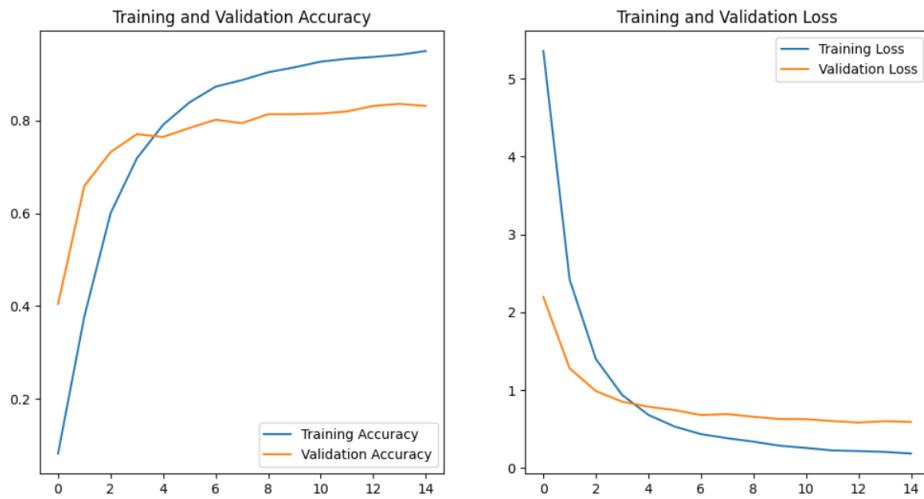


Fig. 6: Primo tentativo con learning rate di 0.001 e 15 epoche

- Il secondo tentativo, con un learning rate ridotto a 0.0001 e un aumento delle epoche a 30, ha mostrato un perggioramento nell’accuratezza al 72.52% e una riduzione della loss. Questo suggerisce che un learning rate più basso con un numero maggiore di epoche potrebbe favorire una migliore convergenza senza overfitting.

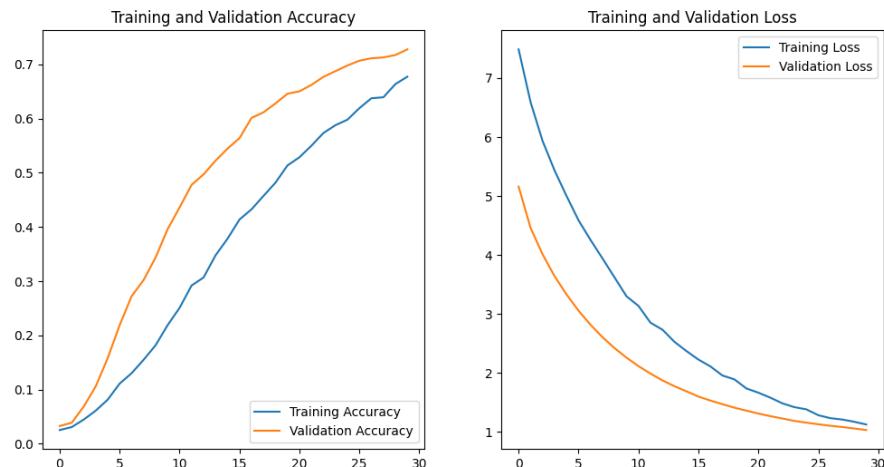


Fig. 7: Secondo tentativo con learning rate di 0.0001 e 30 epoche

- Alla luce delle osservazioni nel secondo tentativo, è stata condotta un'ulteriore sessione di addestramento estendendo il numero di epoche a 50. Questa configurazione ha portato ad un'ulteriore incremento dell'accuratezza al 77.48%, confermando che un maggiore numero di epoche può effettivamente migliorare la performance del modello senza incorrere in overfitting evidente.

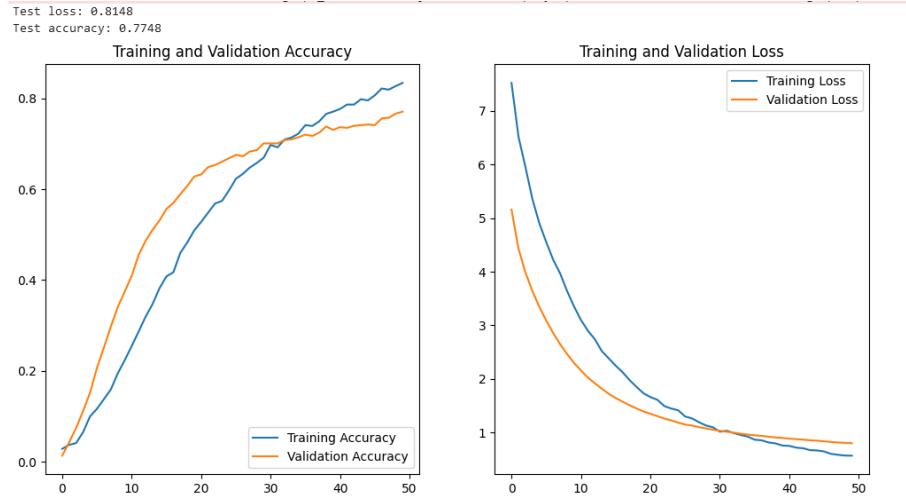


Fig. 8: Terzo tentativo con learning rate di 0.0001 e 50 epoche

- Nel quarto tentativo, il modello è stato valutato utilizzando un set di dati bilanciato per ASL e BSL. La matrice di confusione (Figura 10), le metriche di valutazione (Figura 11), e i grafici di accuratezza e loss (Figura 9) mostrano una buona performance complessiva del modello.

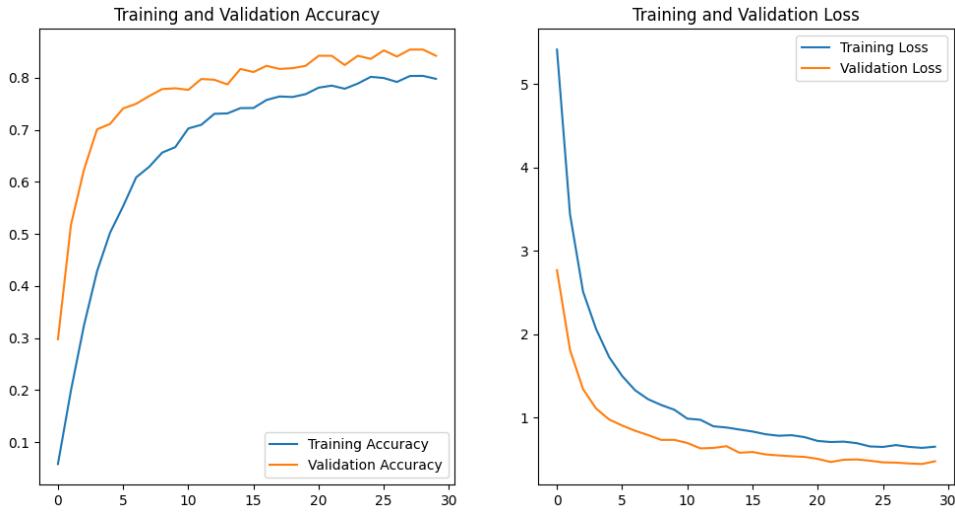


Fig. 9: Quarto tentativo con learning rate di 0.001, data augmentation e 30 epoche

La matrice di confusione evidenzia la capacità del modello di distinguere tra le diverse lettere in ASL e BSL, con la maggior parte delle predizioni corrette concentrate sulla diagonale. Tuttavia, alcune lettere come 'I_BSL' e 'J_BSL' mostrano confusioni reciproche, dovute a caratteristiche visive simili difficili da distinguere per il modello.

Le metriche di valutazione indicano che la precisione, il richiamo e l'F1-score sono generalmente elevati, con le classi migliori che includono 'A_BSL', 'F_BSL', 'M_BSL', 'S_ASL', e 'Y_BSL' con valori pari a 1.00. Tuttavia, alcune classi come 'N_BSL', 'O_BSL', e 'V_ASL' hanno .

I grafici di accuratezza e loss mostrano una crescita costante dell'accuratezza, stabilizzandosi verso la fine dell'addestramento, suggerendo che il modello sta convergendo. La loss di validazione segue un trend decrescente simile, indicando che il modello sta migliorando la sua capacità di generalizzare sui dati non visti. La leggera divergenza tra la loss di addestramento e quella di validazione nelle ultime epoche potrebbe indicare un inizio di overfitting.

3.3.3 Discussioni

La serie di esperimenti condotti ha offerto intuizioni riguardo l'impatto del learning rate, del numero di epoche, e dell'uso della data augmentation sulla performance del modello. Di seguito, discutiamo le principali osservazioni.

- **Influenza del Learning Rate e delle Epoche:** I risultati hanno chiaramente dimostrato che modifiche al learning rate e al numero di epoche hanno un effetto significativo sull'accuratezza e sulla perdita del modello. Un learning rate troppo alto può causare una convergenza prematura, limitando la capacità del modello di esplorare lo spazio delle soluzioni. Al contrario, un learning rate più basso, sebbene rallenti il processo di addestramento, può portare a una migliore generalizzazione, come dimostrato dall'aumento dell'accuratezza con l'aumentare delle epoche;
- **Impatto della Data Augmentation:** L'introduzione della data augmentation ha migliorato significativamente l'accuratezza del modello, confermando il suo valore come strumento per aumentare la diversità del dataset di addestramento e per prevenire l'overfitting. Questo approccio si è rivelato particolarmente utile quando combinato con un learning rate di 0.001 e un numero maggiore di epoche.

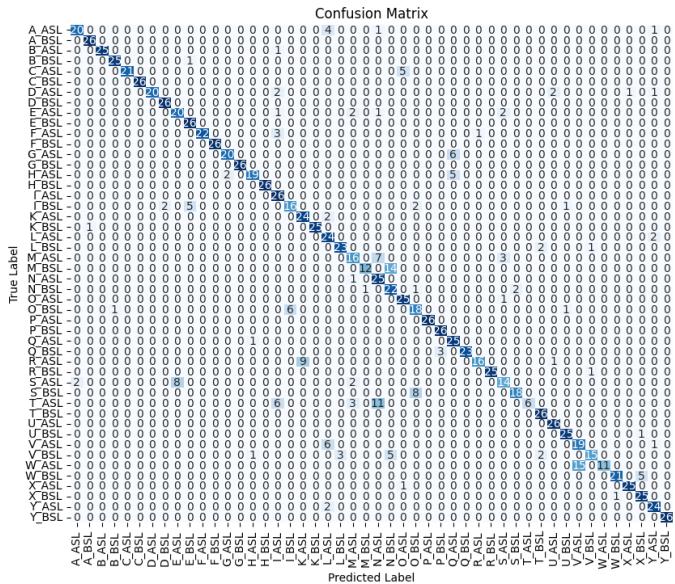


Fig. 10: Matrice di confusione del modello addestrato.

	precision	recall	f1-score	support
A_ASL	0.91	0.77	0.83	26
A_BSL	0.96	1.00	0.98	26
B_ASL	1.00	0.96	0.98	26
B_BSL	0.96	0.96	0.96	26
C_ASL	1.00	0.81	0.89	26
C_BSL	1.00	1.00	1.00	26
D_ASL	1.00	0.77	0.87	26
D_BSL	0.93	1.00	0.96	26
E_ASL	0.71	0.77	0.74	26
E_BSL	0.81	1.00	0.90	26
F_ASL	1.00	0.85	0.92	26
F_BSL	1.00	1.00	1.00	26
G_ASL	0.91	0.77	0.83	26
G_BSL	1.00	1.00	1.00	26
H_ASL	0.90	0.73	0.81	26
H_BSL	1.00	1.00	1.00	26
I_ASL	0.67	1.00	0.80	26
I_BSL	0.73	0.62	0.67	26
K_ASL	0.73	0.92	0.81	26
K_BSL	1.00	0.96	0.98	26
L_ASL	0.63	0.92	0.75	26
L_BSL	0.88	0.88	0.88	26
M_ASL	0.67	0.62	0.64	26
M_BSL	0.92	0.46	0.62	26
N_ASL	0.56	0.96	0.70	26
N_BSL	0.54	0.85	0.66	26
O_ASL	0.81	0.96	0.88	26
O_BSL	0.62	0.69	0.65	26
P_ASL	1.00	1.00	1.00	26
P_BSL	0.90	1.00	0.95	26
Q_ASL	0.69	0.96	0.81	26
Q_BSL	1.00	0.88	0.94	26
R_ASL	0.94	0.62	0.74	26
R_BSL	1.00	0.96	0.98	26
S_ASL	0.70	0.54	0.61	26
S_BSL	0.90	0.69	0.78	26
T_ASL	1.00	0.23	0.38	26
T_BSL	0.87	1.00	0.93	26
U_ASL	0.90	1.00	0.95	26
U_BSL	0.93	0.96	0.94	26
V_ASL	0.56	0.73	0.63	26
V_BSL	0.88	0.58	0.70	26
W_ASL	1.00	0.42	0.59	26
W_BSL	0.95	0.81	0.88	26
X_ASL	0.96	0.96	0.96	26
X_BSL	0.81	0.96	0.88	26
Y_ASL	0.83	0.92	0.87	26
Y_BSL	1.00	1.00	1.00	26
accuracy			0.84	1248
macro avg	0.87	0.84	0.84	1248
weighted avg	0.87	0.84	0.84	1248

Fig. 11: Metriche di valutazione del modello: precision, recall, f1-score, e supporto per ciascuna classe.

3.4 VGG16

VGG16 è una delle architetture di reti neurali convoluzionali più popolari, sviluppata dal Visual Geometry Group dell’Università di Oxford. Questa rete ha avuto un impatto significativo nel campo del riconoscimento delle immagini grazie alla sua profondità e semplicità di progettazione. VGG16 consiste di 16 strati con parametri addestrabili, utilizzando filtri di convoluzione 3x3 e livelli di pooling massimo. Questo modello è stato addestrato sul dataset **ImageNet**, che contiene milioni di immagini e migliaia di categorie, ed è stato utilizzato come base per molte applicazioni di visione artificiale.

3.4.1 Implementazione del Modello

L’implementazione di VGG16 per un compito di classificazione delle immagini inizia con il caricamento e la preparazione del dataset. Utilizzando `ImageDataGenerator` di TensorFlow, le immagini vengono scalate e caricate in batch dalle directory specificate. Il modello VGG16 viene poi importato con pesi pre-addestrati su **ImageNet** e senza i livelli superiori (fully connected layers). In seguito, viene creato un modello sequenziale che aggiunge il VGG16 come base non addestrabile, seguito da un livello di appiattimento, un denso con 256 unità, un dropout per la regolarizzazione e infine un livello denso con 48 unità per la classificazione delle categorie nel dataset specifico.

3.4.2 Addestramento

Il modello viene compilato utilizzando la funzione di perdita `categorical_crossentropy` e l’ottimizzatore `Adam` con un learning rate di 0.0001. Vengono aggiunti due callback, `EarlyStopping` e `ReduceLROnPlateau`, per prevenire l’overfitting e migliorare la performance di addestramento. Il modello viene addestrato per un massimo di 60 epoche con i generatori di immagini di training e validation. Durante l’addestramento, le metriche di accuratezza e perdita vengono monitorate per valutare le prestazioni del modello.

3.4.3 Valutazione del Modello

Dopo l’addestramento, vengono tracciati i grafici di accuratezza e perdita sia per i set di training che di validation per visualizzare l’andamento dell’addestramento e identificare eventuali problemi di overfitting o underfitting. Successivamente, il modello viene valutato sul set di test, che non è stato utilizzato durante l’addestramento, per ottenere un’accuratezza di test definitiva. Questo passaggio è cruciale per determinare la capacità di generalizzazione del modello.

3.4.4 Risultati

- **Andamento dell’accuracy:** Il grafico dell’accuracy mostra l’accuracy del training (linea blu) e del validation (linea arancione) rispetto alle epoche. Si può osservare che l’accuracy del training sale rapidamente nelle prime epoche, raggiungendo quasi il 100% dopo circa 10 epoche e mantenendosi stabile fino alla fine dell’addestramento. Questo indica che il modello ha appreso bene i dati di training. L’accuracy della

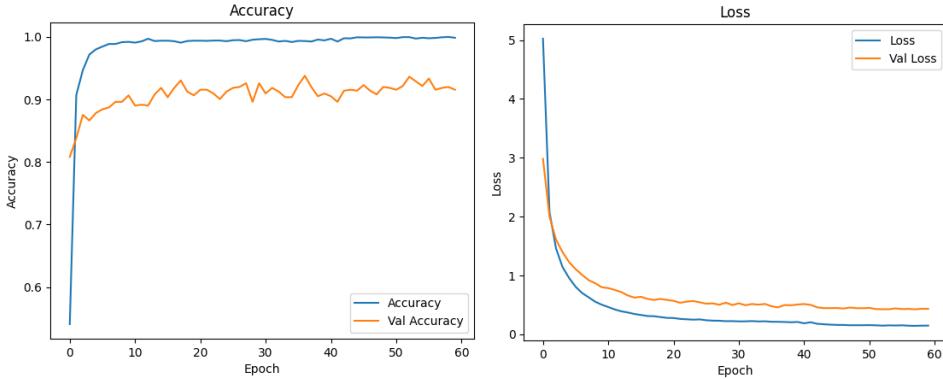


Fig. 12: Primo tentativo VGG16

validation segue una traiettoria simile, raggiungendo valori intorno al 90% nelle prime epoche e rimanendo relativamente stabile con alcune fluttuazioni fino alla fine dell’addestramento. Questa discrepanza tra l’accuratezza di training e quella di validazione suggerisce un leggero overfitting, poiché il modello performa meglio sui dati di training rispetto ai dati di validazione.

- **Andamento della loss:** Il grafico della loss mostra la perdita del training (linea blu) e del validation (linea arancione) rispetto alle epoche. La loss di training diminuisce rapidamente all’inizio, stabilizzandosi verso valori prossimi allo zero con l’avanzare delle epoche, indicando che il modello sta migliorando la sua capacità di minimizzare l’errore sui dati di training. La loss di validation segue un andamento simile, riducendosi rapidamente nelle prime epoche e stabilizzandosi intorno a valori più alti rispetto alla loss di training. Questo conferma ulteriormente la presenza di un leggero overfitting, dato che la perdita di validazione non scende tanto quanto quella di training.
- **Accuracy sul test set:** L’accuracy ottenuta sul set di test è del 90.87%. Questo valore indica la capacità del modello di generalizzare su dati non visti durante l’addestramento. Un’accuratezza del 90.87% è un risultato eccellente, dimostrando che il modello è in grado di classificare correttamente la maggior parte delle immagini del set di test, nonostante il leggero overfitting osservato nei grafici di training e validazione.
- **Conclusioni:** Il modello VGG16 addestrato ha dimostrato una forte capacità di apprendimento e generalizzazione, con un’accuratezza di test molto alta. Tuttavia, i grafici suggeriscono la necessità di ulteriori tecniche di regolarizzazione per ridurre il leggero overfitting osservato. Potrebbero essere esplorate tecniche aggiuntive come l’aumento dei dati (data augmentation) o l’implementazione di dropout aggiuntivi per migliorare ulteriormente le prestazioni del modello su dati non visti.

3.4.5 Miglioramenti applicati

Per migliorare ulteriormente le prestazioni del modello, sono state apportate diverse modifiche significative. Innanzitutto, è stata applicata la data augmentation con rotazioni, traslazioni, zoom, shear e flip orizzontale. Questo processo ha aumentato la variabilità dei dati di training, migliorando la robustezza del modello. In secondo luogo, mentre nel

modello originale tutti gli strati del VGG16 erano congelati, nella versione migliorata sono stati resi allenabili gli ultimi quattro strati, permettendo al modello di adattarsi meglio alle specificità del nuovo dataset.

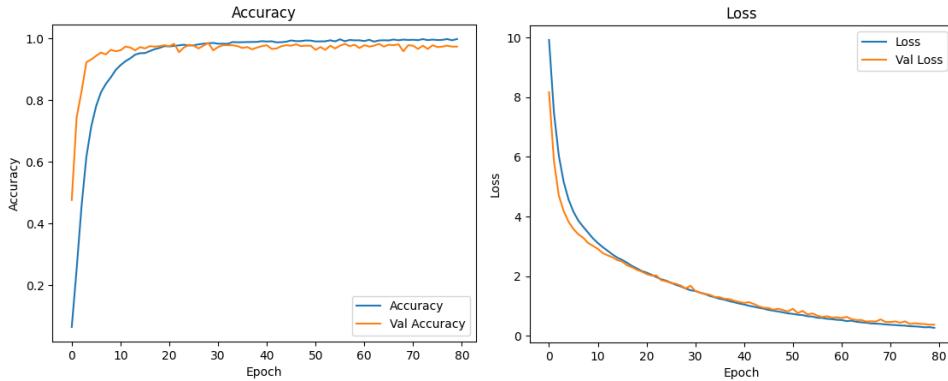


Fig. 13: Secondo tentativo VGG16 con le suddette modifiche

Inoltre, è stato aggiunto un ulteriore livello Dense di 256 unità con dropout. La configurazione precedente prevedeva un solo livello Dense di 256 unità con dropout, ma aggiungendo un secondo livello con dropout si è aumentata la capacità del modello di apprendere rappresentazioni complesse e si è ridotto l’overfitting. Anche il learning rate è stato modificato, passando da 0.0001 a 0.00005, permettendo un apprendimento più fine e riducendo il rischio di saltare minimi locali importanti. Infine, il numero di epoche è stato aumentato da 60 a 80, permettendo al modello di apprendere per un periodo più lungo e migliorando le performance complessive senza osservare overfitting.

3.4.6 Risultati finali

I risultati delle modifiche apportate sono notevoli. Il grafico dell’accuracy mostra che l’accuratezza del training sale rapidamente nelle prime epoche, raggiungendo valori superiori al 90% dopo circa dieci epoche e mantenendosi stabile fino alla fine dell’addestramento. L’accuratezza della validation, invece, raggiunge e supera i valori del 90% molto prima, assestandosi entrambe su valori superiori al 95% dopo 15 epoche. Questo indica che il modello ha appreso bene i dati di training e validation, riducendo significativamente l’overfitting.

Il grafico della loss mostra che la perdita di training diminuisce rapidamente all’inizio, stabilizzandosi verso valori prossimi allo zero con l’avanzare delle epoche, indicando che il modello sta migliorando la sua capacità di minimizzare l’errore sui dati di training. La perdita di validation segue un andamento simile, riducendosi rapidamente nelle prime epoche e stabilizzandosi intorno a valori più alti rispetto alla perdita di training, confermando ulteriormente la drastica riduzione dell’overfitting. L’accuratezza ottenuta sul set di test è del 97.36%, un risultato eccellente che dimostra la capacità del modello di generalizzare su dati non visti durante l’addestramento.

In conclusione, il modello VGG16 migliorato ha dimostrato una forte capacità di apprendimento e generalizzazione, con un’accuratezza di test molto alta. Le tecniche di data augmentation, fine-tuning degli strati finali del VGG16 e l’aggiunta di ulteriori livelli Dense con Dropout hanno contribuito significativamente a migliorare le performance del modello.

3.5 MobileNetV2

MobileNetV2 è un tipo di rete neurale sviluppata da Google per funzionare su dispositivi mobili con poca potenza di calcolo e memoria. È progettata per riconoscere immagini in modo efficiente senza richiedere molte risorse. La rete usa alcune tecniche speciali per essere veloce e leggera. Una di queste è il Bottleneck Residual Block, un blocco che aiuta a mantenere le informazioni importanti senza usare troppi parametri. Un'altra tecnica è la Depthwise Separable Convolution, che divide il processo di convoluzione in due parti più semplici, riducendo il carico di lavoro. MobileNetV2 utilizza anche i blocchi Inverted Residual, che svolgono le operazioni in un ordine diverso rispetto ai blocchi tradizionali, riducendo ulteriormente il numero di parametri necessari. Infine, la rete ha una struttura a cascata che ottimizza le prestazioni complessive.

3.5.1 Implementazione del modello

Il modello MobileNetV2 viene caricato con i pesi pre-addestrati su ImageNet escludendo l'ultimo strato di classificazione della rete originale, questo permette di aggiungere strati finali per adattare il modello a un diverso compito di classificazione. Inoltre, specifica le dimensioni delle immagini in ingresso. Successivamente il modello di base viene congelato in modo tale che i suoi pesi non verranno aggiornati durante l'addestramento. In seguito viene effettuata la creazione del modello sequenziale dove sono presenti GlobalAveragePooling2D, Dense con 512 neuroni e una funzione di attivazione ReLU, BatchNormalization, Dropout impostato a 0.5, un ulteriore Dense che aggiunge un ulteriore strato denso con un numero di neuroni pari al numero delle classi usando la funzione di attivazione softmax. In fine viene compilato il modello usando l'ottimizzatore Adam, viene specificata l'entropia e viene selezionata come metrica di valutazione l'accuratezza.

```
base_model = MobileNetV2(weights='imagenet', include_top=False,
                           input_shape=(img_height, img_width, 3))
base_model.trainable = False
model = Sequential([
    base_model,
    GlobalAveragePooling2D(),
    Dense(512, activation='relu'),
    BatchNormalization(),
    Dropout(0.5),
    Dense(num_classes, activation='softmax')
])
model.compile(optimizer='adam',
              loss='categorical_crossentropy',
              metrics=['accuracy'])
```

3.5.2 Fine-tuning

Il fine-tuning è una tecnica di apprendimento trasferito utilizzata per adattare un modello pre-addestrato su un grande set di dati generico (come ImageNet) a un nuovo compito

specifico. In pratica, si prende un modello che ha già appreso molte caratteristiche utili da un ampio set di dati e si continua il suo addestramento su un nuovo set di dati più piccolo. In questo caso le tecniche di Fine-tuning sono principalmente due:

- **ReduceLROnPlateau**: durante il fine-tuning, può essere necessario ridurre gradualmente il tasso di apprendimento man mano che il modello si avvicina alla convergenza. Questa tecnica riduce automaticamente il tasso di apprendimento quando la perdita di validazione smette di migliorare, permettendo al modello di fare aggiustamenti più piccoli e precisi nei parametri, migliorando così le prestazioni finali. In pratica se la `val_loss` non migliora per tre epoch consecutive, riduce il learning rate del 20% senza scendere sotto il minimo impostato;
- **Early_Stopping**: in questo caso è una tecnica che evita l'overfitting, quello che fa è interrompere l'addestramento se la perdita di validazione non migliora per un certo numero di epoch, preservando così le migliori prestazioni. In pratica se `val_loss` non migliora per cinque epoch consecutive l'addestramento viene fermato.

```
reduce_lr = ReduceLROnPlateau(monitor='val_loss', factor=0.2, patience=3,
min_lr=0.0001)
early_stopping = EarlyStopping(monitor='val_loss', patience=5,
restore_best_weights=True)
```

3.5.3 Risultati

I risultati dell'addestramento variano notevolmente in base alle caratteristiche del learning rate, al numero di epoch, al tipo di Fine-tuning e nell'uso della data augmentation. Di seguito vengono elencati i risultati:

- L'implementazione avente le seguenti caratteristiche: nessuna data augmentation, batch size impostato a 64, learning rate pari a 0.00001, 100 epoch. Ha portato un accuracy pari all'88% come mostrato dai grafici in figura.

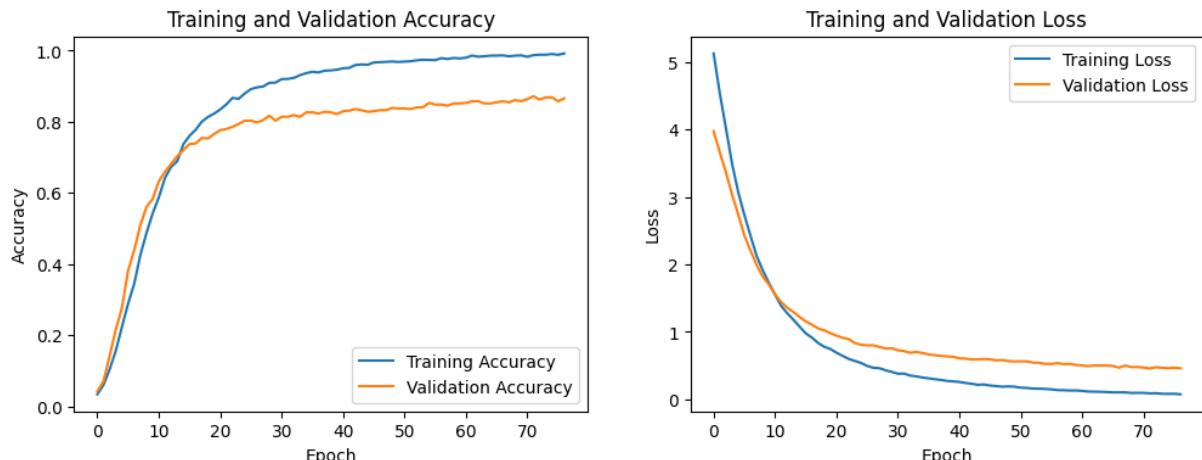


Fig. 14: Implementazione MobileNetV2 senza data augmentation

- La seconda implementazione presenta: data augmentation nel training set, batch size pari a 64, sole 20 epoche e learning rate pari a 0.001. Questo ha portato ad un accuracy del 93% con un grafico, presente nella Figura 15, che presenta delle oscillazioni da parte della validation, questo suggerisce la possibilità di overfitting.

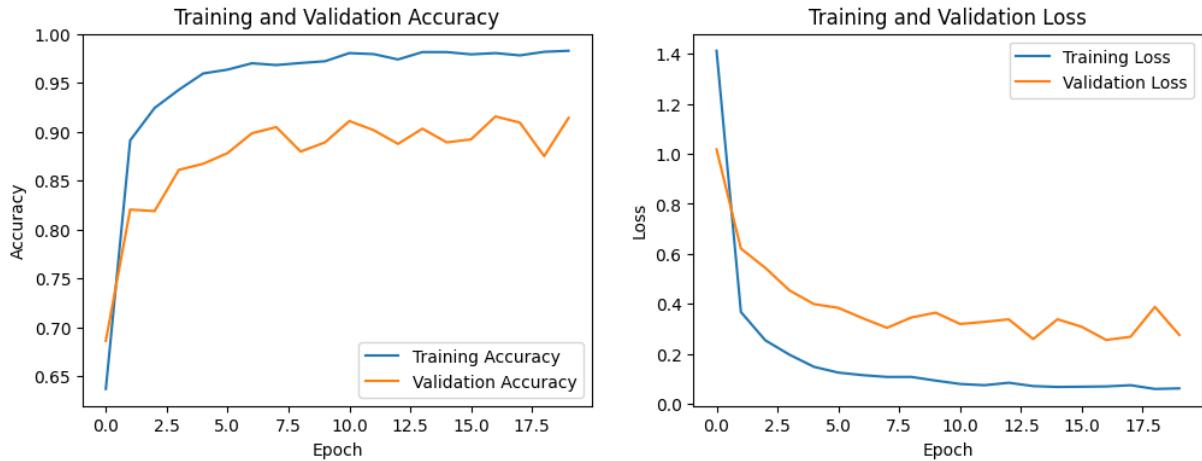


Fig. 15: Implementazione MobileNetV2 con data augmentation, learning rate 0.001

- L'ultima implementazione è come la seconda con la differenza che il learning rate è stato abbassato a 0.0001. Questo ha portato ad un'accuracy peggiore pari al 91%, ma un grafico più lineare raffigurato nella Figura 16.

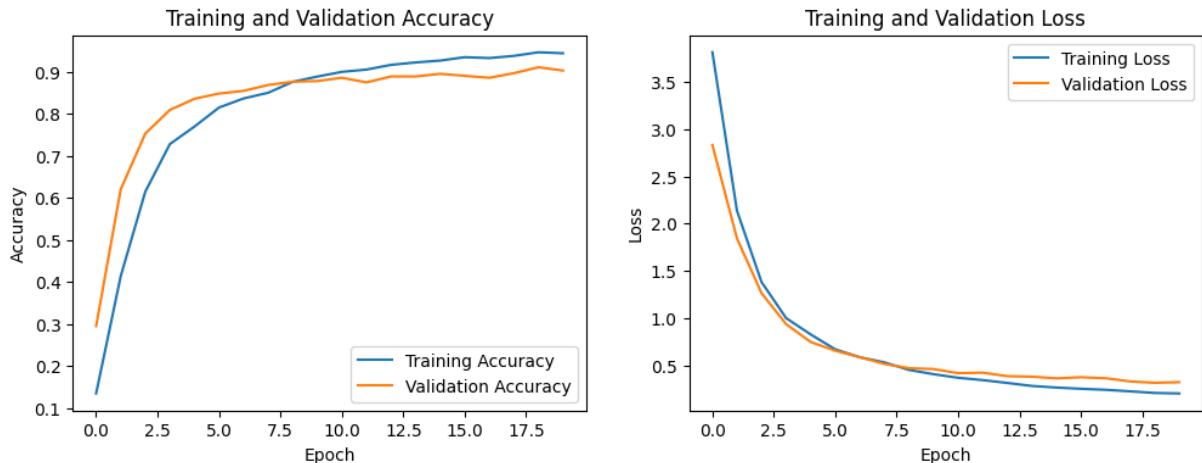


Fig. 16: Implementazione MobileNetV2 con data augmentation, learning rate 0.0001

In conclusione all'ultima implementazione è possibile osservare la seguente matrice di confusione:

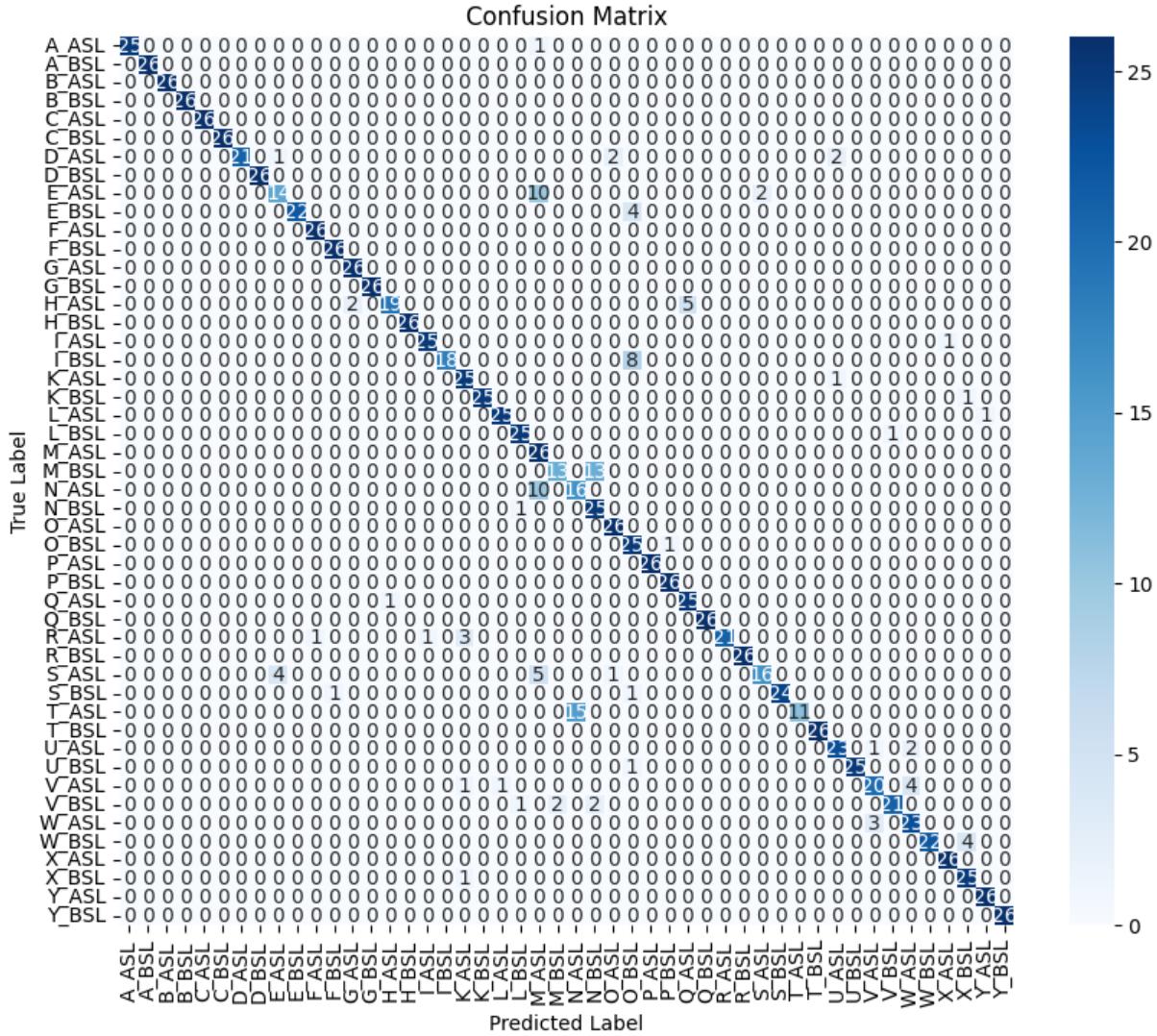


Fig. 17: Matrice di confusione MobileNetV2

Questa mostra che la maggiore confusione del modello è presente nella lettera M ed E del linguaggio ASL, N con T nel medesimo linguaggio, inoltre è possibile osservare che tutte le classi del linguaggio BSL non presentano confusione. Nella Figura 18 sono presenti i valori di precision, recall e F1-score classe per classe, è possibile osservare che:

- La maggior parte delle classi ha valori di precision, recall e F1-score alti;
- La peggiore recall è presente nella classe T_ASL e indica che il modello ha problemi a identificare correttamente questa classe;
- La classe M_ASL ha una precisione di 0.50 ma un richiamo di 1.00, indicando che tutti i casi identificati come "M_ASL" sono corretti, ma molti casi di "M_ASL" non vengono identificati.
- In generale, le classi BSL tendono ad avere prestazioni leggermente migliori rispetto alle corrispondenti classi ASL.

	precision	recall	f1-score
A_ASL	1.00	0.96	0.98
A_BSL	1.00	1.00	1.00
B_ASL	1.00	1.00	1.00
B_BSL	1.00	1.00	1.00
C_ASL	1.00	1.00	1.00
C_BSL	1.00	1.00	1.00
D_ASL	1.00	0.81	0.89
D_BSL	1.00	1.00	1.00
E_ASL	0.74	0.54	0.62
E_BSL	1.00	0.85	0.92
F_ASL	0.96	1.00	0.98
F_BSL	0.96	1.00	0.98
G_ASL	0.93	1.00	0.96
G_BSL	1.00	1.00	1.00
H_ASL	0.95	0.73	0.83
H_BSL	1.00	1.00	1.00
I_ASL	0.96	0.96	0.96
I_BSL	1.00	0.69	0.82
K_ASL	0.83	0.96	0.89
K_BSL	1.00	0.96	0.98
L_ASL	0.96	0.96	0.96
L_BSL	0.93	0.96	0.94
M_ASL	0.50	1.00	0.67
M_BSL	0.87	0.50	0.63
N_ASL	0.52	0.62	0.56
N_BSL	0.62	0.96	0.76
O_ASL	0.90	1.00	0.95
O_BSL	0.64	0.96	0.77
P_ASL	1.00	1.00	1.00
P_BSL	0.96	1.00	0.98
Q_ASL	0.83	0.96	0.89
Q_BSL	1.00	1.00	1.00
R_ASL	1.00	0.81	0.89
R_BSL	1.00	1.00	1.00
S_ASL	0.89	0.62	0.73
S_BSL	1.00	0.92	0.96
T_ASL	1.00	0.42	0.59
T_BSL	1.00	1.00	1.00
U_ASL	0.88	0.88	0.88
U_BSL	1.00	0.96	0.98
V_ASL	0.83	0.77	0.80
V_BSL	0.95	0.81	0.88
W_ASL	0.79	0.88	0.84
W_BSL	1.00	0.85	0.92
X_ASL	0.96	1.00	0.98
X_BSL	0.83	0.96	0.89
Y_ASL	0.96	1.00	0.98
Y_BSL	1.00	1.00	1.00

Fig. 18: Precision, recall, F1-score classe per classe

3.5.4 Discussione

I risultati dimostrano chiaramente che la data augmentation ha un impatto significativo sulle prestazioni del modello. L'accuracy con data augmentation è superiore rispetto a quella senza data augmentation, suggerendo che l'augmentation aiuta il modello a generalizzare meglio sui dati di validazione. Il learning rate di 0.001 ha prodotto i migliori risultati. Questo suggerisce che MobileNetV2 riesce a convergere meglio e più rapidamente con un learning rate leggermente più alto, pur rimanendo entro i limiti che evitano instabilità nella formazione. MobileNetV2 si è dimostrato un'ottima scelta per lo svolgimento dell'elaborato grazie alla sua struttura efficiente e alla capacità di ottenere alte prestazioni anche con configurazioni diverse.

3.6 Altre reti utilizzate

- Roboflow: il training di Roboflow si è dimostrato una conferma dell'applicazione di MobileNetV2. Proprio perché Roboflow 2.0 Classification fa uso di questa rete, applicando il training sul dataset è stata raggiunta un accuracy del 93.5%, molto vicina all'accuracy più alta presente nel capitolo precedente. Di seguito sono mostrati i grafici ottenuti.

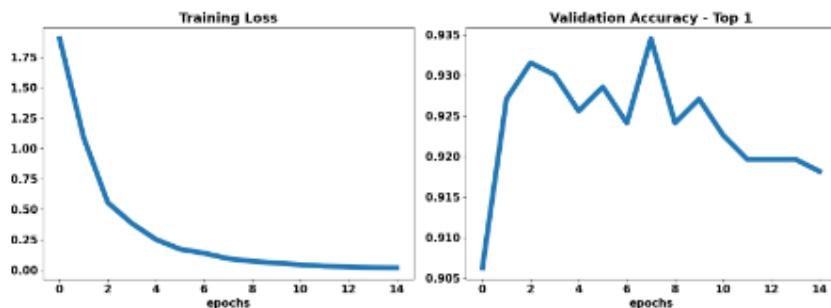


Fig. 19: Risultati del training di Roboflow

- LeNet-5: è una delle prime e più influenti architetture di reti neurali convoluzionali (CNN), sviluppata da Yann LeCun, Léon Bottou, Yoshua Bengio e Patrick Haffner nel 1998. Questo modello è stato progettato specificamente per il riconoscimento di caratteri scritti a mano e ha rappresentato un punto di svolta nel campo del deep learning e del riconoscimento di immagini. È suddiviso in sette livelli, le sue caratteristiche principali sono:
 - Convoluzione: permette di estrarre caratteristiche locali dalle immagini, mantenendo le informazioni spaziali;
 - Pooling: riduce la dimensione delle mappe di caratteristiche, migliorando l'efficienza computazionale e riducendo l'overfitting;
 - Connessioni Locali: le connessioni locali tra strati convoluzionali e di pooling aiutano a catturare le caratteristiche gerarchiche delle immagini;
 - Strati Completamente Connessi: integrano le caratteristiche estratte e facilitano la classificazione finale.

Il modello è stato implementato tramite il seguente codice:

```
model = Sequential([
    Conv2D(6, (5, 5), activation='relu',
           input_shape=(img_height, img_width, 3)),
    MaxPooling2D(pool_size=(2, 2)),
    Conv2D(16, (5, 5), activation='relu'),
    MaxPooling2D(pool_size=(2, 2)),
    Flatten(),
    Dense(120, activation='relu'),
    Dense(84, activation='relu'),
    Dense(num_classes, activation='softmax')
])
```

All'interno del codice si può osservare la presenza del Pooling, i vari livelli connessi, i filtri (6,16) ed il numero di neuroni usati (120, 84). L'activation è stata testata sia tramite la ReLU, sia tramite la tangente Tanh. L'attivazione ReLU viene usata perché riduce il problema del vanishing gradient, un problema che rallenta l'addestramento a causa dei pesi che diventano molto piccoli, lo svantaggio è che può causare il dying ReLU problem dove i neuroni smettono di aggiornarsi con output sempre a zero. L'attivazione Tanh viene preferita quando le attivazioni sono centrate su zero, facilitando la convergenza, normalizza gli input, ma al contrario del ReLU può portare al vanishing gradient problem ed inoltre è più complessa e costosa. Nei tentativi eseguiti il risultato migliore raggiunto è pari al 62/63% di accuracy, utilizzando la data augmentation ed entrambi gli attivatori ReLu e Tanh. Modificando il modello sostituendo MaxPooling2D con AveragePooling2D il risultato è stato migliore ottenendo un 71% di accuracy da 60 epoche e un learning rate pari a 0.001.

- **AlexNet:** è una CNN profonda che ha rivoluzionato il campo della visione artificiale vincendo la competizione ImageNet Large Scale Visual Recognition Challenge (ILSVRC) nel 2012. Progettata da Alex Krizhevsky, Ilya Sutskever e Geoffrey Hinton, AlexNet ha introdotto varie innovazioni, tra cui l'uso di ReLU come funzione di attivazione, l'uso intensivo del dropout per prevenire l'overfitting e l'addestramento distribuito su più GPU. La rete è composta da cinque livelli convoluzionali seguiti da tre livelli completamente connessi.

Il modello **AlexNet** è stato costruito utilizzando la libreria Keras di TensorFlow, con la seguente architettura:

1. Convoluzione 1: 96 filtri 11x11, stride di 4, attivazione ReLU.
2. MaxPooling 1: Pooling 3x3, stride di 2.
3. Convoluzione 2: 256 filtri 5x5, padding 'same', attivazione ReLU.
4. MaxPooling 2: Pooling 3x3, stride di 2.
5. Convoluzione 3: 384 filtri 3x3, padding 'same', attivazione ReLU.
6. Convoluzione 4: 384 filtri 3x3, padding 'same', attivazione ReLU.

7. Convoluzione 5: 256 filtri 3x3, padding 'same', attivazione ReLU.
8. MaxPooling 3: Pooling 3x3, stride di 2.
9. Flatten: Appiattimento dei tensori 3D in vettori 1D.
10. Dense 1: 4096 unità, attivazione ReLU, dropout 0.6, regolarizzazione L2.
11. Dense 2: 4096 unità, attivazione ReLU, dropout 0.6, regolarizzazione L2.
12. Dense 3: 48 unità, attivazione softmax (48 classi).

Il modello è stato compilato con la funzione di perdita `categorical_crossentropy` e l'ottimizzatore `Adam` con un learning rate iniziale di 0.0001. Durante l'addestramento, sono stati utilizzati i seguenti callback:

- `EarlyStopping`: Per fermare l'addestramento se la performance di validazione non migliorava per 10 epoche.
- `ReduceLROnPlateau`: Per ridurre il learning rate di un fattore di 0.2 se la performance di validazione non migliorava per 5 epoche.
- `LearningRateScheduler`: Per ridurre esponenzialmente il learning rate dopo 20 epoche.

L'addestramento è stato eseguito per 80 epoche.

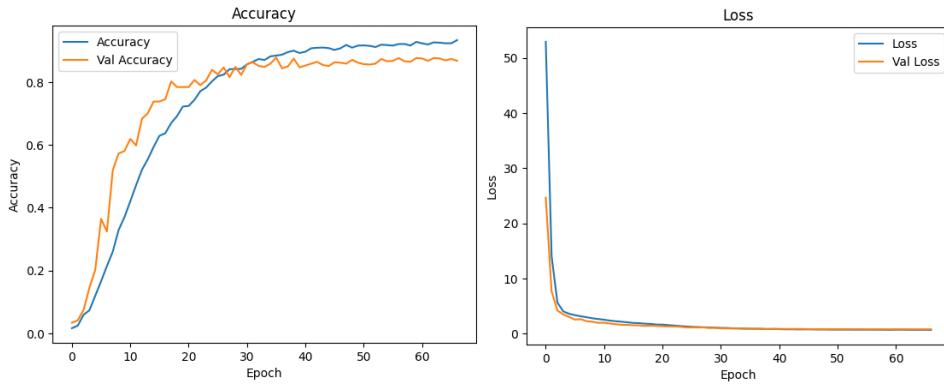


Fig. 20: Esecuzione AlexNet

I risultati dell'addestramento mostrano che il modello ha raggiunto una buona accuratezza sia sul set di training che su quello di validazione, come illustrato nei grafici di accuratezza e perdita:

- Accuracy: L'accuratezza di training e validation ha mostrato un miglioramento costante fino a stabilizzarsi intorno all'epoca 60.
- Loss: La perdita di training e validation è diminuita rapidamente nelle prime epoche e si è stabilizzata successivamente.

L'evaluation finale sul set di test ha mostrato un'accuratezza di circa 88.46%, indicando una buona performance del modello nel riconoscimento delle immagini.

L'applicazione di `AlexNet` al dataset ha prodotto risultati soddisfacenti, con una buona accuratezza di training, validation e test. Le tecniche di data augmentation e i callback utilizzate hanno contribuito a migliorare la generalizzazione del modello e a prevenire l'overfitting. Questi risultati dimostrano l'efficacia di `AlexNet` come modello per il riconoscimento delle immagini anche su dataset complessi e variabili.

4 Classificazione di immagini esterne

La classificazione delle immagini è una tecnica essenziale nel campo del machine learning, soprattutto nell'ambito della visione artificiale. In questa sezione, descriveremo un approccio pratico alla classificazione delle immagini utilizzando il modello di MobileNetV2 che presenta un'accuracy del 90%.

4.1 Introduzione al classificatore

La classificazione delle immagini coinvolge l'identificazione automatica del contenuto visivo di un'immagine e l'assegnazione di una o più etichette predefinite. Con l'avvento delle CNN, questo processo è diventato altamente efficiente, permettendo applicazioni in vari settori come la medicina, la sicurezza e l'automazione industriale. In questo progetto, utilizziamo un modello pre-addestrato e salvato in un file H5 per la classificazione di immagini di segni ASL e BSL. All'interno del codice usato per la classificazione sono presenti i seguenti passi:

- Caricamento del modello: il modello viene caricato utilizzando la funzione `load_model` di Keras. Il percorso del modello salvato in formato H5 viene specificato nella variabile `model_path`, e il modello viene caricato nella variabile `model`.

```
model_path = '/content/drive/My Drive/cnn_model_transfer_learning2.h5'  
model = load_model(model_path)
```

- Preprocessing delle immagini: la funzione `preprocess_image` è responsabile del caricamento e della preparazione delle immagini per la classificazione. Questa funzione prende il percorso dell'immagine (`img_path`) e la ridimensiona a 160x160 pixel, quindi converte l'immagine in un array di numpy. Successivamente, normalizza i valori dei pixel dividendoli per 255 e aggiunge una dimensione extra per conformarsi al formato richiesto dal modello;

```
def preprocess_image(img_path, target_size=(160, 160)):  
    img = image.load_img(img_path, target_size=target_size)  
    img_array = image.img_to_array(img)  
    img_array = np.expand_dims(img_array, axis=0)  
    img_array /= 255.0 # Normalizza i pixel  
    return img_array
```

- Dizionario delle classi: le classi delle lettere ASL e BSL sono definite in una lista. Viene quindi creato un dizionario che associa gli indici delle classi alle etichette corrispondenti, facilitando la traduzione degli output del modello in etichette leggibili;
- Classificazione delle immagini: il codice scorre attraverso una serie di immagini numerate (da 1 a 5). Per ogni immagine, il percorso viene costruito e l'immagine viene preprocessata utilizzando la funzione `preprocess_image`. Il modello predice la classe dell'immagine preprocessata, e l'indice della classe con la probabilità più alta

viene determinato utilizzando argmax. Se l'indice della classe predetta esiste nel dizionario delle classi, l'etichetta corrispondente viene estratta. Infine, l'immagine originale viene visualizzata con l'etichetta della classe predetta.

```

for i in range(1, 5): # Le immagini sono numerate da 1 a 5
    img_path = os.path.join(image_dir, f'{i}.jpg')
    processed_img = preprocess_image(img_path)
    prediction = model.predict(processed_img)

    predicted_class_index = np.argmax(prediction, axis=1)[0]

    if predicted_class_index in class_labels:
        predicted_class_label = class_labels[predicted_class_index]
    else:
        print(f"Predicted class index {predicted_class_index}\
            not found in class labels")
        continue

    img = image.load_img(img_path)
    plt.imshow(img)
    plt.title(f"Predicted Class: {predicted_class_label}")
    plt.show()

    print(f"Image: {img_path}, Predicted Class: {predicted_class_label}")

```

4.2 Classificazione alfabeto ASL

In questa sezione è presente la classificazione delle immagini relative all'alfabeto ASL (American Sign Language). Tutte le immagini per eseguire questi test sono state ricreate o ripescate dal primo tentativo di creazione del dataset.

1. Parola: Albero.

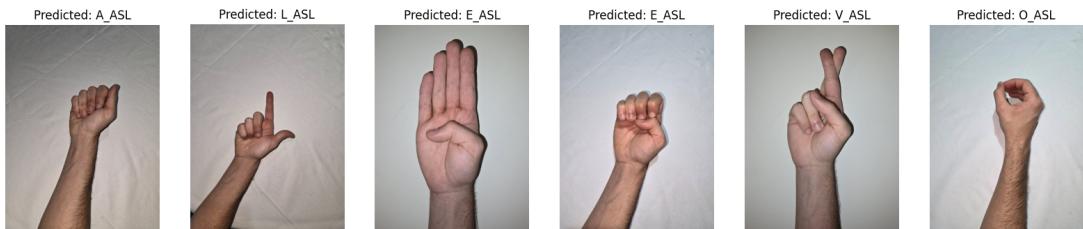


Fig. 21: Albero ASL

Come si può notare dalla figura, sono state classificate correttamente 4 lettere su 6. Questo risultato è dovuto alla somiglianza delle posizioni delle dita in alcune lettere. Ad esempio, nella lettera B (identificata come E), la posizione del pollice è la stessa della lettera E che segue immediatamente. L'algoritmo ha interpretato correttamente l'alfabeto ASL, ma la precisione dell'identificazione per questa parola è del 66,6%.

2. Parola: Ciao.

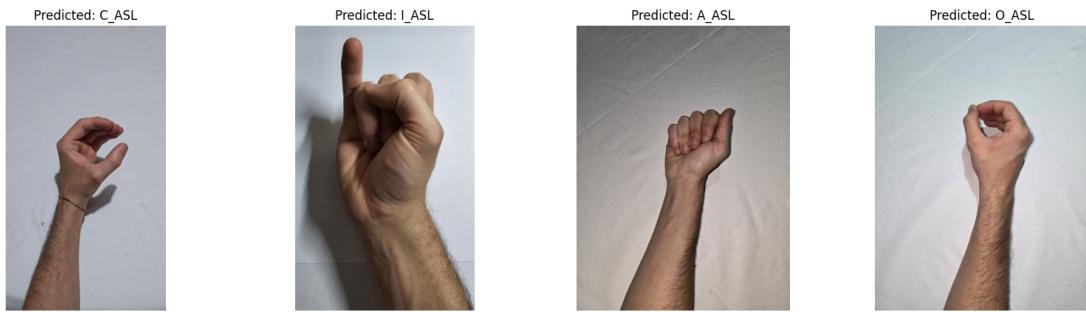


Fig. 22: Ciao ASL

In questo caso l'accuratezza è del 100% riuscendo ad identificare 4 lettere su 4.

3. Parola: Mento.

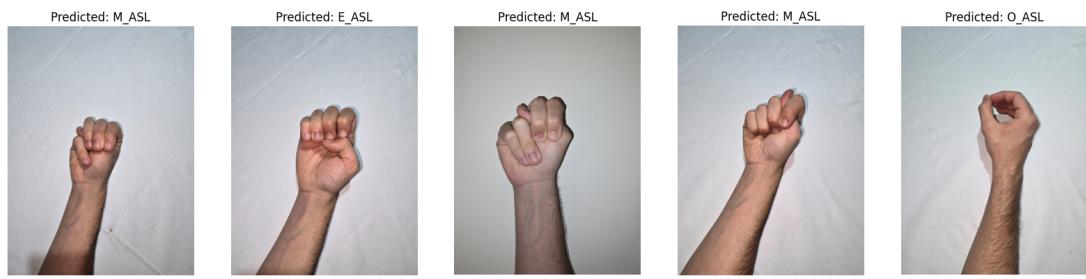


Fig. 23: Mento ASL

La scelta della parola è stata accurata per individuare i difetti più gravi del modello. Nella figura soprastante si può notare la confusione che viene generata tra le lettere M, N e T. La differenza tra queste tre lettere è data dalla posizione del pollice: nella lettera M si trova tra il mignolo e l'anulare, nella lettera N tra l'anulare e il medio, e infine nella lettera T tra il medio e l'indice. Anche in questo caso, l'identificazione dell'alfabeto non ha presentato problemi, con un'accuratezza del 60%.

4.3 Classificazione alfabeto BSL

Come nella sezione precedente, in questo capitolo vengono verificate la classificazione delle immagini relative all'alfabeto BSL. Le immagini usate per il test sono state ricreate o riprese dai primi tentativi di creazione del dataset.

- Parola: Albero.

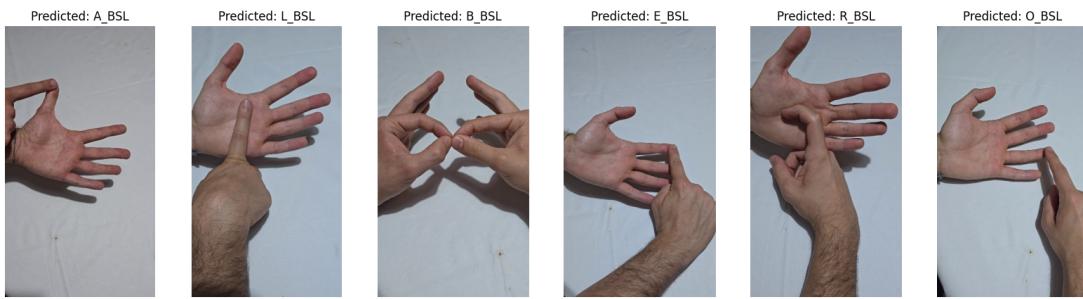


Fig. 24: Albero BSL

La parola è stata correttamente identificata. La difficoltà maggiore poteva derivare dalle somiglianze tra le lettere A, E e O. Tuttavia, il risultato è stato un'accuratezza del 100%, senza alcun dubbio sull'alfabeto utilizzato.

- Parola: Ciao.

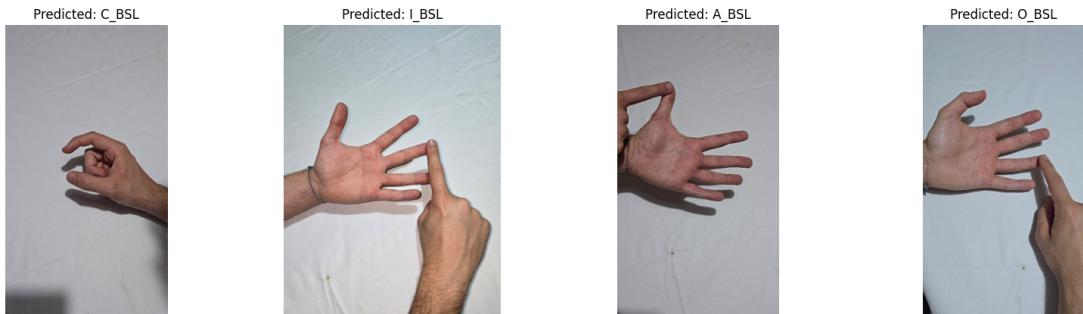


Fig. 25: Ciao BSL

Anche in questo test l'unico problema poteva riguardare le vocali, ma il risultato ottenuto è perfetto, con un'accuratezza del 100%.

- Parola: Mento

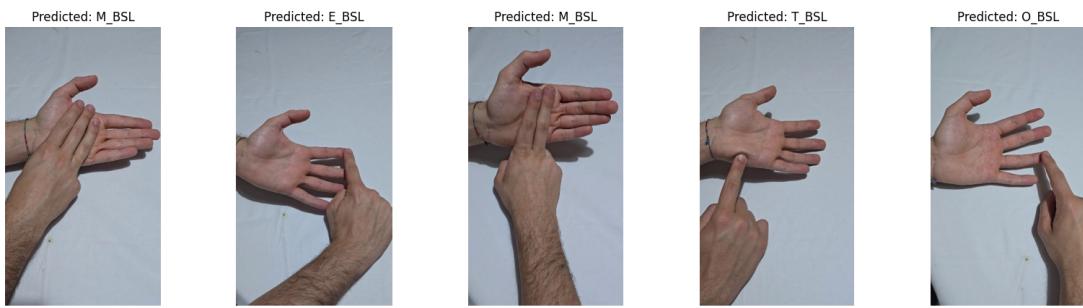


Fig. 26: Mento BSL

Durante questo test si può osservare una confusione del modello nel distinguere la lettera M (all'inizio della frase) dalla lettera N (nella terza posizione). La differenza tra le due è che nella M sono presenti tre dita, mentre nella N solo due. Questa somiglianza causa confusione nel modello, risultando in un'accuratezza dell'80%.

4.4 Classificazione con entrambi gli alfabeti

In questa sezione si vuole verificare la possibilità di formare parole utilizzando entrambi gli alfabeti. Il test è stato eseguito sulla parola 'Mento', scelta per la sua complessità, che presenta le maggiori difficoltà di identificazione per entrambi gli alfabeti. Nonostante i tentativi di correggere le lettere che avevano causato errori nei test precedenti, i risultati non sono cambiati. Il modello continua a confondere la lettera N con la M in entrambi gli alfabeti, mostrando una persistente difficoltà nel distinguere tra le due lettere. Questa confusione è dovuta alla somiglianza delle posizioni delle dita per ASL e delle mani per BSL. Tuttavia, un miglioramento è stato ottenuto utilizzando una variante differente della T in ASL, che è stata correttamente identificata, come illustrato nella figura sottostante. Questo suggerisce che alcune varianti possono aiutare a ridurre gli errori di identificazione. L'accuratezza di entrambi gli alfabeti sulla parola 'Mento' rimane all'80%, nonostante i miglioramenti ottenuti.

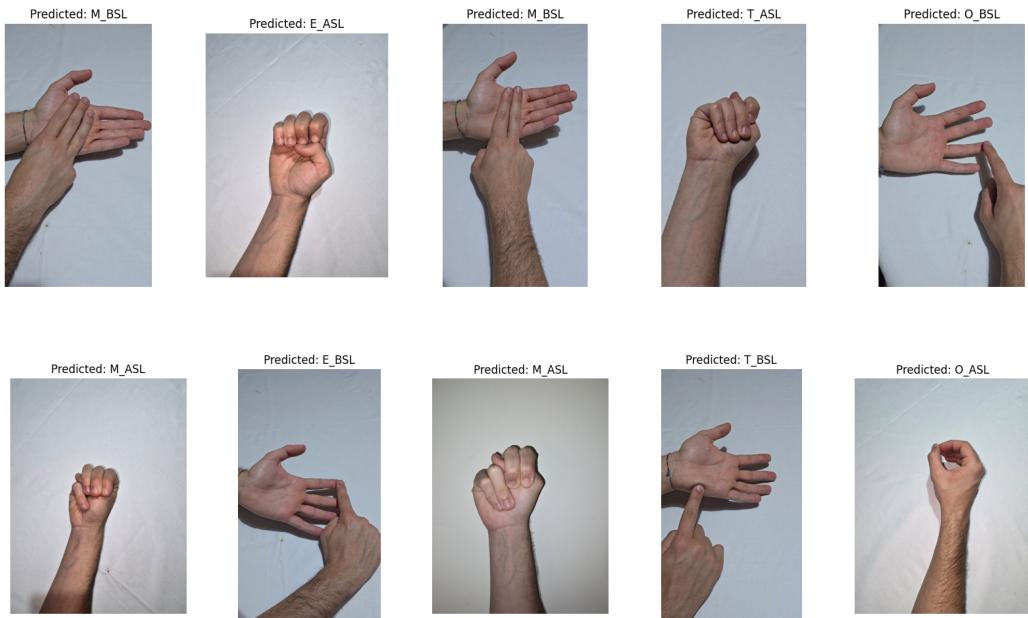


Fig. 27: Mento Mixed

4.5 Classificazione con soggetti non presenti nel dataset

Il dataset utilizzato, come menzionato in precedenza, è basato su un singolo soggetto. In questa sezione, viene esaminata la classificazione di due lettere, una per ciascun alfabeto, per osservare come vengono classificate le mani dei tre soggetti coinvolti nel primo test del dataset. L'obiettivo di questa analisi è verificare se il modello, addestrato su un singolo soggetto, sia in grado di generalizzare e classificare correttamente le lettere eseguite da soggetti diversi.

4.5.1 Alfabeto BSL

Come è possibile osservare nella figura sottostante, il test ha correttamente classificato 2 lettere su 3. La lettera in questione è la A dell'alfabeto BSL. Tuttavia, il modello non

è riuscito a distinguere correttamente la lettera eseguita dal primo soggetto. Questo risultato indica che il modello riesce a generalizzare con un'accuratezza del 66% per l'alfabeto BSL, anche quando viene applicato a mani che non erano presenti nel dataset di addestramento. L'analisi dettagliata dei risultati mostra che, sebbene il modello abbia avuto successo con i soggetti due e tre, ha faticato a riconoscere correttamente la lettera A del primo soggetto. Fattori come la differenza nelle dimensioni delle mani, la posizione delle dita e l'angolazione della mano potrebbero aver influenzato la precisione della classificazione.

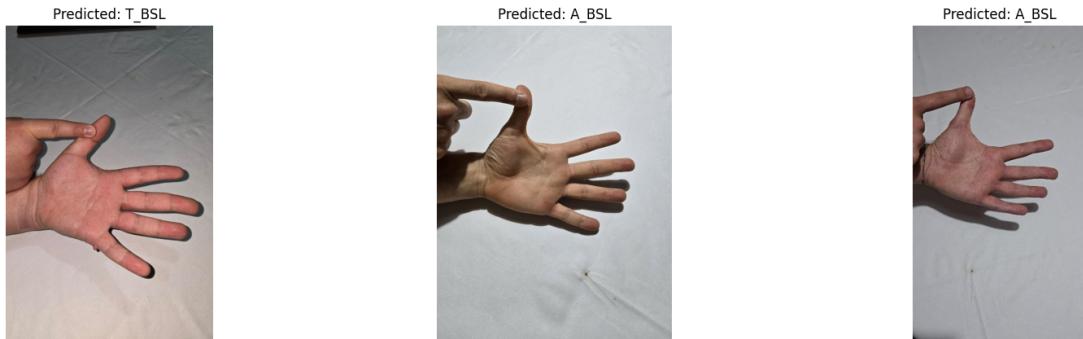


Fig. 28: A BSL in tre soggetti differenti

4.5.2 Alfabeto ASL

In questo esempio, si può osservare che, nonostante il cambio di alfabeto, il risultato rimane invariato. Il modello riesce a classificare correttamente le lettere eseguite dagli ultimi due soggetti, ma fallisce nel caso del primo soggetto. Anche per l'alfabeto ASL, l'accuratezza ottenuta è del 66%. Questa consistenza nei risultati tra i due alfabeti suggerisce che il modello ha una certa robustezza nella classificazione, ma anche che esistono delle sfide comuni che non sono state completamente superate. La difficoltà nel riconoscere la lettera eseguita dal primo soggetto può derivare da diverse variabili, tra cui la differenza nelle dimensioni delle mani, la precisione nei movimenti delle dita, e l'angolazione della mano durante l'esecuzione del segno.

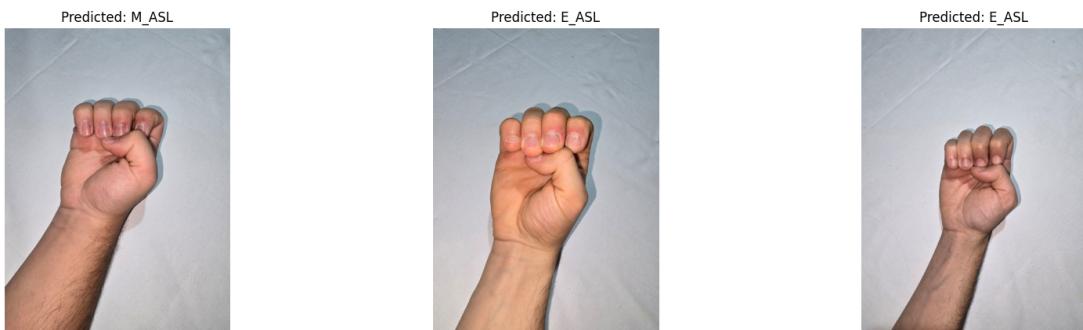


Fig. 29: E ASL in tre soggetti differenti

4.6 Classificazione con DenseNet121

Per valutare e confrontare le prestazioni ottenute con il modello MobileNetV2, è stato necessario eseguire dei test utilizzando un modello alternativo. Il modello scelto per questo confronto è **DenseNet121**, noto per la sua efficienza e accuratezza. I risultati dei test hanno dimostrato che DenseNet raggiunge un'accuratezza di quasi l'82% sul set di test, evidenziando così una performance leggermente inferiore rispetto a **MobileNetV2**. I test eseguiti sono uguali ai precedenti ottenendo i seguenti risultati:

- Parola: Albero.

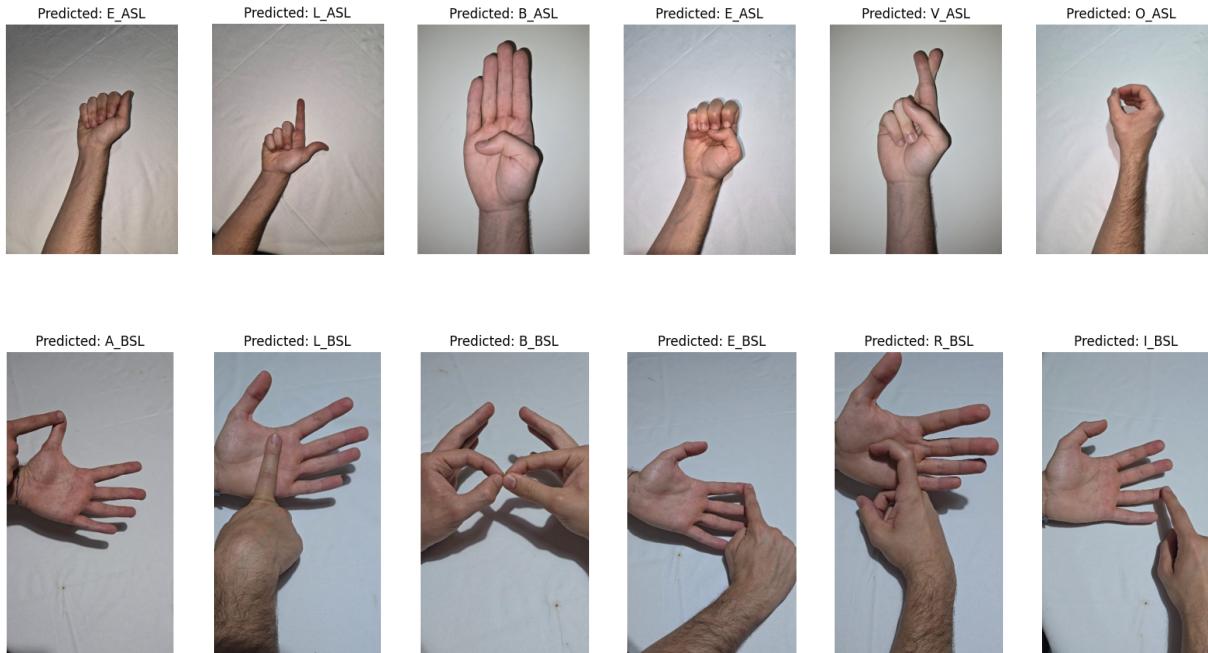


Fig. 30: 1° immagine Albero ASL, 2° immagine Albero BSL

Utilizzando questo modello, la parola "Albero" viene evidenziata correttamente senza problemi, dimostrando una buona capacità di riconoscere l'alfabeto utilizzato. Per quanto riguarda l'alfabeto ASL, il modello ha identificato correttamente 4 lettere su 6, ottenendo un'accuratezza simile a quella del modello precedente. Tuttavia, mentre MobileNetV2 tendeva a confondere la lettera B con la E e la R con la V, questo modello confonde la A con la E e, come MobileNetV2, la R con la V. Questo evidenzia che entrambi i modelli commettono lo stesso tipo di errore con la lettera R. Nell'alfabeto BSL, l'accuratezza è diminuita dal 100% con il modello precedente all'83% con questo nuovo modello. Questa riduzione è dovuta alla confusione tra la lettera O e la lettera I, che ha compromesso la precisione complessiva del riconoscimento delle lettere.

- Parola: Ciao.

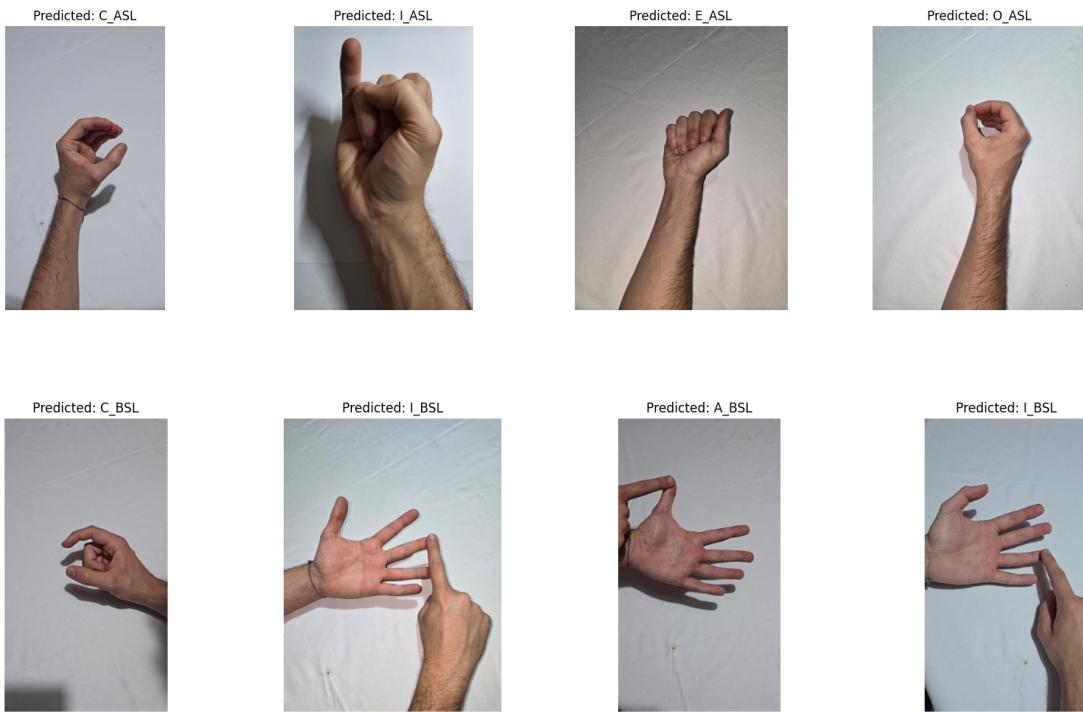


Fig. 31: 1° immagine Ciao ASL, 2° immagine Ciao BSL

Usando questa parola, che contiene diverse vocali, possiamo osservare un'accuratezza per entrambi gli alfabeti che scende dal 100% al 75%. La diminuzione di accuratezza nell'alfabeto ASL è dovuta alla lettera A, che viene classificata erroneamente come E. Nell'alfabeto BSL, invece, l'errore è causato dalla lettera O, che viene identificata come I.

- Parola: Mento.

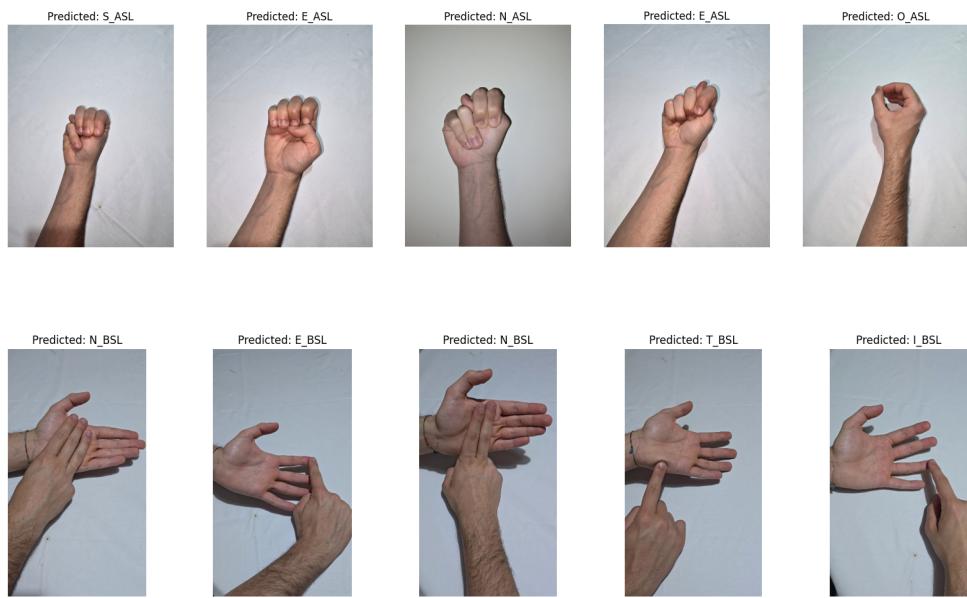


Fig. 32: 1° immagine Mento ASL, 2° immagine Mento BSL

Questo test rappresenta una sfida significativa per entrambi i modelli. Nell’alfabeto ASL, vengono identificate correttamente 3 lettere su 5, mantenendo un’accuratezza pari a quella di MobileNetV2. Anche in questo caso, le incertezze sono dovute alle lettere M, N e T. Per quanto riguarda l’alfabeto BSL, il modello riesce a classificare correttamente lo stesso numero di lettere, ma perde accuratezza rispetto al modello precedente, passando dall’80% al 60%.

In conclusione, si può affermare che il modello DenseNet121 riesce a identificare le lettere quasi allo stesso livello del modello MobileNetV2, mostrando prestazioni migliori su alcune lettere e peggiori su altre. L’accuratezza complessiva del set di test differisce di un 8% a favore di MobileNetV2, evidenziando una leggera superiorità di quest’ultimo, confermata anche durante i test eseguiti.

5 Classificazione con dataset variato

È stato condotto un nuovo studio utilizzando un dataset specificamente creato per valutare la classificazione delle lettere usando un dataset contenente diverse mani. Il dataset comprende 5 lettere (A, B, F, L, R) selezionate per ciascun alfabeto, ciascuna eseguita da due soggetti differenti. Uno dei soggetti è lo stesso utilizzato nel dataset originale, mentre l’altro è il primo soggetto precedentemente incluso nei test di classificazione con soggetti non appartenenti al dataset originale. I dati sono stati raccolti utilizzando la tecnica sfruttata per l’acquisizione del dataset originale, che ha coinvolto la registrazione di 120 frame per ogni lettera, con una frequenza di campionamento di 2 frame al secondo. La divisione dei frame è stata organizzata nel seguente modo: 80 frame sono stati utilizzati per l’addestramento del modello, 14 frame per la fase di validazione e 26 frame per il test finale. Questi valori valgono per un singolo soggetto, quindi sono da raddoppiare per il totale degli esemplari presenti nel dataset. Particolarmente, per il terzo video di ogni lettera, la suddivisione è stata eseguita assegnando il 35% dei frame alla fase di validazione e il restante 65% alla fase di test. L’obiettivo principale di questo studio è quello di valutare la capacità del modello di classificare correttamente le lettere nei due alfabeti, considerando la variazione individuale tra i soggetti. Nel contesto di questo studio, la presenza di due soggetti introduce variazioni significative nelle esecuzioni delle lettere gestuali. Queste differenze possono essere osservate in diversi aspetti, tra cui:

- **Posizione della mano:** i due soggetti potrebbero avere variazioni nella posizione della mano durante l’esecuzione di ciascuna lettera. Questo può influenzare l’angolazione e l’orientamento della mano rispetto alla telecamera;
- **Posizione delle dita:** le posizioni specifiche delle dita, come il pollice, l’indice, il medio, l’anulare e il mignolo, potrebbero variare tra i soggetti. Questa variabilità è particolarmente rilevante per lettere che richiedono configurazioni specifiche delle dita per essere distinte correttamente;
- **Dimensioni e forma della mano:** le differenze nelle dimensioni della mano e nella conformazione delle dita possono influire sull’aspetto visivo delle lettere eseguite. Ad esempio, una mano più grande potrebbe avere una configurazione delle dita leggermente diversa rispetto a una mano più piccola.

5.1 Addestramento del dataset

Per l'addestramento del modello di classificazione delle lettere gestuali, è stata utilizzata la rete preaddestrata **MobileNetV2**. Questa scelta è motivata dalla sua efficienza computazionale e dalle buone prestazioni nella visione artificiale. La configurazione dei parametri include dimensioni delle immagini di 160x160 pixel, un batch size di 64 e un numero massimo di epoche pari a 30. Durante il processo di addestramento, è stata implementata la tecnica di data augmentation sul training set. In aggiunta alla **MobileNetV2** preaddestrata, sono stati introdotti ulteriori livelli per adattare il modello specificamente al compito di classificazione delle lettere. Questi livelli includono un Global Average Pooling per ridurre la dimensionalità dell'output della **MobileNetV2**, seguito da un livello Dense con 512 unità e funzione di attivazione ReLU per apprendere rappresentazioni più complesse delle caratteristiche delle lettere. È stato aggiunto anche un livello di Batch Normalization per accelerare la convergenza del modello durante l'addestramento, seguito da un Dropout del 50% per ridurre il rischio di overfitting. Infine, il modello è stato completato con un livello Dense di output con un numero di classi pari a 10 (considerando le lettere gestuali da classificare), utilizzando l'attivazione softmax per ottenere le probabilità predette per ciascuna classe. Per quanto riguarda l'ottimizzazione, è stato configurato un learning rate di 0.0001 per regolare l'aggiornamento dei pesi del modello durante l'addestramento. Questo parametro è cruciale per determinare la velocità di convergenza dell'algoritmo di ottimizzazione durante la discesa del gradiente.

5.2 Risultati dell'addestramento

I risultati dell'addestramento mostrano una discesa lineare della loss e un'accuracy del 98%, come evidenziato nei grafici riportati di seguito:

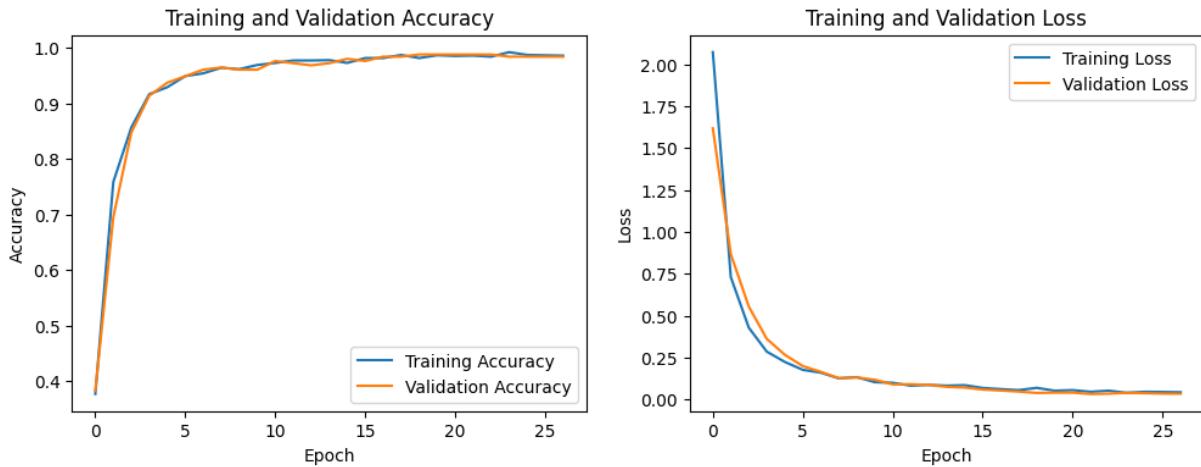


Fig. 33: Grafici riguardanti l'addestramento tramite MobileNetV2

Analizzando la matrice di confusione, è emerso che la maggior parte delle lettere viene identificata senza difficoltà. L'unica eccezione si verifica tra la lettera R_ASL e B_ASL, che risultano occasionalmente confuse. Gli indicatori F1-score, precision e recall mostrano valori quasi perfetti per la maggior parte delle lettere.

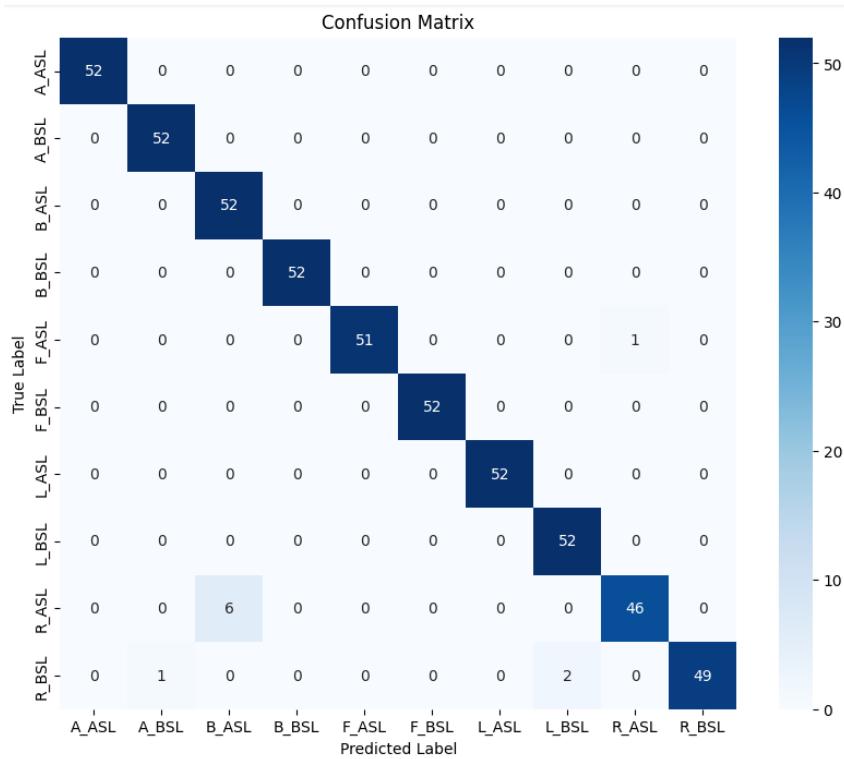


Fig. 34: Matrice di confusione

	precision	recall	f1-score
A_ASL	1.00	1.00	1.00
A_BSL	0.98	1.00	0.99
B_ASL	0.90	1.00	0.95
B_BSL	1.00	1.00	1.00
F_ASL	1.00	0.98	0.99
F_BSL	1.00	1.00	1.00
L_ASL	1.00	1.00	1.00
L_BSL	0.96	1.00	0.98
R_ASL	0.98	0.88	0.93
R_BSL	1.00	0.94	0.97

Fig. 35: Precision e Recall per ogni classe

5.3 Risultati della classificazione

Una volta finita la fase di addestramento, si passa alla fase di test della classificazione. I tentativi eseguiti vengono effettuati su tutti e tre i soggetti, come nei test precedenti, per verificare che, in questo caso, il primo soggetto venga identificato essendo all'interno del dataset. I test sono divisi nei due alfabeti:

- Alfabeto BSL:

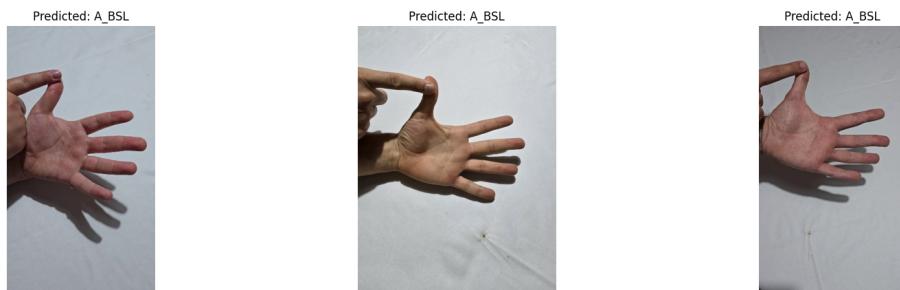


Fig. 36: A BSL

Nel test dell’alfabeto BSL, si può osservare un risultato significativo: la lettera A BSL viene correttamente identificata in tutti e tre i soggetti. Questo rappresenta un notevole miglioramento rispetto ai test condotti con il dataset precedente, dove la lettera A BSL non veniva riconosciuta con la stessa precisione. L’accuratezza della classificazione per la lettera A BSL in ciascuno dei soggetti è stata del 100%.

- Alfabeto ASL:

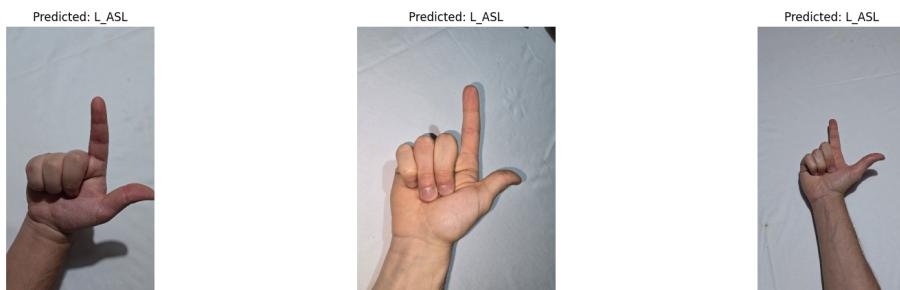


Fig. 37: L ASL, Dataset con due soggetti

Anche nell’alfabeto ASL tutti i soggetti sono stati correttamente identificati, raggiungendo il massimo dell’accuratezza. Questo risultato indica che il sistema di classificazione è efficace non solo con l’alfabeto BSL, ma anche con quello ASL, dimostrando una notevole versatilità e robustezza del modello. Per un confronto dettagliato, è stato riutilizzato il modello con il dataset principale. In questo scenario, come visibile nella figura sottostante, il primo soggetto non è stato correttamente identificato, evidenziando le limitazioni del dataset precedente.

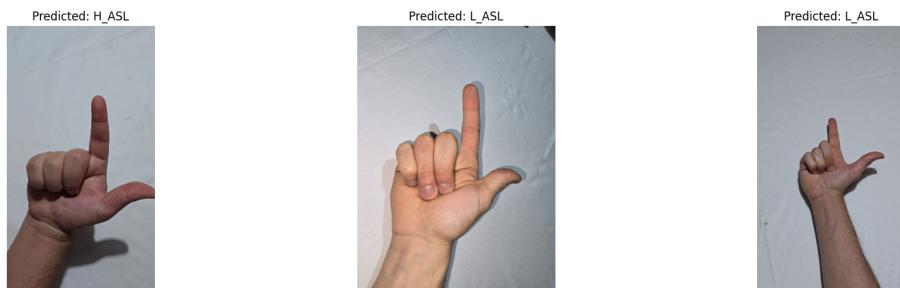


Fig. 38: L ASL, Dataset con un solo soggetto

5.4 Discussione

È facile osservare dai risultati appena analizzati che il dataset contenente molteplici soggetti offre una precisione superiore nell'identificazione dei vari individui. Questo dataset riesce a superare efficacemente i problemi legati alla posizione della mano, delle dita, nonché alla dimensione e forma della mano e delle dita. In particolare, l'inclusione di una maggiore varietà di soggetti nel dataset ha permesso al modello di gestire meglio le variazioni individuali nelle esecuzioni dei segni. Questo ha portato a un sistema di classificazione più robusto, capace di adattarsi alle diverse configurazioni delle mani e delle dita che possono presentarsi in situazioni reali. Un dataset di dimensioni più grandi, con una rappresentazione ancora più ampia di varianti di segni, posizioni e caratteristiche fisiche, permetterebbe al sistema di affrontare con successo un maggior numero di scenari reali.

6 Conclusione

Per concludere, possiamo riassumere i risultati ottenuti in due punti principali:

- **L'apprendimento:** l'analisi è stata eseguita su diverse reti neurali, tra cui LeNet5, MobileNetV2, VGG16, AlexNet e DenseNet121. Ogni rete è stata testata con vari parametri, come learning rate, dimensioni delle immagini, batch size e numero di epoche, per ottimizzare le loro prestazioni nel contesto del progetto. I risultati migliori sono stati ottenuti con MobileNetV2 (93%), AlexNet (91%) e VGG16 (97%). Le differenze tra queste reti neurali risiedono principalmente nella loro architettura e nella gestione delle convoluzioni e dei parametri. MobileNetV2 utilizza convoluzioni depthwise separabili per migliorare l'efficienza computazionale e ridurre il numero di parametri, mantenendo al contempo alte prestazioni di classificazione. VGG16 e AlexNet, d'altra parte, impiegano architetture più tradizionali con convoluzioni profonde. La differenza sostanziale tra queste reti risiede nella velocità di addestramento: MobileNetV2 richiede solo 60 secondi per epoca, mentre AlexNet e VGG16 impiegano ciascuna 90 secondi per epoca. Possiamo concludere che l'architettura MobileNetV2 è potenzialmente la migliore per il progetto, non solo per l'elevata precisione ottenuta (93%), ma anche per la notevole velocità con cui questo risultato è stato raggiunto.
- **La classificazione:** l'analisi della classificazione ha dimostrato che, utilizzando un dataset basato su un singolo soggetto, è possibile identificare anche soggetti esterni al dataset che presentano una fisionomia e una posizione della mano simili a quella del soggetto originale. Tuttavia, questa capacità di generalizzazione ha dei limiti, poiché la variabilità tra individui non è completamente catturata con un solo soggetto. Quando il dataset viene ampliato per includere due soggetti, il modello diventa più robusto. In questo caso, riesce a identificare correttamente soggetti che in precedenza non erano riconosciuti con precisione. L'inclusione di un secondo soggetto introduce una maggiore diversità nel dataset, permettendo al modello di apprendere una gamma più ampia di variazioni nelle fisionomie e nelle posizioni delle mani. Questo migliora significativamente la capacità del modello di distinguere tra differenti individui e gesti, rendendolo più efficace e affidabile. Le prove di classificazione hanno inoltre confermato un aspetto fondamentale: l'assenza

di incertezze riguardo all’alfabeto dei segni utilizzato, sia esso ASL o BSL. Questo risultato indica che il modello è in grado di riconoscere i segni con elevata precisione, indipendentemente dal sistema di linguaggio dei segni utilizzato.

6.1 Sviluppi futuri

Gli sviluppi futuri riguarderanno l’ampliamento del dataset e l’incremento del numero di soggetti inclusi. Attualmente, il modello si basa su un numero limitato di soggetti, il che può limitare la sua capacità di generalizzare su una popolazione più ampia. Per migliorare questa capacità, sarà necessario includere una varietà maggiore di soggetti con caratteristiche fisiche e demografiche differenti, come età, genere e diverse conformazioni delle mani. Inoltre, sarà essenziale raccogliere dati che rappresentino una gamma più ampia di posizioni delle mani e delle dita, includendo anche varianti di gesti che possono differire leggermente tra loro. Questo permetterà al modello di riconoscere con maggiore precisione e affidabilità i gesti in diverse condizioni e da diverse angolazioni. L’obiettivo finale è rendere il modello idoneo per applicazioni in tempo reale. Questo richiede non solo un miglioramento dell’accuratezza, ma anche ottimizzazioni significative in termini di velocità di elaborazione e consumo di risorse computazionali. Inoltre, l’espansione del dataset e il miglioramento del modello apriranno la strada a nuove applicazioni pratiche nell’ambito del riconoscimento gestuale. Ad esempio, questo potrebbe includere l’integrazione in dispositivi di assistenza per persone con disabilità, sistemi di controllo gestuale per la domotica, interfacce uomo-macchina avanzate e applicazioni di realtà aumentata.

6.2 Struttura GitHub

All’interno del repository di GitHub sono contenuti il codice e i dati utilizzati durante lo sviluppo dell’elaborato, compresi i codici per l’addestramento e quelli riguardanti l’estrapolazione dei frame dal video. La repository è divisa in tre cartelle:

- **Dataset:** contiene il dataset presente all’interno di Roboflow, diviso a sua volta in train, test e validation;
- **Strumenti utilizzati:** all’interno di questa cartella sono contenuti gli script sviluppati per poter estrarre i frame dai video;
- **Notebook:** Il contenuto della cartella include tutti i notebook creati, ognuno dei quali è dedicato a una rete neurale specifica. Oltre ai vari notebook delle reti, è presente il notebook chiamato "Demo.ipynb". Questo permette di testare i modelli attraverso due modalità:
 1. **Upload di un’immagine:** carica un’immagine inerente a uno dei due alfabeti per verificarne l’identificazione.
 2. **Selezione di un alfabeto e una parola:** vengono prese le immagini che rappresentano quella parola e identificate con i tre modelli MobileNetV2, VGG16 e DenseNet121.

Luigi Priano (1000002081), Gianluca Giardina (1000015165),
Giuseppe Napoli (1000012802)

Per la sua esecuzione basta aprire il collab ed eseguirlo. Gli altri notebook all'interno della cartella sono:

- **MobileNetV2.ipynb**: il notebook contiene l'addestramento con **MobileNetV2** e, successivamente, la classificazione utilizzata per l'elaborato, che coinvolge sia **MobileNetV2** che **DenseNet121**;
- **MobileNetV2Dataset2.ipynb**: questo notebook è stato utilizzato per addestrare **MobileNetV2** con il dataset variato contenente più soggetti;
- **densenet.ipynb**: presenta l'addestramento di **DenseNet121**;
- **VGG6.ipynb**: notebook contenente gli sviluppi della rete **VGG16**;
- **AlexNet.ipynb**: notebook contenente gli sviluppi della rete **AlexNet**;

7 Riferimenti

- [1] ProjectML_ASLvsBSL, Luigi Priano, Gianluca Giardina, Giuseppe Napoli, (2024)
https://github.com/GianlucaGiardina/ProjectML_ASLvsBSL
- [2] What is MobileNetV2? Features, Architecture, Application and More, Nitika Sharma (2024)
<https://www.analyticsvidhya.com/blog/2023/12/what-is-mobilenetv2/>
- [3] VGG-16 CNN Networks | Deep Learning Engineer Italia, Andrea Provino (2020)
<https://www.andreaprovino.it/vgg-16-cnn-networks-deep-learning-engineer-italia>
- [4] AlexNet CNN Networks – Deep Learning Engineer Italia, Andrea Provino (2020)
<https://www.andreaprovino.it/alexnet>