



UNIVERSITÀ DEGLI STUDI DI SALERNO
DIPARTIMENTO DI INFORMATICA

Tesi di Laurea Magistrale in Informatica
Curriculum Data Science

**Riconoscimento Real-Time della
Lingua dei Segni mediante
Rete Neurale Spazio-Temporale**

Relatori

Prof. Vincenzo Deufemia
Ing. Antonio Grimaldi

Candidato

Simone Finelli

Anno Accademico 2019-2020

Ringraziamenti

Un caloroso ringraziamento va alla mia famiglia, pilastro fondamentale della mia vita, senza la quale tutto questo non sarebbe stato possibile.

Alla mia fidanzata Martina, che ha saputo darmi la forza di andare avanti, anche nei momenti più difficili della mia vita.

A Davide, che in questi due anni ha condiviso con me gioie e dolori.

Al mio relatore Prof. Vincenzo Deufemia, che ha dato fiducia al progetto, rendendosi disponibile in qualsiasi situazione.

Ai tutor aziendali Ing. Antonio Grimaldi e Ing. Antonio Schiano, che mi hanno fornito il supporto necessario per affrontare il progetto.

Un sentito ringraziamento va a tutto lo staff di NTT DATA che ha permesso lo svolgimento del tirocinio esterno e lo sviluppo di questa tesi.

Abstract

La lingua dei segni è uno dei principali mezzi utilizzati dalle persone non udenti per relazionarsi con il mondo esterno. Così come le lingue parlate, le lingue dei segni presentano una propria sintassi e delle regole ben precise. Questo comporta che, per poter interloquire, una persona udente deve necessariamente conoscere la lingua dei segni utilizzata dal non udente.

Data l'evidente difficoltà nel relazionarsi tra persone udenti e non, questa tesi nasce con lo scopo di accorciare la distanza, che ad oggi esiste, tra le due parti. Tale obiettivo è stato raggiunto con la creazione di un sistema in grado di tradurre real-time la lingua dei segni nel corrispettivo testo. In particolare, l'approccio proposto è in grado di effettuare una traduzione parola per parola della lingua dei segni americana all'interno di uno streaming video.

Per l'implementazione del sistema sono state coinvolte due grandi macro-aree dell'intelligenza artificiale: la computer vision e la comprensione del linguaggio naturale. Dalla fusione di queste due aree è stato progettato ed implementato il motore inferenziale, cuore dell'applicazione real-time. In particolare, la rete neurale proposta è formata da una rete convoluzionale ed una rete ricorrente. La prima si occupa di estrarre, da una sequenza di frame, le feature spaziali al fine di individuare la persona che parla con i segni; la seconda, invece, si occupa di trovare le relazioni temporali, che sussistono tra le feature spaziali, al fine di rilevare il movimento effettuato dalla persona, per poi classificarlo.

Il sistema proposto è stato sperimentato utilizzando un dataset con una quantità di samples tale da poter generalizzare sia il contesto che la persona. Il dataset considerato si chiama *Word-Level American Sign Language* (WLA-SL) ed è composto da più di 20.000 video ripartiti tra 2.000 *glosses* (parole

di senso compiuto nella lingua dei segni americana). Inoltre, tale dataset è stato ripartito, in base al numero di glosses, in quattro sotto-dataset (100, 300, 1.000, 2.000 glosses), con l’obiettivo di poterne apprezzare la complessità di utilizzo.

Al fine di testare e valutare qualitativamente il modello inferenziale proposto, sono stati utilizzati come riferimento i risultati ottenuti dagli esperimenti degli autori del dataset. In particolare, su tutti e quattro i sotto-dataset, l’approccio proposto ha dimostrato avere performance migliori, ottenendo un aumento dell’accuracy di circa l’8~10% per i primi due, del 5% sul terzo, fino ad arrivare al 2% sull’intero dataset.

Indice

1	Introduzione	1
2	Stato dell'Arte	5
3	Dataset	8
3.1	Scelta del Dataset	8
3.2	Word-Level American Sign Language	9
3.2.1	Processo di Raccolta dei Video	10
3.2.2	Processo di Annotazione dei Video	10
3.2.3	Caratteristiche del Dataset	11
4	Background	13
4.1	Convolutional Neural Networks	13
4.1.1	Convolutional Layer	14
4.1.2	Pooling Layer	17
4.1.3	Architettura tipica di una ConvNet	17
4.1.4	Data Augmentation	18
4.1.5	Transfer Learning	19
4.2	Recurrent Neural Networks	21
4.2.1	Long Short-Term Memory	22
4.2.2	Gated Recurrent Unit	26
5	Modello Proposto	28
5.1	Fase 1: Frame Sampling	29
5.1.1	Custom Frame Generator	29

5.1.2	Data Augmentation	32
5.2	Fase 2: Estrazione delle feature Spaziali	33
5.2.1	Time Distributed Layer	33
5.2.2	MobileNetV2	36
5.3	Fase 3: Estrazione delle feature temporali	38
5.3.1	Tipi di GRU	38
5.4	Fase 4: Classificazione	41
5.5	Modello Finale	41
5.5.1	Implementazione	43
6	Risultati	46
6.1	WLASL 100	47
6.2	WLASL 300	48
6.3	WLASL 1000	49
6.4	WLASL 2000	50
6.5	Risultati Finali	51
6.6	WLASL 20 Custom	52
7	Web App	55
7.1	Server	56
7.1.1	Flask	56
7.1.2	Flask-SocketIO	56
7.2	Client	58
8	Conclusioni	63

Capitolo 1

Introduzione

La lingua dei segni è utilizzata da oltre 72 milioni di persone non udenti nel mondo, l'80% delle quali vive nei Paesi in via di sviluppo [1]. Esistono all'incirca trecento lingue ‘non parlate’ [46], ognuna delle quali può essere considerata una lingua a se stante, strutturalmente distinta dalle lingue parlate. Infatti, è luogo comune pensare che le lingue dei segni dipendano e derivino in qualche modo dalle lingue parlate e, di conseguenza, che siano le lingue parlate stesse ad essere espresse in segni, ma tutto ciò è sbagliato. Questo perché, sebbene, le lingue dei segni prendino in prestito qualcosa anche dalle lingue parlate, esse presentano la propria sintassi e grammatica. Tali lingue, infatti, sono sviluppate dalle persone che le usano e non da persone udenti. Ad esempio, la lingua dei segni americana (American Sign Language - ASL) condivide più sintassi con il giapponese parlato che con l'inglese [43]. Così come per la lingua parlata, dove l'inglese è la lingua più diffusa, anche tra le lingue dei segni quella americana è la più utilizzata. Infatti essa è utilizzata, oltre che negli USA, anche in Canada, Messico e 19 altri Paesi [13].

Dato il grande numero di persone coinvolte nell'utilizzo di queste lingue è automatico pensare come tutto ciò sia di grande interesse per la comunità scientifica. In particolare, si sta cercando di colmare il divario che tuttora esiste tra le persone udenti e non; con un particolare interesse nel garantire maggiore accessibilità per questi ultimi nel mondo moderno. Esistono già in letteratura alcuni studi che cercano di risolvere il difficile problema del

riconoscimento del linguaggio dei segni, ma la maggior parte di essi si basano su dispositivi che non sempre sono alla portata di tutti. Ad esempio, tra questi troviamo studi che fanno uso di telecamere di profondità [23, 44, 47], oppure di guanti colorati per distinguere nettamente le mani della persona [45]. Tuttavia, tali requisiti vanificano l'applicabilità dell'approccio stesso. Per tale motivo, si sta cercando di puntare al riconoscimento dei segni attraverso tecniche indipendenti da strumentazione esterna, basandosi solamente sull'input di una normale fotocamera o videocamera. Ad oggi, tutto questo è possibile grazie alla comparsa di metodi di visione artificiale che utilizzano il deep learning come strategia di elaborazione.

In maniera generale, il problema del riconoscimento della lingua dei segni può essere affrontato attraverso tre approcci principali:

- il riconoscimento della lingua dei segni a **livello di carattere**, dove il sistema ha la capacità di riconoscere le singole lettere dell'alfabeto;
- riconoscimento della lingua dei segni a **livello di parola**, dove il sistema ha la capacità di riconoscere intere parole;
- riconoscimento della lingua dei segni a **livello di frase**, dove il sistema ha la capacità di riconoscere intere frasi.

In questo studio di tesi si è posta l'attenzione all'attività di riconoscimento della lingua dei segni americana (ASL) a livello di parola. La scelta della lingua è facilmente intuibile in quanto, essendo quella più utilizzata, coinvolge automaticamente un numero maggiore di persone.

Per quanto riguarda l'approccio, invece, è stata necessaria una fase iniziale di approfondimento dello stato dell'arte dei lavori esistenti. Da questo studio iniziale è emerso che lavorare a livello di singolo carattere è un approccio ben conosciuto nella comunità scientifica ed è già stato ampiamente utilizzato. Nonostante ciò, tale approccio è molto limitato e non consente di essere utilizzato per applicazioni reali: sarebbe impensabile tradurre interi discorsi lettera per lettera.

Di contro, affrontare il riconoscimento della lingua dei segni a livello di frase è molto difficile in quanto si ha bisogno di lavorare con gli esperti del

dominio: interpreti, persone non udenti, insegnati, etc. Questo perché, come detto in precedenza, le lingue dei segni sono lingue a se stanti con le proprie regole, dunque, pensare in termini di lingua parlata porterebbe solo a risultati non utilizzabili nella pratica.

Inoltre, per poter trascrivere un discorso non basterebbe catturare semplicemente la posizione delle braccia e delle mani, ma anche altre biometrie, come, ad esempio, il voto e le sue espressioni. Ad esempio, nella lingua americana parlata la domanda viene posta cambiando il tono di voce o modificando l'ordine delle parole; di contro le persone non udenti che usano ASL pongono una domanda alzando le sopracciglia, allargando gli occhi e inclinando il corpo in avanti [29].

Dopo aver fatto tutte queste osservazioni, la scelta dell'approccio da seguire è ricaduta sul secondo approccio, cioè il riconoscimento della lingua dei segni a livello di singola parola. Anche se l'approccio potrebbe sembrare più semplice del terzo, esso presenta comunque delle difficoltà non indifferenti:

- il significato dei segni dipende principalmente dalla combinazione dei movimenti del corpo, dei movimenti manuali e della posizione della testa;
- il vocabolario dei segni nell'uso quotidiano è ampio e di solito nell'ordine di migliaia di parole, e sottili differenze possono renderne difficile il riconoscimento;
- una parola nella lingua dei segni può avere più controparti nelle lingue naturali.

Per poter creare un modello in grado di catturare tali sottigliezze bisogna avere a disposizione un dataset corposo ed omogeneo. Tali caratteristiche sono state ritrovate nel recente studio di Dongxu Li *et al.* [25] che ha proposto il dataset WLASL. Quest'ultimo è composto da migliaia di video, nei quali i signers (persone che conoscono il linguaggio dei segni) annotano le varie parole. Si lascia, però, l'approfondimento delle caratteristiche di tale dataset nei prossimi capitoli.

Grazie al dataset WLASL, si è riusciti a creare un sistema di riconoscimento real-time dei segni a livello di parola. Il tutto è stato implementato sfruttando solo lo streaming di una webcam, evitando strumenti o tecniche specifiche, come la stima della posa o degli stimoli facciali. Il modello inferenziale implementato per l'applicazione real-time è nato alla fusione di due grandi aree dell'intelligenza artificiale: la Computer Vision ed il Natural Language Processing (NLP). In particolare, sono state sfruttate le Convolutional Neural Network (CNN o ConvNet), per poter catturare le feature spaziali, e le Gated Recurrent Units (GRU) per poter catturare le feature temporali (individuazione dei movimenti dei signer effettuati nel tempo). In sostanza, quello che si è fatto, è stato sfruttare informazioni spaziali e temporali al fine di individuare in una sequenza di frame (video) la parola segnata.

I risultati ottenuti mostrano come l'approccio utilizzato può essere in grado di riconoscere in tempo reale le parole segnate da una persona posizionata frontalmente rispetto all'inquadratura. Inoltre, il modello di rete neurale proposto riesce a superare le performance dell'approccio presentato in [25], incrementando l'accuracy da 8,44% a 10,07%.

Questo documento di tesi è organizzato nel modo seguente: il secondo capitolo effettua una ricapitolazione delle tecniche più utilizzate per il riconoscimento della lingua dei segni. Il terzo capitolo descrive in maniera approfondita il dataset utilizzato. Il quarto capitolo fornisce al lettore il background necessario per affrontare i capitoli successivi, focalizzandosi sulle reti neurali convoluzionali e le reti neurali ricorrenti. Il quinto presenta l'architettura del modello proposto, mentre il sesto capitolo presenta i risultati sperimentali ottenuti con il dataset WLASL. Il settimo capitolo descrive la WebApp sviluppata per testare il sistema. Infine, l'ultimo capitolo riassume le conclusioni ed i possibili sviluppi futuri.

Capitolo 2

Stato dell'Arte

Da decenni lo studio del riconoscimento della lingua dei segni è stato sempre al centro della comunità scientifica. Infatti, già nel 1983, fu presentato un primo lavoro di riconoscimento sfruttando dei guanti meccanici in grado di tradurre la lingua dei segni americana [15]. Con il passare del tempo, sempre più approcci sono stati pensati ed implementati. Ad esempio, i lavori [5, 9] estendono quello proposto nel '83, effettuando il riconoscimento della lingua dei segni basandosi sui movimenti tracciati attraverso dei guanti sensoriali.

Pochi anni dopo, si è cominciato ad abbandonare gli approcci prettamente meccanico/sensoriale. Infatti, nel 1988, Tamura *et al.* [42] proposero il primo riconoscimento dei segni basato interamente sulla visione. Il sistema costruito era in grado di 10 segni giapponesi basandosi sulle soglie dei colori.

Con il tempo l'uso delle telecamere è diventato sempre più dominante, infatti, molti studiosi hanno investigato approcci in grado di utilizzare informazioni sulla profondità [23, 44, 47] o impiegando direttamente più telecamere per il riconoscimento [4]. L'utilizzo di più telecamere si è mostrato essere troppo vincolante dimostrarsi essere un per applicazioni reali. Proprio per questo lo studio del riconoscimento si è indirizzato verso l'utilizzo di una sola telecamera RGB, al fine di consentire una più pratica soluzione.

Il grande passo, però, è stato fatto grazie al progresso dell'apprendimento di reti profonde e l'avvento delle convolutional neural networks (ConvNet), in ambito del trattamento delle immagini. In particolare, negli ultimi tempi,

l'uso delle ConvNet è diventato sempre più preponderante, invogliando a proporre sempre più varianti. Infatti, sono molteplici sia i lavori che utilizzano le ConvNets3D, sia quelli che mescolano l'utilizzo delle ConvNet2D con le reti neurali ricorrenti. In queste tipologie di approccio, sicuramente, risalta il lavoro effettuato da Razieh Rastgoo *et al.* [34] e quello svolto da Jie Huang *et al.* [19].

L'approccio proposto in [34] prevede l'implementazione di un modello a cascata per il riconoscimento del linguaggio dei segni che sfrutta le informazioni spazio-temporali estratte esclusivamente dalle mani. Gli autori integrano nel sistema tre approcci di deep learning: il Single Shot Detector (SSD) [26], una Convolutional Neural Network (ConvNet) [24] ed una Long Short-Term Memory (LSTM) [18]. In particolare, il modello proposto comprende due parti principali: rilevamento delle mani nello streaming video e il riconoscimento del segno. Si inizia con la rilevazione e l'estrazione delle mani, attraverso l'SSD, dai frame che compongono la sequenza video. Si procede poi con l'estrazione delle feature visuali attraverso una ConvNet, l'extra spatial hand relation (cioè l'individuazione della pendenza e dell'orientamento delle mani) e l'hand pose. Infine, le feature visuali vengono fornire alla LSTM per l'estrazione delle feature temporali al fine di individuare il segno effettuato. Il modello è stato valutato su un dataset proposto dagli stessi autori di [34], includendo 10.000 video di 100 segni persiani, segnati da 10 collaboratori su 10 sfondi diversi.

Sebbene il lavoro presentato da Rastgoo *et al.* sia eccellente, in quanto l'accuracy raggiunta sul test set è pari all'86,32%, si possono notare delle evidenti differenze con l'approccio proposto in questo lavoro di tesi:

1. Lingua dei segni: nel paper [34] si focalizza l'attenzione sulla lingua persiana, mentre nel lavoro proposto si lavora con la lingua americana, ovvero, quella più utilizzata nel mondo;
2. Approccio: l'approccio proposto in [34] prevede vari step e una rete molto più complessa da allenare rispetto a quella presentata in questa tesi, rendendola non particolarmente adatta in un riconoscimento real-time.

L'approccio proposto in [19], si basa su una particolare rete chiamata ConvNet3D. Questa rete non è altro che una generalizzazione di una ConvNet, in quanto, invece di sfruttare 2 dimensionalità per i vari layer ne utilizza 3. Questo fa sì che la rete sia in grado di apprendere contemporaneamente sia feature spaziali che temporali, infatti, la terza dimensione può essere vista come il tempo.

Le reti convoluzionali 3D sono in grado, per natura, di gestire sequenze di frame senza richiedere il supporto delle reti neurali ricorrenti, a discapito però di un abbassamento di prestazioni dovuto alla maggior complessità strutturale. La cosa più importante, però, è il tipo di input che gli autori di [19] danno in pasto alla rete progettata. Essa, infatti, pende in input non solo la sequenza di frame, ma anche le depth images (immagini che mettono in risalto il corpo, segmentandolo) associate ai frame e il body skeleton della persona che effettua il segno. Tutte queste informazioni spaziali danno alla rete la possibilità di associare, in maniera migliore, il gesto con il significato ad esso collegato.

Sebbene la rete utilizzata dagli autori di [19] è molto promettente bisogna fare alcune considerazioni:

1. la rete richiede un numero di ore molto elevato per il training;
2. inoltre, data la natura intrinseca di una ConvNet3D, essa tende a non comportarsi bene in task real-time se non utilizzando un approccio parallelizzabile.

Per i motivi sopra indicati si è deciso, per il modello proposto in questo studio di tesi, di abbandonare le ConvNet3D e di sfruttare un approccio simile a quello descritto in [34]. C'è però da fare una considerazione: sebbene i risultati ottenuti in [34] siano molto convincenti, si è proceduto con un approccio più snello in modo da poterlo utilizzare in maniera efficiente in un'applicazione real-time. Anche la struttura proposta in [19] risulta essere troppo complessa per poter essere utilizzata in ambito real-time.

Capitolo 3

Dataset

Qualsiasi sia il tipo di problema che si vuole risolvere con un approccio basato sul deep learning, si deve creare o avere a disposizione un dataset di grandi dimensioni o quanto meno adeguato al task che si vuole risolvere. Tutto ciò vale, naturalmente, anche per il problema del riconoscimento della lingua dei segni.

3.1 Scelta del Dataset

Ad oggi, esistono nella comunità scientifica diversi lavori di creazione di dataset per il riconoscimento della lingua dei segni americana a livello di parola. Tra questi si può sicuramente mettere a confronto il Purdue RVL-SLLL database [27], il RWTH-BOSTON-50 dataset [48], il Boston ASLLVD dataset [2], il MS-ASL dataset [20] e il WLDSL dataset [25].

	#Gloss	#Videos	#Signers	Language
Purdue RVL-SLLL	39	546	14	American
RWTH-BOSTON-50	50	483	3	American
Boston ASLLVD	2.742	9.794	6	American
MS-ASL	1.000	25.513	222	American
WLDSL	2.000	21.083	119	American

Tabella 3.1: Datasets a confronto.

La Tabella 3.1 riassume bene le differenze che sussistono tra tali datasets. Si può subito intuire come i primi tre siano stati scartati a prescindere. Infatti, il Purdue RVL-SLLL e il RWTH-BOSTON-50 presentano una quantità di samples molto bassa, il che li rende non consoni per applicazioni di deep learning. Mentre, il Boston ASLLVD pur avendo una quantità discreta di video a disposizione, risultano comunque molto più pochi rispetto ai samples degli ultimi due dataset. Dunque, per la scelta finale concorrono i dataset MS-ASL e WLALS.

La scelta non è facile in quanto entrambi i dataset sono molto validi ed hanno alle spalle uno studio molto recente. Il dataset MS-ASL può essere definito più bilanciato rispetto al WLALS in quanto presenta un numero maggiore di samples ripartiti in un numero di glosses minore. Inoltre, il numero di signers maggiore può aiutare nella generalizzazione.

Nonostante tutto, la scelta è ricaduta sul dataset WLALS. Questo perché, presentando un numero maggiore di gloss può essere considerato più vicino a un vocabolario reale di uso quotidiano. Inoltre, sia il numero di samples che il numero di signers rimane molto elevato per un dataset di questo genere, confermandosi una scelta del tutto valida. A rafforzare ulteriormente tale decisione è stato l'ottenimento del dataset già pre-processato da parte degli autori del paper; cosa che non è avvenuta per il dataset MS-ASL.

Si passa ora ad approfondire le caratteristiche del dataset WLALS e del tipo di pre-processing effettuato.

3.2 Word-Level American Sign Language

Il Word-Level American Sign Language dataset [25], abbreviato con WLALS, è un dataset video creato appositamente per far fronte alla complessa problematica del riconoscimento della lingua dei segni americana. Dal nome del dataset si ci rende subito conto che questo tipo di dataset è pensato appositamente per focalizzare l'attenzione sul riconoscimento della lingua dei segni basandosi sulle singole parole.

Il processo di creazione del dataset può essere diviso in due fasi:

1. processo di raccolta dei video;
2. processo di annotazione dei video.

3.2.1 Processo di Raccolta dei Video

Li *et al.* hanno individuato 68.129 video provenienti da 20 fonti diverse [25], tra le quali, ad esempio, siti di educazione alla lingua dei segni, YouTube, etc. Una volta collezionati tutti i video dalle varie fonti è avvenuta un prima selezione:

- sono stati rimossi i video in cui erano presenti due o più gloss;
- sono state rimosse le gloss che avevano un numero minimo di video inferiore a 7.

Dopo questa prima fase di selezione sono stati ottenuti 34.404 video e 3.126 glosses. Inoltre, in tutti i video si ha un solo signer che effettua una sola gloss (parola), mantenendo una posizione frontale rispetto all'inquadratura. Tale fattore è determinante per avere un dataset di alta qualità, in quanto, nella maggior parte delle situazioni, le persone comunicano posizionandosi una di fronte all'altra.

3.2.2 Processo di Annotazione dei Video

La parte più onerosa della creazione del dataset è ricaduta nella fase di annotazione. Infatti, in questo caso gli autori non si sono limitati ad apporre per ogni video la rispettiva gloss segnata, ma hanno anche annotato ogni video con altre meta informazioni:

- *confine temporale*: vengono indicati in modo preciso i frame di inizio e fine in cui compare il segno;
- *bounding box della persona*: con l'algoritmo YOLOv3 [35] è stato trovato il bounding box della persona. Si noti che il metadato riportato si riferisce al bounding box più grande, in quanto durante la segnatura tale box può allargarsi e restringersi in base ai momenti della persona;

- *ID del signer*: grazie all'utilizzo del face detector e del face embedding di FaceNet [39], gli autori sono riusciti ad attribuire un ID ad ogni signer;
- *annotazione dialettale*: questa fase ha comportato l'annotazione di tutti i video in modo da capire quali di essi fosse una variazione dialettale di altri. Per fare ciò gli autori si sono serviti di una loro interfaccia grafica e di annotatori che sono stati precedentemente formati per tale lavoro.

Inoltre, dopo aver proceduto con le annotazioni dialettali, sono state eliminate dal dataset tutte le variazioni di segno che presentassero meno di cinque video.

3.2.3 Caratteristiche del Dataset

Tutti i metadati raccolti non sono presenti solo a titolo informativo, ma sono stati sfruttati per far in modo che la suddivisione in train, validation e test set avvenisse in maniera del tutto omogenea. Ad esempio, gli autori hanno effettuato la ripartizione in modo tale che le variazioni dialettali non comparissero solo in uno dei sub-sets; così come anche per i signers.

Alla fine, dopo aver effettuato le due fasi di pre-processing e tenendo conto di tutte le informazioni recuperate, si è arrivati al dataset finale: il WLASL, composto da 21.083 video ripartiti su 2.000 glosses.

Tutte le informazioni del dataset sono state raccolte dagli autori in un file JSON, il quale elenca le gloss in ordine decrescente in base al numero di samples. Grazie a tale ordinamento, il dataset può essere suddiviso facilmente in 4 sotto-dataset. Questa suddivisione è particolarmente importante in quanto fa notare come la complessità del riconoscimento della lingua dei segni aumenti drasticamente all'aumentare del numero di parole da gestire. Si racchiudono, nella Tabella 3.2, le caratteristiche del dataset utilizzato.

	#Gloss	#Videos	#Signers	Samples	Mean
WLASL100	100	2.038	97	20,4	
WLASL300	300	5.117	109	17,1	
WLASL1000	1.000	13.168	116	13,2	
WLASL2000	2.000	21.083	119	10,5	

Tabella 3.2: WLASL dataset.

A conclusione del capitolo, si propone un’immagine raffigurante alcuni frames estratti dai video che compongono il dataset (Figura 3.1).



Figura 3.1: Frames estratti dal dataset WLASL [25].

Capitolo 4

Background

Al fine di ottenere una lettura più scorrevole da qui in avanti, è opportuno far acquisire al lettore il background necessario. In particolare si ci soffermerà sugli elementi chiave del modello prodotto: le reti convoluzionali e le reti ricorrenti. Il modello verrà poi approfondito nella sua parte più tecnica nei capitoli successivi.

4.1 Convolutional Neural Networks

Le reti neurali convoluzionali, o meglio conosciute come Convolutional Neural Networks (CNNs oppure ConvNets), sono state introdotte per la prima volta da LeCun et al. [24]. Nate dallo studio dallo studio della corteccia visiva del cervello, sono state utilizzate nel riconoscimento delle immagini già dagli anni '80.

Ciò che rende le ConvNets così adatte al trattamento delle immagini è dovuto proprio alla loro somiglianza nel funzionamento con i neuroni della corteccia visiva. Gli studiosi, infatti, si sono accorti che tali neuroni si focalizzano solo un piccolo campo ricettivo locale in cui cercare dei pattern (linee, cerchi, etc.). Questi campi ricettivi possono anche sovrapporsi ed insieme scansionare l'intero campo visivo.

Inoltre, alcuni neuroni hanno la facoltà di inglobare nel loro campo ricettivo quello di altri neuroni, riuscendo, così, ad apprendere pattern più

complessi. Attraverso ciò, i neuroni sono in grado di rilevare, nell'intero campo visivo, qualsiasi tipo di pattern e dettaglio.

Al fine di poter tradurre la struttura dei neuroni della corteccia visiva a livello artificiale è stato necessario introdurre due layer particolari: il convolutional layer e il pooling layer.

4.1.1 Convolutional Layer

Il più importante elemento di una ConvNet è il convolutional layer. Al fine di riprodurre il campo ricettivo presente in natura, i neuroni in un convolutional layer sono connessi solo parzialmente ai neuroni presenti nel layer precedente. Allo stesso modo, i pixel dell'immagine data in input sono connessi solo a determinati neuroni. Tutto questo può essere ben apprezzato nella Figura 4.1.

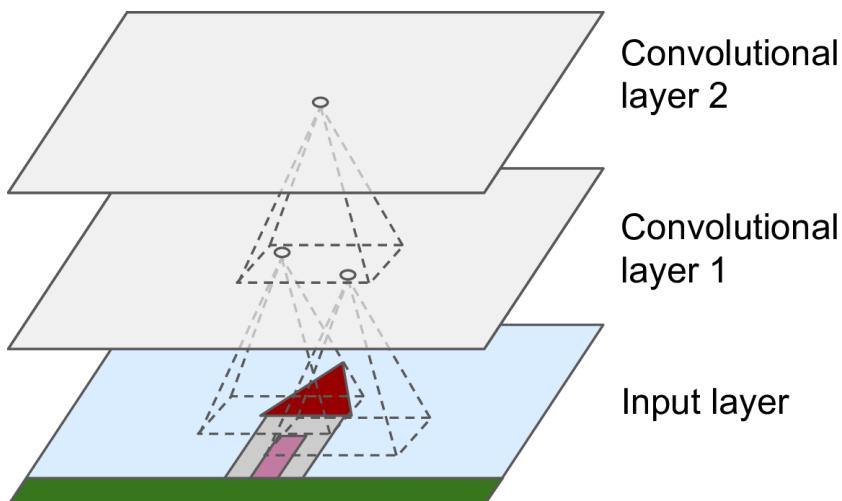


Figura 4.1: Funzionamento dei livelli di convoluzione^[12].

Questa architettura consente alla rete di concentrarsi dapprima su semplici features, fino ad arrivare a features sempre più complesse negli strati più alti. Ed è proprio per questo che le ConvNets si prestano bene nel riconoscimento delle immagini, o meglio di ciò che si trova all'interno di tali.

Ma come può il convolutional layer ‘capire’ cosa c’è nell’immagine? Per rispondere a tale domanda si deve introdurre il concetto di *convolutional*

kernel o, alternativamente, di filtro. Un kernel non è altro che una matrice di pesi, grande quanto il campo recettivo di un neurone, che viene fatta scorrere sull'intera immagine. Tale operazione di scorimento viene chiamata in matematica convoluzione ed è da questo che il layer prende il nome.

Per capire bene il concetto basta guardare la Figura 4.2. L'immagine di destra è stata ottenuta applicando il filtro all'immagine di sinistra. In questo caso, il kernel è formato da una matrice 3×3 in cui sulla linea verticale centrale ha tutti 1 e nelle restanti posizioni 0. Questo fa sì che il convolutional layer, o meglio i suoi neuroni, apprendano (si focalizzino) features di tipo verticale (linee). Infatti, come risultato della convoluzione, non si ottiene altro che il risalto delle linee verticali dell'immagine di partenza.

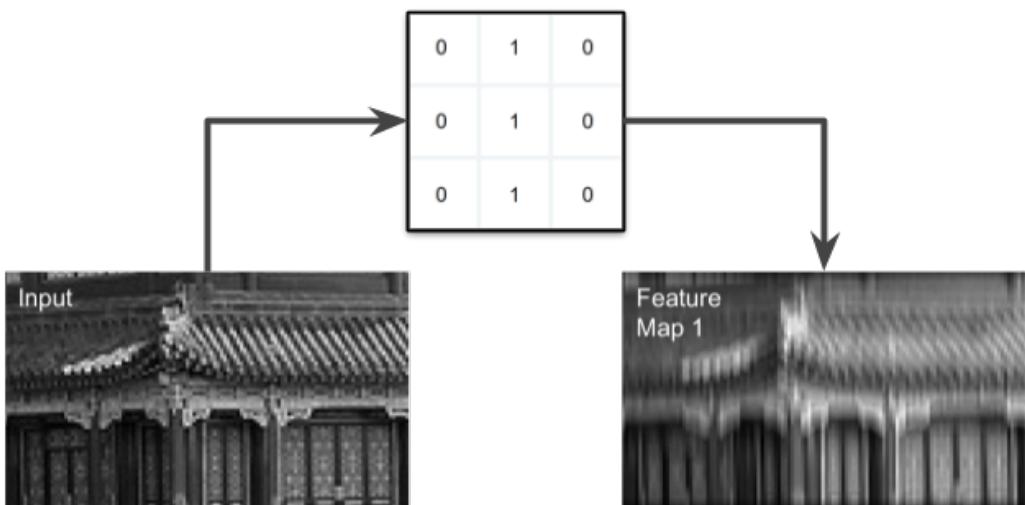


Figura 4.2: Applicazione del kernel verticale in una convoluzione^[12].

In realtà, i convolutional layer non sono formati da un unico strato, ma si possono immaginare come un insieme di strati, dove tutti i neuroni di uno strato apprendono un unico filtro. Grazie alla sovrapposizione di più strati, la rete è in grado di apprendere ed applicare più filtri contemporaneamente; al fine di rilevare features spaziali multiple in una sola convoluzione. Anche in questo caso la Figura 4.3 può aiutare a schiarire le idee.

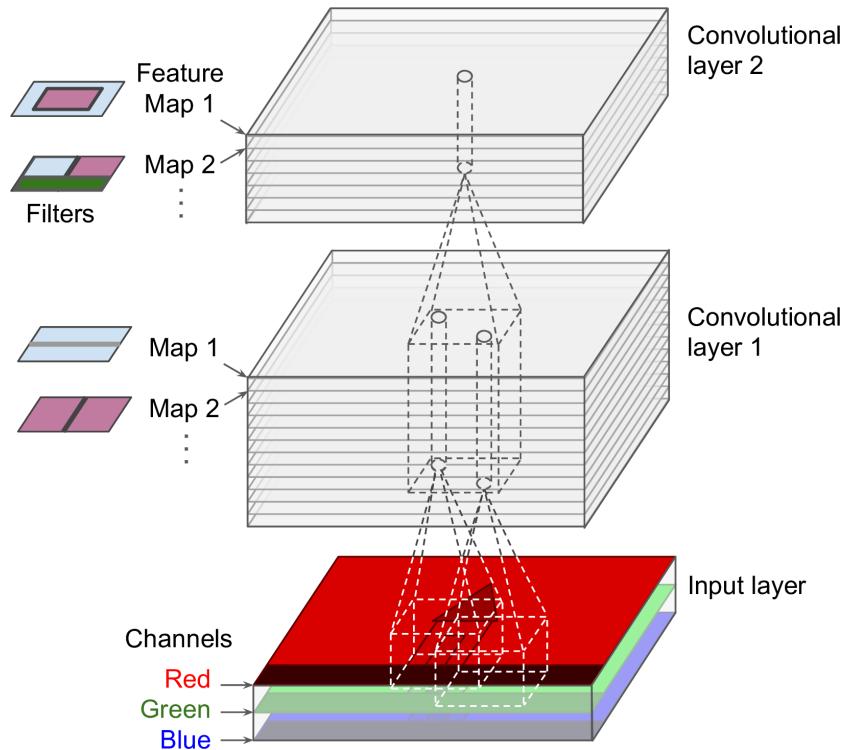


Figura 4.3: Struttura reale dei convolutional layer^[12].

La cosa interessante è che non si devono indicare i tipi di kernel da far apprendere alla rete, ma è qualcosa che lei effettua in automatico.

Infine, nella Figura 4.4, viene riportato un esempio di features spaziali apprese durante l'applicazione di molteplici layer di convoluzione.

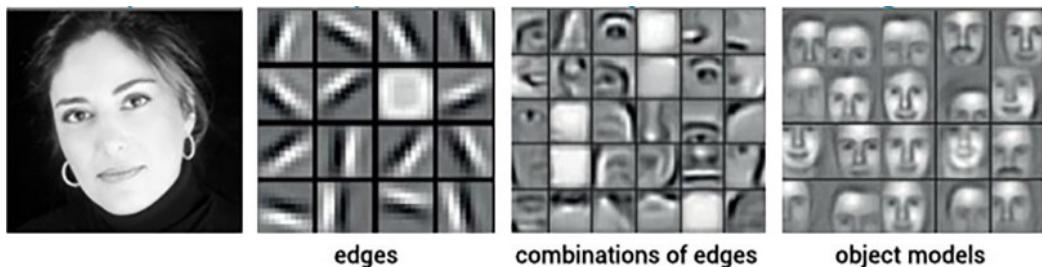


Figura 4.4: Features spaziali apprese durante il training di una ConvNet^[30].

Come accennato prima, i pattern appresi dai diversi layer di convoluzione diventano mano mano sempre più complessi, rendendo tali reti capaci di apprendere qualsiasi tipo di conoscenza spaziale.

4.1.2 Pooling Layer

I livelli di raggruppamento, chiamati anche pooling layers, hanno un funzionamento molto simile a quelli convoluzionali. In questo caso però, il loro obiettivo è quello di sottocampionare l'immagine proveniente da uno strato di convoluzione, come mostrato in Figura 4.5.

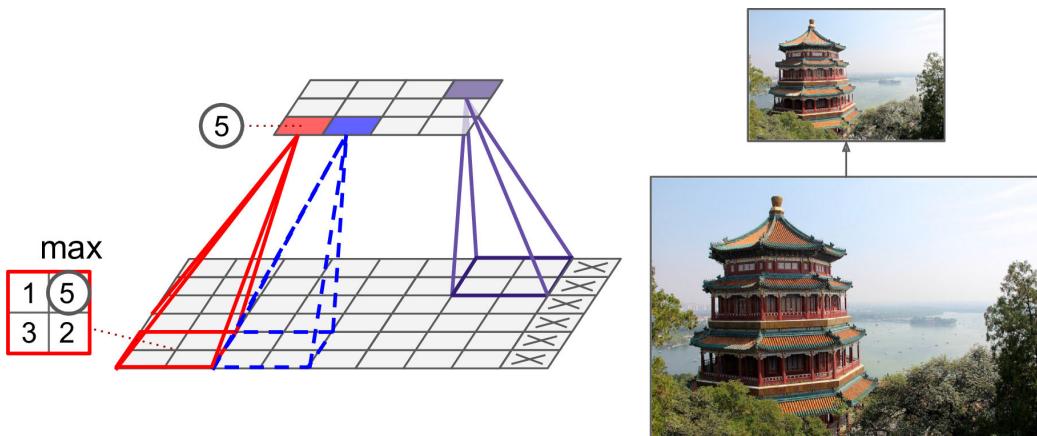


Figura 4.5: Sottocampionamento attraverso un pooling layer^[12].

Come si può notare dalla Figura 4.5, proprio come negli strati convoluzionali, ogni neurone in un pooling layer è collegato a un numero limitato di neuroni nel layer precedente. Tuttavia, un neurone di pooling non ha pesi, perché, tutto ciò che deve fare è applicare ai valori del campo ricettivo una funzione di aggregazione, come il massimo o la media.

Effettuando l'aggregazione, il pooling layer assolve ad un duplice compito:
 1. ridurre il carico computazionale, l'utilizzo della memoria e il numero di parametri; 2. rimpicciolire l'immagine per focalizzarsi su features spaziali di più alto livello.

4.1.3 Architettura tipica di una ConvNet

Capito quali sono i compiti del convolutional e del pooling layer, si può passare ad analizzare la strutturata di una ConvNet.

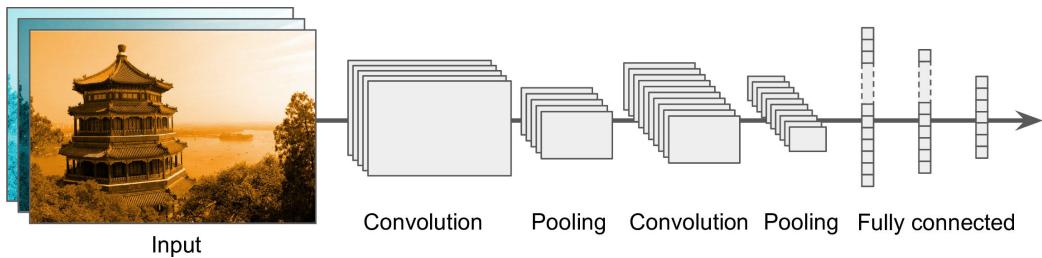


Figura 4.6: Architettura tipica di una ConvNet^[12].

Dalla Figura 4.6 si può osservare una classica architettura di una ConvNet.

Normalmente vengono impilati uno o più layer convoluzionali, seguiti da un pooling layer, quindi altri layer convoluzionali, seguiti da un altro pooling layer e così via.

Con l'utilizzo dei pooling layer l'immagine diventa sempre più piccola, ma utilizzando sempre più filtri, essa diventa anche più profonda; permettendo così alla rete di individuare features molto più complesse.

Infine, nella parte superiore dello stack, viene aggiunta una normale rete neurale feedforward, utilizzata per adempiere al task da svolgere: classificazione, regressione, etc.

4.1.4 Data Augmentation

Molto spesso, quando si ha a che fare con problematiche incentrate prevalentemente sulla computer vision, il dataset a disposizione potrebbe non essere troppo corposo. Al fine di aumentare il numero di samples in maniera artificiale è stata creata una tecnica chiamata data augmentation.

La data augmentation non fa altro che generare a run-time, dall'immagine originale, una serie di varianti realistiche. Queste varianti sono ottenute attraverso operazioni di zoom, scaling, rotation, shifting, etc. Alcune di queste operazioni sono mostrate nella Figura 4.7.

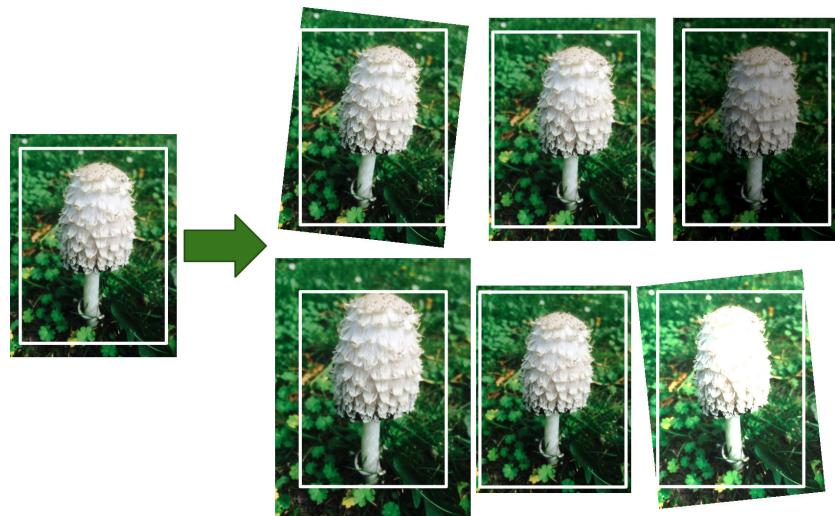


Figura 4.7: Applicazione della data augmentation all’immagine di sinistra^[11].

Lo scopo principale di questo approccio è quello, come detto in precedenza, di aumentare i dati di training; tuttavia esso ha anche un secondo fine. Creando queste varianti non si fa altro che indurre il modello ad essere più tollerante alle variazioni di posizione, orientamento e dimensione degli oggetti nelle immagini. Questa tolleranza si traduce nell’avere un modello più robusto, meno incline all’overfitting e soprattutto in grado di generalizzare meglio.

Si può, dunque, sfruttare la data augmentation sia per aumentare i dati, sia per regolarizzare il modello.

4.1.5 Transfer Learning

In ambito delle ConvNet è molto conosciuta una tecnica chiamata transfer learning [49], letteralmente trasferimento dell’apprendimento.

Questa tecnica nasce dall’esigenza di evitare di dover allenare profonde reti neurali convoluzionali e con una quantità enorme di dati.

Infatti, per allenare tali reti bisogna avere molto spazio di archiviazione per i modelli e una potenza di calcolo non indifferente. Inoltre, questo discorso viene accentuato quando si parla di dataset di immagini e video.

Per tali motivi, nella maggior parte dei task, si ricorre all'uso del transfer learning.

Il transfer learning è un metodo di machine learning in cui un modello, sviluppato e addestrato per una specifica attività, viene riutilizzato come punto di partenza per far fronte ad un diverso task. Tale approccio permette, dunque, di usare i modelli pre-addestrati come punto di partenza su cui continuare a costruire una rete custom.

In altri termini, il transfer learning permette il riuso delle conoscenze acquisite durante la risoluzione di un problema per velocizzare l'apprendimento della rete su un problema diverso, ma correlato. Ad esempio, la conoscenza acquisita per imparare a riconoscere i volti, potrebbe essere riutilizzata quando si cerca di riconoscere la presenza di una mascherina sui volti stessi.

Nel tempo sono state presentate ed allenate varie tipologie di ConvNets, tra cui VGG16 [40], VGG19 [40], ResNet [17], InceptionV3 [41], etc.

Tutte queste reti sono state e sono tutt'ora impiegate per il transfer learning, utilizzando, come pesi, quelli ottenuti allenando la rete sul dataset ImageNet [8] o simili.

Per il lavoro di tesi proposto si analizzerà la rete MobileNetV2 [37]. Questa rete presenta una serie di vantaggi:

- il numero di parametri da gestire è abbastanza ridotto rispetto alle altre reti utilizzate per il transfer learning;
- è una rete neurale a bassa latenza, ottima dunque per applicazioni real-time;
- date le due caratteristiche precedenti, essa si presta bene per dispositivi embedded e mobile.

Nonostante MobileNetV2 abbia una minor precisione a causa del ridotto numero di parametri, essa è ampiamente utilizzata nella pratica per via delle sue prestazioni. Ulteriori dettagli verranno discussi nei prossimi capitoli.

4.2 Recurrent Neural Networks

Sebbene per le persone sia facile comprendere qualcosa in base alle precedenti esperienze, ciò non è così scontato in ambito delle reti neurali artificiali (ANNs o NNs). Ad esempio, effettuare la previsione dell'andamento di un'azione di mercato, identificare l'azione effettuata all'interno di un video o completare una frase, sono tutte attività che hanno bisogno di tener conto di informazioni che si susseguono nel tempo. Tutto questo per una normale rete neurale non è possibile.

Per gestire tali mancanze delle NNs, sono nate le Recurrent Neural Networks (RNNs) [36]. Le RNN risolvono questo problema utilizzando, in maniera ciclica, il proprio output insieme all'input, come mostrato a sinistra nella Figura 4.8. Così facendo tali reti riescono a prendere decisioni in base a ciò che è accaduto precedentemente, come se si avesse memoria.

Infatti, è proprio il ciclo che consente il passaggio delle informazioni da un stato della rete a quello successivo. Per avere un'idea completa di come questo ciclo funzioni basta pensare la RNN in maniera srotolata: immagine a destra della Figura 4.8.

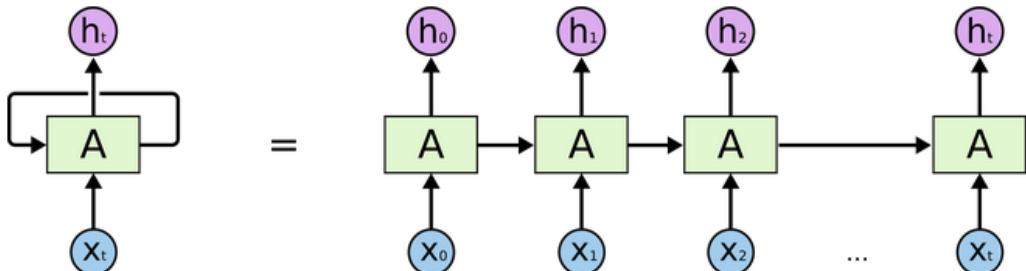


Figura 4.8: Architettura di una Rete Neurale Ricorrente^[31].

Ad oggi le RNN sono state utilizzate per molteplici task, come lo speech recognition, il language modelling, la traduzione, l'image captioning, etc. Tuttavia esse non sono perfette, infatti, soffrono del problema delle dipendenze a lungo termine (the problem of long-term dependencies).

Il problema delle dipendenze a lungo termine

Per capire bene il problema si deve appurare di quanto indietro le RNN possono tenere memoria.

Basandoci sulla struttura vista nella Figura 4.8 si può facilmente intuire come le RNN si comportino bene quando l'informazione da tener presente, prima di ‘dimenticarla’, è a pochi passi indietro nel tempo.

Ma cosa succede se ciò che si deve predire dipende da informazioni molto più nel passato? Semplicemente, con il trascorrere del tempo, le RNN diventano incapaci di collegare le informazioni.

Infatti, anche se in teoria, le RNN sono assolutamente in grado di gestire tali ‘dipendenze temporali a lungo termine’, in pratica, essere non sono capaci di farlo [3].

Con l'avvento delle Long Short-Term Memory networks si è ovviato, però, a tale problema.

4.2.1 Long Short-Term Memory

Le reti neurali Long Short-Term Memory, di solito chiamate LSTMs, sono state introdotte per la prima volta da Hochreiter & Schmidhuber nel 1997 [18].

Grazie alla loro abilità di apprendere dipendenze a lungo termine, sono divenute subito molto popolari. Infatti, ricordare le informazioni per lunghi periodi di tempo è praticamente il loro comportamento predefinito, non qualcosa che hanno bisogno di imparare. Per tale motivo si prestano bene a tutti quei problemi che le normali RNN non possono risolvere in pratica.

La struttura delle LSTM, in maniera molto generale, assomiglia a quelle delle normali RNN, Figura 4.9.

Come si nota anche le LSTM possono essere rappresentate da una sequenza a catena, ma il modulo ripetuto ha una struttura diversa e molto più complessa. Infatti, in questo caso sono presenti quattro livelli di rete neurale che interagiscono in un modo molto speciale.

Per poter capire bene come una LSTM possa compiere il suo lavoro, bisogna dapprima individuare le componendi che entrano in gioco:

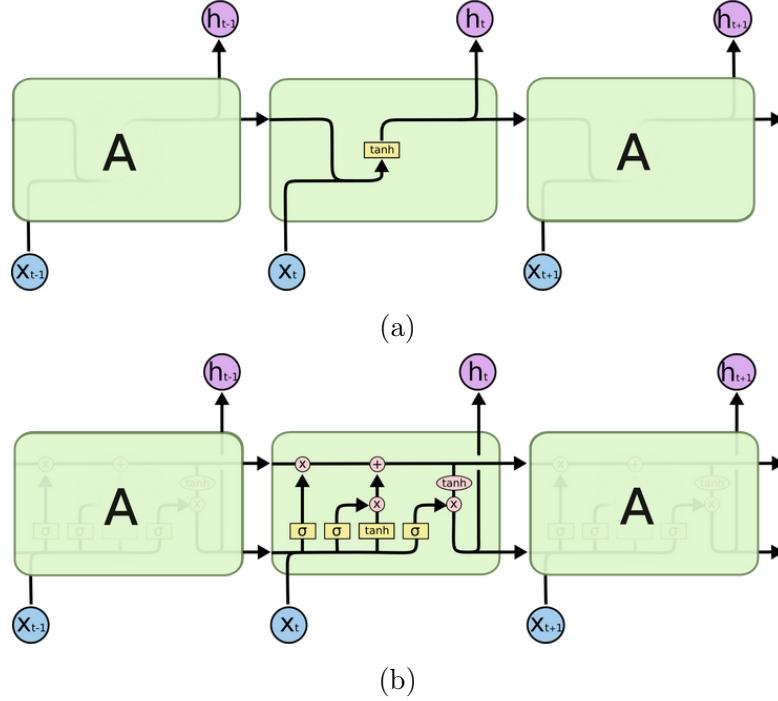


Figura 4.9: (a) Struttura di una RNN. (b) Struttura di una LSTM^[31].

- X_t : input al tempo t . Vettore numerico rappresentante l'informazione in input;
- h_t : output al tempo t . Vettore di output della stessa dimensione dell'input;
- riquadri gialli: rappresentano le quattro reti neurali con le rispettive funzioni di attivazione;
- cerchi rosa: rappresentano le operazioni point-wise;
- linee: rappresentano il percorso seguito dai vettori rappresentanti l'input e l'output. Quando una linea si divide avviene una copia del vettore, quando due linee si incontrano si ha una concatenazione dei vettori.

La parte più importante di una cella LSTM è la linea orizzontale che scorre in alto. Tale linea può essere vista come un ‘nastro trasportatore’ che

fa viaggiare l'hidden state della LSTM da una cella ad un'altra. Grazie a ciò è molto facile far fluire le informazioni e ricordale.

Al fine di continuare a ricordare informazioni, aggiungerne di nuove o di rimuoverne altre, la LSTM sfrutta i così detti gates. In particolare, una LSTM utilizza tre gates per controllare lo stato della cella.

Sfruttando tali gates, l'LSTM riesce, in maniera del tutto arbitraria, a capire quale informazione tenere e quale eliminare.

I Gates

I gates sono moduli formati da operazioni point-wise e le reti neurali accennate in precedenza. Le reti neurali, infatti, grazie alle loro funzioni di attivazione sigmoidee possono decidere la quantità di informazione da far passare: andando da 0 (nessuna informazione) a 1 (tutta l'informazione). Vediamo da vicino quali sono le particolarità di ciascun gate.

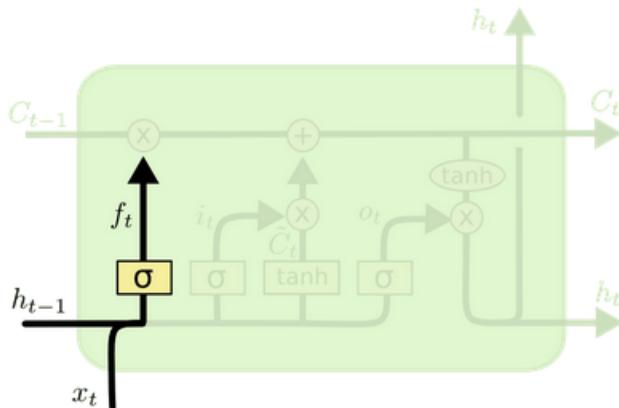
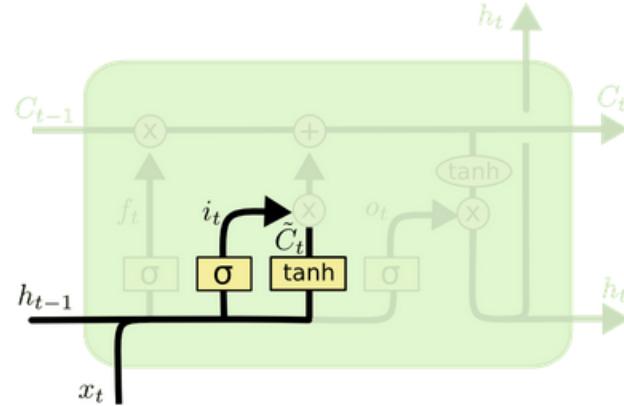


Figura 4.10: Gate 1 - Forget gate^[31].

Il primo, mostrato in Figura 4.10, si occupa di decidere quali informazioni eliminare dallo stato della cella. Questa decisione è presa dal layer sigmoideo chiamato per l'appunto “forget gate layer”.

Il forget layer prende in input h_{t-1} e x_t e restituisce, per ogni valore del vettore concatenato, un numero compreso tra 0 e 1. Tale valore verrà poi impiegato in un'operazione point-wise con C_{t-1} , ovvero, il vettore rappresentante lo stato nascosto della cella.

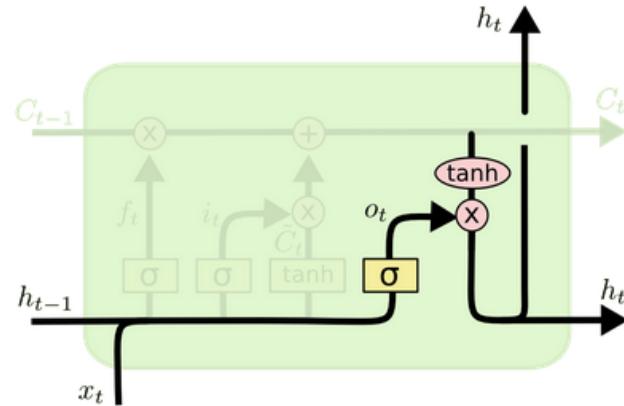
Figura 4.11: Gate 2 - Input gate^[31].

Il secondo gate, mostrato in Figura 4.11 si occupa di decidere quali nuove informazioni dovranno essere memorizzate nello stato della cella.

Questo gate è più complesso del primo, infatti, esso presenta due fasi ben distinte:

- I. il sigmoid layer, chiamato “input gate layer”, decide quali valori aggiornare;
- II. il tanh layer genera un vettore di nuovi valori candidati, che potenzialmente, potrebbero essere aggiunti allo stato;

Superati questi due gate iniziali, lo stato interno della cella LSTM risulterà aggiornato attraverso le varie operazioni point-wise.

Figura 4.12: Gate 3 - Output gate^[31].

Aggiornato lo stato interno, esso viene utilizzato insieme al terzo gate (Figura 4.12) per produrre l'output dell'intera cella. Infatti, tale output è dato dall'applicazione di un'operazione point-wise tra lo stato interno, filtrato con un layer tanh, e il vettore proveniente dall'ultimo gate.

Puntualizzando, Il layer sigmoideo decide quali parti dell'input devono essere conservate, mentre, l'operazione point-wise aiuta a mantenere le nuove informazioni che nei passi precedenti sono state aggiunte.

4.2.2 Gated Recurrent Unit

Sebbene le LSTM riescano a risolvere una miriade di problemi, nel corso degli anni sono state comunque presentate alcune varianti. La Gated Recurrent Unit, introdotta da Cho et al. [6], è una di esse.

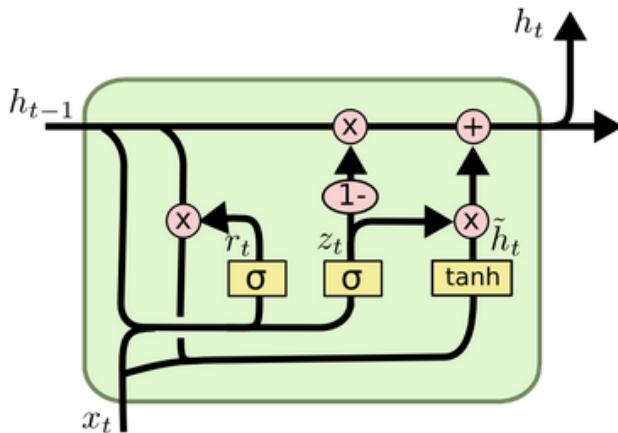


Figura 4.13: Gated Recurrent Unit^[31].

Il funzionamento, mostrato nella Figura 4.13, è molto simile ad una LSTM, ma presenta alcune importanti differenze.

Prima di tutto, la cella GRU non presenta un hidden state, ma è la cella stessa che assume la funzionalità di stato nascosto. Dunque, l'intero stato della cella viene fornito come output.

In secondo luogo, una cella GRU presenta solo due gate: il *reset gate* e l'*update gate*. Il reset gate viene utilizzato dal modello per decidere la quantità dell'informazione passata che è necessario trascurare. L'output gate,

invece, è responsabile della determinazione della quantità di informazione, presente nelle fasi temporali precedenti, che deve essere trasmessa nello stato successivo.

Questi due gate, dunque, decidono quali informazioni dello stato nascosto devono essere aggiornate o resettate, permettendo alla rete di capire quali di esse utilizzare.

La differenza sostanziale, però, sta nel fatto che la GRU non presenta l'*output gate* e combina l'*input gate* e il *forget gate* in unico gate: l'*update gate*. Tutto questo si traduce nell'inserire nuove informazioni nello stato solo quando si dimentica qualcosa di più vecchio.

Queste variazioni della GRU, rispetto alla LSTM, confluiscono in un design più semplice, efficiente e meno incline all'utilizzo di memoria. Infatti, gestendo meno parametri, una GRU risulta molto più facile da allenare, a discapito, però, di una minore accuratezza.

Ma come decidere se utilizzare una GRU o una LSTM? Se la sequenza temporale da tener conto è molto lunga, le LSTM sono l'ideale, in quanto esse utilizzano più parametri per gestire le informazioni. Mentre, se la sequenza non è troppo lunga o semplicemente si cerca un modello più efficiente, è bene optare per una GRU.

Dato che per il modello proposto, il training verrà effettuato con una sequenza di frame non molto lunga, si predilige la scelta delle GRU per ottenere un gain in termini di consumo di memoria ed efficienza.

Capitolo 5

Modello Proposto

Il riconoscimento della lingua dei segni non è un compito semplice, anzi per poterlo affrontare bisogna interpellare vari rami del deep learning, o se si vuole, dell'intelligenza artificiale in generale.

Pensando ad alto livello, quello che si vuole ottenere è un modello che sia in grado di gestire video, o meglio sequenze di frame, feature spaziali e temporali. Le feature spaziali servono per riconoscere gli oggetti, in questo caso la persona con le varie parti del corpo, mentre le feature temporali servono per interlacciare tali feature spaziali, relazionandole nel tempo. Infatti, nel caso specifico da trattare si dovrà associare a sequenze di movimenti una label, ovvero, la *gloss* del dizionario usato.

Dunque, il modello, per essere in grado di svolgere il suo lavoro dovrà essere composto dalla seguenti quattro parti:

1. **Frame Sampling**: modulo custom in grado di estrarre i frame da un video;
2. **ConvNet**: convolutional neural network utilizzata per estrarre le feature spaziali dai frames;
3. **GRU**: recurrent neural network utilizzata per estrarre le feature temporali che intercorrono nella sequenza di frame;
4. **Feedforward**: fully-connected neural network utilizzata per effettuare la classificazione.

Verranno ora analizzare in dettaglio le varie parti in modo da comprenderne meglio il funzionamento del modello proposto.

5.1 Fase 1: Frame Sampling

Per questa prima fase è necessario disporre di un generatore di frame in grado di estrapolare i vari frame da un video secondo una determinata politica di campionamento.

Per prima cosa si è cercato di trovare qualche modulo già sviluppato per far fronte a tale necessità. Da una attenta ricerca si è giunti ad un modulo presente tra le librerie di python.

Sebbene il modulo *keras-video-generators* [28] sia già pronto all'uso, bisogna fare alcune importanti considerazioni:

- il modulo non permette una politica di campionamento custom;
- a livello di implementazione il modulo è concepito per lavorare solo su un'organizzazione ben precisa dei video e delle cartelle, rendendo il tutto molto più complicato da gestire.

Per tali ragioni si è proceduto con la progettazione di un frame generator custom, in modo da adattarsi in tutto e per tutto alle esigenze del modello proposto.

5.1.1 Custom Frame Generator

Al di là dell'implementazione vera e propria, quello che più interessa di un custom generator è sicuramente la politica di campionamento; ed è su questo che si concentrerà l'analisi.

Alla fine di avere un buon campionamento, il fattore di omogeneità è sicuramente predominante. Infatti, ottenere un campionamento che non rispecchi l'intera sequenza dei frame, cioè il video, comporterebbe una cattiva rappresentazione del video stesso.

Dopo diverse fasi di sperimentazione si è giunti alla politica del "doppio campionamento". Come suggerisce il nome, tale politica presenta due

fasi ben distinte: il pre-campionamento e il campionamento vero e proprio. Queste due fasi possono essere sintetizzate come di seguito:

- Prima fase - **pre-campionamento**: Il primo campionamento divide il totale dei frame del video in 12 chunks mantenendoli il più bilanciati possibile. Ottenuti i chunks vengono scartati il primo e l'ultimo. In questo modo si cerca di eliminare gli istanti iniziali e finali in cui il signer non esegue nessun gesto;
- Seconda fase - **campionamento**: Il secondo campionamento divide i restanti frame in 10 chunks. Ottenuti i chunks, si selezionano i frame seguendo tale politica: per i primi due e gli ultimi due viene preso il frame più interno, mentre per i chunks rimanenti viene selezionato il frame centrale.

Dopo questo particolare campionamento, un video viene rappresentato attraverso 10 frame. Sebbene 10 frame possano sembrare pochi, si assicura che data la lunghezza media del video (2,41 sec.) e del frame rate pari a 25, essi sono più che sufficienti per dare una rappresentazione omogenea dell'intera sequenza.

Al fine, però, di rendere tutto più chiaro, nelle Figure 5.1 e 5.2, viene riportata la rappresentazione grafica del campionamento implementato.

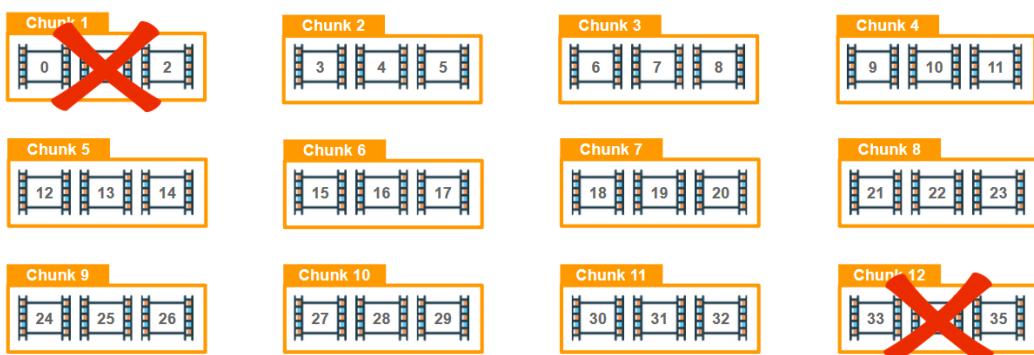


Figura 5.1: Prima fase: Pre-campionamento.

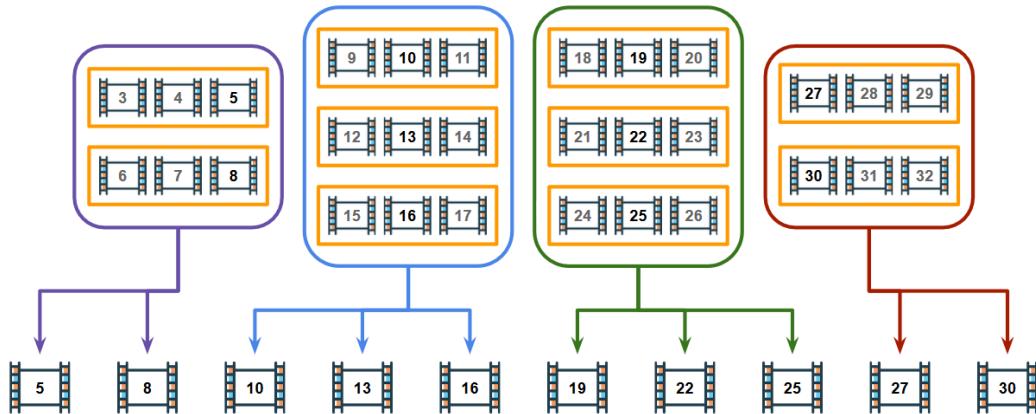


Figura 5.2: Seconda fase: Campionamento.

Nell'esempio proposto è stato usato un video fittizio lungo 36 frame, al fine di avere chunks della stessa lunghezza e ben organizzati. In particolare, nella Figura 5.1 si mette in evidenza la suddivisione dei frame totali in 12 blocchi. I chunks segnati con delle ‘X’ rappresentano quelli scartati. Nella Figura 5.2, invece, viene effettuato il campionamento vero e proprio. Al fine di capire bene quali frame vengono selezionati dall'algoritmo basta guardare i 10 numeri all'interno dei 10 frame selezionati dai 10 chunks. Si noti che per i primi due e gli ultimi due blocchi vengono selezionati i frame 5, 8, 27 e 30 (quelli più interni), mentre per i restanti chunks si seleziona il frame più centrale: 10, 13, 16, 19, 22 e 25.

Infine, nella Figura 5.3 vengono mostrati degli esempi concreti di applicazione del doppio campionamento a due video del dataset WLASL.



Figura 5.3: Applicazione del doppio campionamento.

5.1.2 Data Augmentation

Nel Capitolo 3 si è introdotta la data augmentation e come essa può essere utile sia per regolarizzare il modello, sia per aumentare il numero di samples a disposizione per il training.

Anche in questo caso si è dovuto procedere, in maniera del tutto custom, ad introdurre la data augmentation nel modulo di campionamento. Avendo però pieno controllo del modulo si è potuto decidere in maniera del tutto arbitraria il numero di opzioni da applicare a run-time.

La lista delle opzioni utilizzate è di seguito elencata:

- rotazione oraria e antioraria;
- shift orizzontale e verticale;
- zoom sull'asse verticale e orizzontale;
- flip orizzontale;
- variazione di luminosità.

In base al task da trattare si è deciso di applicare solo le opzioni più adeguate. Infatti, applicare il flip verticale in questo tipo di problematica non avrebbe senso. Anche avere dei valori troppo accentuati porterebbe a dei risultati non voluti, ad esempio, i gradi di rotazione sono stati scelti per non ruotare troppo l'immagine. Nella pratica, infatti, non si avrà mai una persona posizionata di 90° rispetto alla telecamera.

In questo caso però, bisogna sottolineare il fatto che la data augmentation deve essere applicata a sequenze di 10 frame alla volta. Infatti, scelto il numero di opzioni, esse devono essere applicate a tutta la sequenza, come mostrato in Figura 5.4.

Nell'esempio proposto si hanno due coppie di sequenze. Per ogni coppia, la sequenza in alto rappresenta la sequenza di frame originale, mentre quella in basso rappresenta la stessa, ma modificata.

Per concludere, si noti che il numero e il valore delle varie opzioni viene scelto in maniera causale e a run-time.

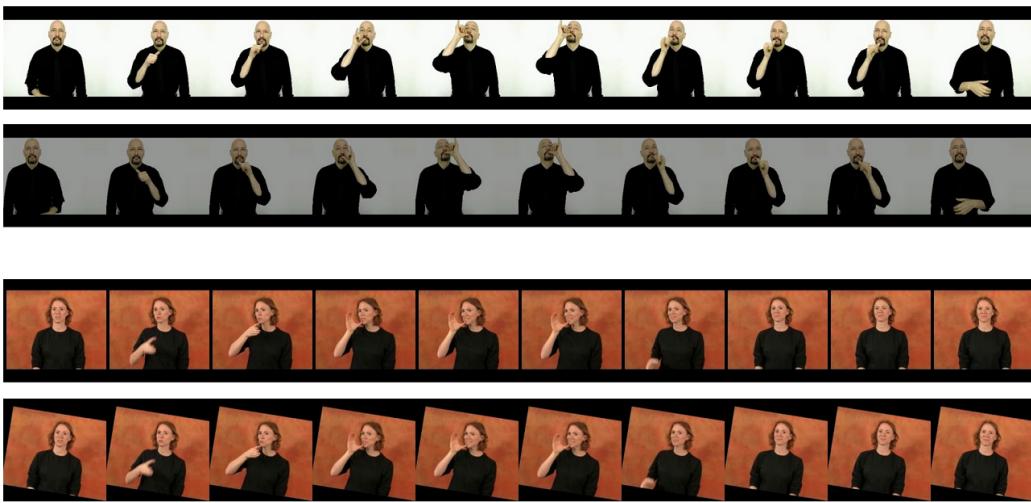


Figura 5.4: Applicazione della data augmentation al frame sampling.

5.2 Fase 2: Estrazione delle feature Spaziali

La seconda fase prevede l'inzio dell'implementazione vera e propria della rete neurale. Come accennato in precedenza, la prima rete neurale che viene utilizzata all'interno del modello proposto è una convolutional neural network. Essa sarà utilizzata come estrattore di feature spaziali dai vari frame.

Nel Capitolo 3 si è spiegato come una ConvNet riesce ad estrarre le feature spaziali attraverso i layer di convoluzione e di pooling. Tuttavia il discorso fatto va bene se si deve trattare una sola immagine alla volta, ma ora deve essere esteso per poterlo applicare ad una sequenza di immagini.

5.2.1 Time Distributed Layer

Sebbene, le ConvNets non sono predisposte per poter gestire sequenze di immagini in maniera naturale, ci sono vari modi per poter ovviare a tale problema.

Un primo approccio per far fronte a questa problematica è mostrato nella Figura 5.5. Questo approccio, molto intuitivo, non fa altro che gestire la sequenza di N immagini attraverso l'uso di N ConvNets. Anche se in teoria l'approccio dovrebbe funzionare, nella pratica, però, sorgono delle inevitabili problematiche.

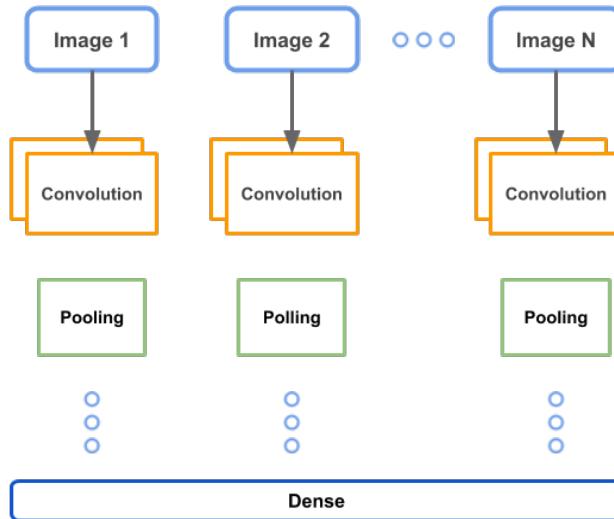


Figura 5.5: Approccio intuitivo.

In particolare quello che succede è che:

- il training diventa molto lungo in quanto si ha il bisogno di addestrare diversi flussi di convoluzione (uno per immagine/frame);
- ogni flusso di convoluzione aggiorna in maniera indipendente i propri pesi. Di conseguenza le varie feature rilevate non saranno correlate tra i vari flussi;
- alcuni flussi di convoluzione non rilevano ciò che altri flussi possono.

Tutte queste problematiche spingono ad abbandonare questo approccio. Il motivo principale, però, è dovuto al fatto che quello che si vuole ottenere è che i vari flussi di convoluzione trovino le stesse feature per ogni immagine. L'alternativa, dunque, è quella di usare un layer particolare chiamato TimeDistributed Layer [22].

Il TimeDistributed layer (TDL) è un particolare layer che non fa altro che applicare i layers di una ConvNet, in maniera sequenziale, ai vari input. In altre parole, il TimeDistributed layer applica la stessa istanza di ConvNet ad ogni timestamp, facendo sì che gli stessi pesi siano condivisi tra i vari timestamp. Grazie a questa condivisione, il TDL riesce a rilevare le stesse feature spaziali su tutta la sequenza di immagini.

Ad esempio, se noi iniettassimo 5 immagini in una ConvNet contenuta in un TimeDistributed layer, i pesi non saranno aggiustati 5 volte, ma solo una volta in quanto essi sono distribuiti tra ogni blocco definito nel TimeDistributed layer. Tutto questo, inoltre, si traduce in un uno sforzo computazionale minore rispetto al primo approccio.

Infine, si propone una rappresentazione grafica (Figura 5.6) del TDL per far apprezzare al miglior modo il suo funzionamento.

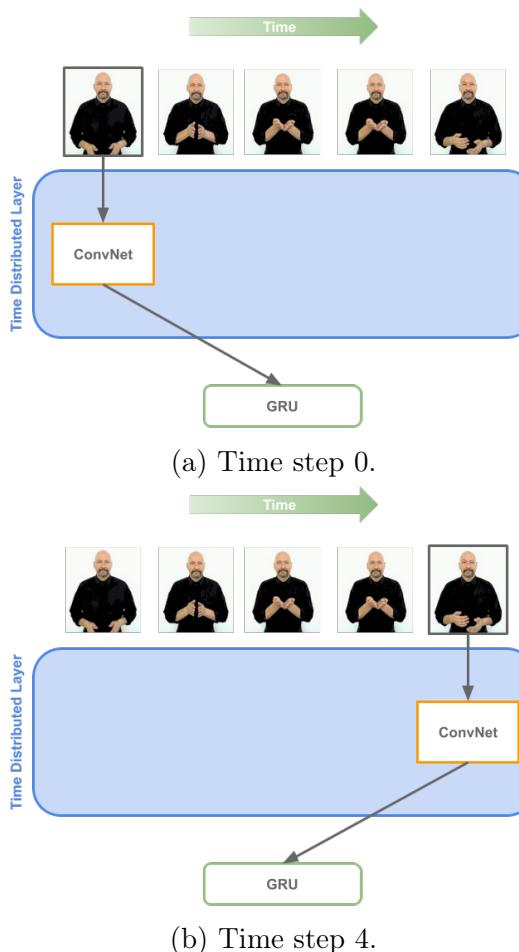


Figura 5.6: Funzionamento del TimeDistributed Layer.

Capito come poter estrarre le feature spaziali in modo corretto da una sequenza di immagini (frames), bisogna solo scegliere quale ConvNet le dovrà estrarre.

5.2.2 MobileNetV2

Creare una buona ConvNet ‘from scratch’ risulta molto impegnativo e time-consuming. Inoltre l’hardware a disposizione deve essere non solo abbastanza potente, ma anche in grado di lavorare a pieno carico per un numero di ore non indifferente. Tuttavia, come discusso nel Capito 3, la tecnica del transfer learning aiuta a superare questo ostacolo.

Negli anni sono state presentate molte reti neurali pre-addestrate appositamente per tale scopo. Tra di esse è stata scelta la rete MobileNetV2 [37].

Tra le tante ConvNets a disposizione si è deciso di utilizzare tale rete in quanto essa è stata progettata e ottimizzata appositamente dal team di Google per utilizzarla in applicazioni real-time, nei dispositivi mobili ed embedded. Volendo creare un sistema real-time per la traduzione della lingua dei segni, tale rete calza a pennello.

Caratteristiche

Al fine di dare priorità all’efficienza, la seconda versione di MobileNet, che si basa interamente sulla prima versione, introduce due nuove feature fondamentali:

- aggiunta di linear bottlenecks tra i layers;
- aggiunta di shortcut connections tra le bottlenecks.

I linear bottlenecks codificano gli input e gli output intermedi del modello migliorando così l’apprendimento delle feature. Mentre, l’inner layer, tra due bottlenecks, incapsula la capacità del modello di trasformarsi da concetti di basso livello (singoli pixel) a concetti di alto livello (categorie di immagini). Infine, così come le residual connections tradizionali, le shortcut connections consentono di ottenere un training più rapido e un’accuracy migliore.

Di seguito viene riporta la rappresentazione grafica (Figura 5.7) dei moduli aggiuntivi presenti nella rete MobileNetV2.

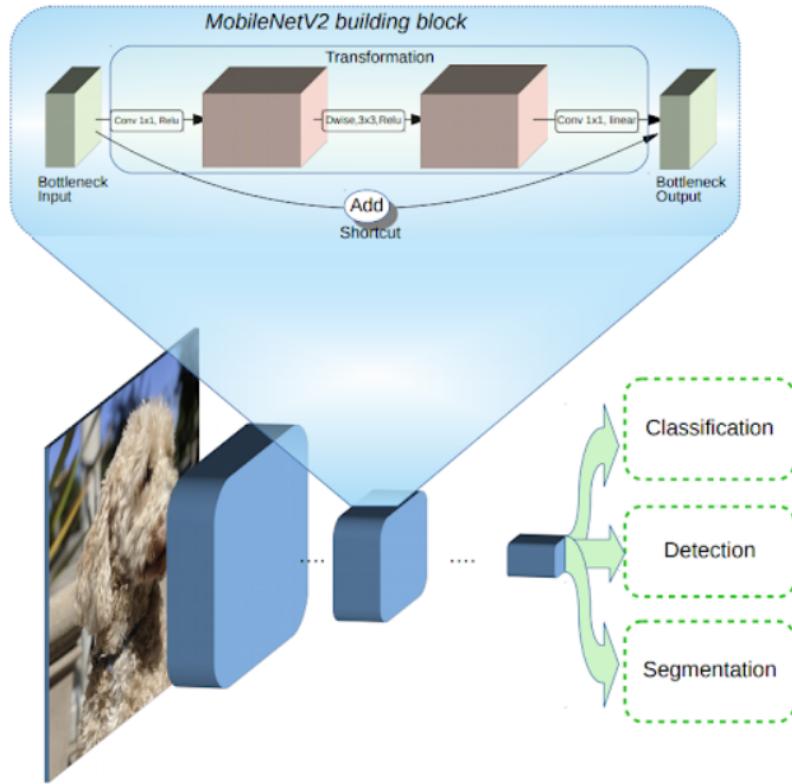


Figura 5.7: Moduli aggiuntivi della rete MobileNetV2^[38].

La Tabella 5.1, invece, propone un'overview dei layer che compongono l'intera rete MobileNetV2.

Input	Layer	Filters	Stride	Repeated layer
224 x 224	conv2d	3	2	1
112 x 112	bottleneck	32	1	1
112 x 112	bottleneck	16	2	2
56 x 56	bottleneck	24	2	3
28 x 28	bottleneck	32	2	4
14 x 14	bottleneck	64	1	3
14 x 14	bottleneck	96	2	3
7 x 7	bottleneck	160	1	1
7 x 7	conv2d	320	1	1
7 x 7	avgpool	1280	-	1
1 x 1	conv2d	1280	-	-

Tabella 5.1: Architettura della rete MobileNetV2.

5.3 Fase 3: Estrazione delle feature temporali

La terza fase di progettazione del modello consiste nel riuscire a catturare le relazioni temporali, che sussistono nella sequenza di frame, attraverso reti neurali che siano in grado di farlo in maniera del tutto “naturale”.

Come discusso ampiamente nel Capitolo 3, per il modello proposto, si è deciso di utilizzare le reti ricorrenti GRU in quanto più efficienti e veloci da addestrare rispetto alle LSTM.

Prima di capire come combinare una rete neurale ricorrente (GRU) con una convoluzionale (MobileNetV2 + Time Distributed Layer), è bene definire qual è il tipo di problema che si vuole affrontare.

5.3.1 Tipi di GRU

Quando si parla di RNN, dunque anche di GRUs, il problema da risolvere può ricadere in una delle seguenti architetture:

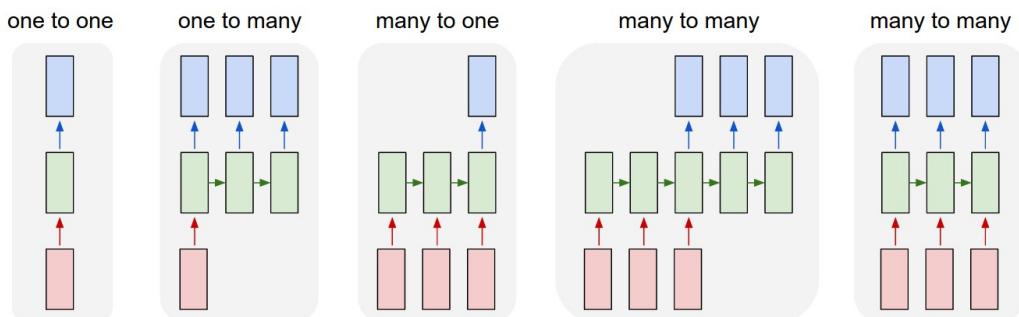


Figura 5.8: Possibili architetture da usare con le reti ricorrenti^[21].

- **One to one:** l’architettura più semplice che si può avere con una RNN. In questo caso abbiamo un solo input a cui viene associato un output. Questo approccio può essere utile quando ciò che si vuole predire è legato strettamente a ciò che è accaduto al passo precedente;
- **One to many:** questa architettura consente di associare più output ad un solo input. Questo approccio è molto utile, ad esempio, in ambito musicale: data una singola nota generare la sequenza successiva di note;

- **Many to one:** questa architettura prende come input una sequenza e ritorna come output un solo valore. Molteplici applicazioni sfruttano questa architettura. Ad esempio, in ambito del sentiment analysis, quello che si vuole fare è predire, dato una recensione, un valore tra 1 e 5, indicante il grado di apprezzamento;
- **Many to many:** l'ultima architettura prende in ingresso una sequenza di input e ritorna come output un'altra sequenza. In questo caso però possiamo avere due varianti: 1. la lunghezza di input e output è uguale; 2. la lunghezza di input e output è diversa (si pensi alla traduzione come una possibile applicazione di questo genere).

Appurato che tutte queste architetture posso essere utilizzate da una GRU, bisogna solo capire quale di essa è più adatta al modello da implementare.

Ragionandoci brevemente è facile intuire che l'approccio più sensato per l'obiettivo prefissato è il many to one. Questo perché quello che si deve fare è dare in pasto alla rete una sequenza di N frame ed effettuare una classificazione in base al singolo valore ritornato dalla GRU.

Dato le GRU gestiscono le informazioni attraverso vettori monodimensionali, bisogna convertire i vettori bidimensionali, rappresentati i frame pre-processati dalla ConvNet, in vettori monodimensionali; per fare ciò si utilizza il *GlobalAveragePooling layer*. Si noti, inoltre, che anche questo layer dovrà essere inserito nel TimeDistributed layer in quanto deve essere applicato a tutti gli N frame della sequenza.

Una volta passato attraverso il GlobalAveragePooling layer si ottengono N rappresentazioni vettoriali monodimensionali (un vettore per frame). Questi N vettori verranno poi iniettati nella GRU, in maniera sequenziale, al fine di catturare le relazioni temporali che sussistono tra i vari frame nel tempo.

Lo schema rappresentato in Figura 5.9 può aiutare a far maggior chiarezza su ciò che si è appena detto.

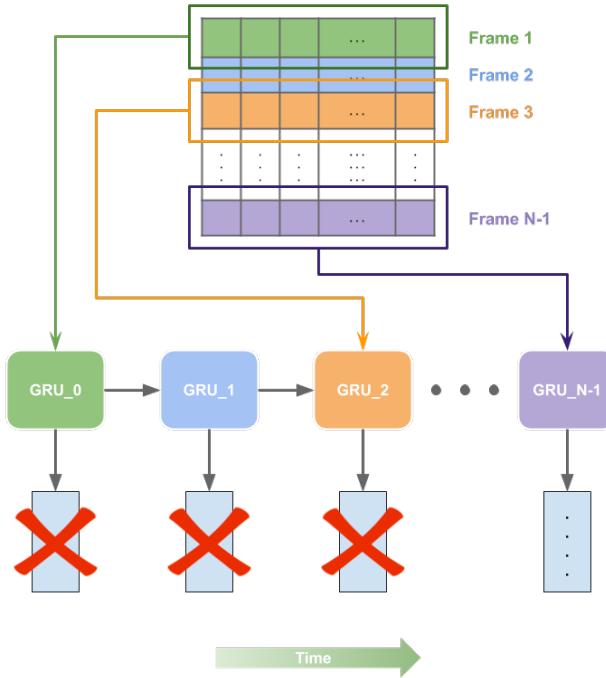


Figura 5.9: Applicazione della GRU alla sequenza di frame.

Dalla Figura 5.9 si può apprezzare il modo in cui gli N frame sono gestiti da una GRU al fine di poter individuare i movimenti della persona all'interno della sequenza. In particolare, per aver chiaro il suo comportamento si deve pensare ad essa in maniera srotolata nel tempo. Questo perché, come ogni rete neurale ricorrente, la GRU può essere vista come un loop in cui ad ogni iterazione gestisce un'informazione. In questo caso specifico, la GRU dovrà gestire un frame ad ogni ciclo di loop.

Questo modo di vedere la GRU aiuta a capire come essa possa estrarre le feature temporali dai singoli frame, o meglio dalle feature spaziali estratte dai frame. Si ricordi che i vettori nella figura sovrastante non sono altro che la rappresentazione numerica delle features spaziali estratte grazie alla ConvNet.

Analizzando da sinistra verso destra, il loop creato dalla GRU, si ha che la prima cella GRU (*GRU_0*) prende in input il vettore rappresentante il primo frame ed inizia con l'estrazione delle feature temporali. Una volta terminato questo primo passo, il successivo vede entrare in gioco la cella *GRU_1*, la quale sfruttando le informazioni temporali e l'input del secondo frame decide

quali features ricordare e quali dimenticare al fine di iniziare a catturare il movimento nella sequenza. Procedendo iterativamente si arriverà all'ultima cella GRU che avrà appreso il movimento effettuato e darà in output una sua rappresentazione vettoriale.

Ottenuta la rappresentazione numerica nel movimento essa verrà usata in fase di classificazione per predire la gloss esatta.

5.4 Fase 4: Classificazione

L'ultimo passo che rimane da affrontare è quello della classificazione, ovvero della predizione della gloss in base alla sequenza dei frame dati in input.

In particolare, il task ricade nella categoria multi-class classification in quanto abbiamo N labels (glosses = parole) disgiunte.

Per effettuare tale classificazione abbiamo bisogno di una semplice rete neurale feedforward, utilizzando la softmax come funzione di attivazione nell'ultimo layer. Tale funzione è molto importante in quanto permette di associare delle probabilità alle classi di interesse. In altre parole, inviata una sequenza di frame alla rete, essa assocerà una probabilità ad ogni gloss: la gloss predetta sarà quella avente la probabilità più alta.

Per quanto riguarda le dimensionalità, invece, non si avranno problemi in quanto la GRU darà come output un vettore monodimensionale, consentendo alla rete feedforward di gestirlo in maniera del tutto naturale.

5.5 Modello Finale

Mettendo insieme i vari pezzi discussi nei precedenti paragrafi, si ottiene il seguente modello in Figura 5.10. Tale modello è un'architettura di tipo end-to-end, ovvero, addestrabile in maniera unica senza dover procedere ad allenare, in maniera separata, la rete neurale convoluzione e la rete ricorrente.

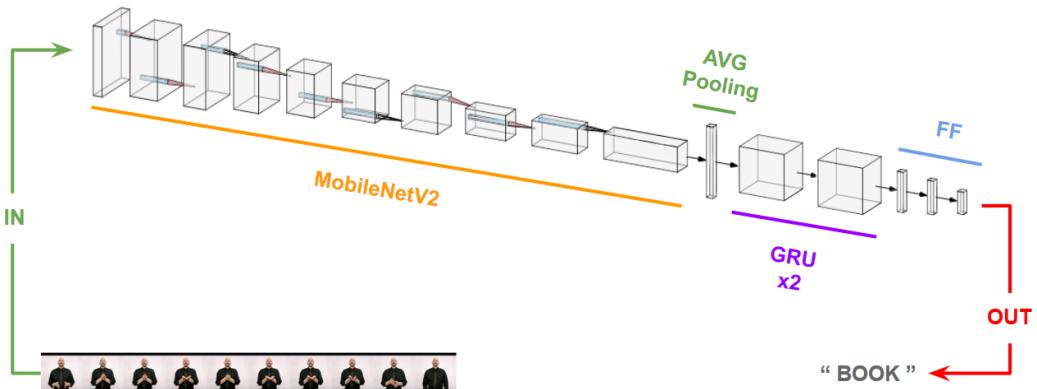


Figura 5.10: Architettura finale.

Il modello presentato nella figura sovrastante mette in evidenza la profondità della rete neurale da allenare.

Ripercorrendo quello che è il compito preposto di tale rete, essa dovrà:

1. campionare ad ogni iterazione una nuova sequenza di 10 frame dal video selezionato per il training;
2. sfruttare la ConvNet contenuta nel TimeDistributed layer per individuare la persona all'interno dei video;
3. utilizzare il Pooling layer per trasformare le dimensionalità dei vettori, ottenuti dalle rete MobileNetV2, passando da una rappresentazione 2D ad una 1D;
4. utilizzare due GRU, una di seguito all'altra, per estrarre in maniera precisa ed efficace le relazioni temporali che sussistono tra i vari frame;
5. classificare il vettore rappresentativo della sequenza di frame, dato in output dalla GRU, attraverso una rete feedforward.

Come accennato in precedenza tale rete è allenata in maniera end-to-end, ciò significa che in un step di training verranno effettuati tutti e quattro i passaggi sopra esplicati.

5.5.1 Implementazione

Nella pratica, il modello è stato implementato attraverso il framework Keras [7] e il codice che lo traduce nella corrispettiva rete neurale è riportato di seguito:

```

1 model = Sequential([
2     # convnet - MobileNetV2
3     TimeDistributed(
4         MobileNetV2(weights='imagenet', include_top=False,
5                     input_shape=[224, 224, 3]),
6         input_shape=[10, 224, 224, 3]
7     ),
8     TimeDistributed(GlobalAveragePooling2D()),
9     # stacked GRUs
10    GRU(256, return_sequences=True),
11    BatchNormalization(),
12    GRU(256),
13    # feedforward
14    Dense(units = 1000, activation='relu'),
15    Dropout(0.75),
16    Dense(units = 2000, activation='softmax')
17 ])

```

Listing 5.1: Modello proposto.

Alla riga 3 viene istanziato il TimeDistributed layer. Esso prende in input la MobileNetV2, le dimensioni del frame e il numero indicante la lunghezza della sequenza da gestire.

Per quanto riguarda la MobileNetV2 ci sono due elementi che devo essere commentati:

1. i presi pre-addestrati sono quelli utilizzati effettuando il training sul dataset ImageNet [8];
2. i frame in input devono avere la dimensione fissa di 224x224x3, dove 224x224 indica la larghezza e l'altezza del frame, il 3, invece, indica il numero di canali per gestire il colore (3 = RGB, 1 = B/W).

La grandezza del frame è abbastanza standard nei vari modelli di transfer learning, infatti, oltre che in MobileNetV2 la si ritrova nelle altre principali

reti. Per quanto riguarda la dimensione dell'input, il TimeDistributed layer indica il numero di frame da gestire nel tempo, ovvero 10.

Alla riga 8 viene incluso il GlobalAveragePooling layer nel TimeDistributed layer, per i motivi precedentemente citati.

Alle righe 10-12 viene implementata la RNN. In questo caso si è preferito impilare due GRU per far sì che il modello possa operare a time-scale differenti, ottenendo, potenzialmente, una migliore accuratezza nella predizione. Inoltre, tra le due GRU viene utilizzata la batch normalization per cercare di rendere il training più veloce e più stabile.

Alle righe 14-16 viene implementata la rete feedforward per effettuare la classificazione. Tale rete è costituita da semplici dense layer, dove l'ultimo ha come funzione di attivazione la softmax, come specificato in precedenza. Inoltre, viene applicato del dropout per evitare che il modello overfitti.

Il modello presentato è quello utilizzato per il training sull'intero dataset (WLASL200). Infatti, l'ultimo layer presenta 2.000 neuroni che stanno ad indicare le 2.000 parole da predire. Come ci si può immaginare la rete feedforward è destinata a cambiare per i restanti sub-datasets del WLASL. In particolare, per il WLASL100 la rete feedforward è strutturata come seguente:

```

1 # feedforward
2 Dense(units = 200, activation='relu'),
3 Dropout(0.66),
4 Dense(units = 150, activation='relu'),
5 Dropout(0.66),
6 Dense(units = 100, activation='softmax')
```

Listing 5.2: WALS100.

Invece, per il WLASL300 la rete feedforward è la seguente:

```

1 # feedforward
2 Dense(units = 400, activation='relu'),
3 Dropout(0.6),
4 Dense(units = 350, activation='relu'),
5 Dropout(0.6),
6 Dense(units = 300, activation='softmax')
```

Listing 5.3: WALS1300.

Mentre, per il WSLA1000 la rete feedforward è la seguente:

```

1 # feedforward
2 Dense(units = 512, activation='relu'),
3 Dropout(0.7),
4 Dense(units = 1000, activation='softmax')
```

Listing 5.4: WSLA1000.

Inoltre, è stato addestrato un modello su un dataset custom di soli 20 gloss da impiegare nella demo real-time. Si approfondirà questo discorso nei Capitoli 5 e 6, ma, per completezza, si riporta di seguito la rete feedforward per tale modello:

```

1 # feedforward
2 Dense(units = 64, activation='relu'),
3 Dropout(0.65),
4 Dense(units = 32, activation='relu'),
5 Dropout(0.65),
6 Dense(units = 20, activation='softmax')
```

Listing 5.5: WSLA20 custom.

In generale, il modello proposto è molto simile al modello implementato dagli autori del paper [25]. La modifica sostanziale si ha nell'utilizzo della rete MobileNetV2 al posto della rete VGG16.

Capitolo 6

Risultati

I risultati riportati sui vari sotto-dataset presentati in [25] sono stati ottenuti effettuando il training nell'ambiente di sviluppo Google Colaboratory (Colab) [14]. Colab è un prodotto di Google Research che consente a chiunque di scrivere ed eseguire codice Python in un ambiente già configurato. In particolare, esso si presta bene per la gestione di notebook per lo svolgimento di task di machine learning e intelligenza artificiale. Il vantaggio principale, però, sta nel poter aver accesso in maniera gratuita alle risorse di elaborazione, comprese le GPUs. Per garantire la gratuità del servizio, Google adotta politiche restrittive sull'uso intensivo delle GPUs; per tale motivo è importante applicare politiche di stop/restart del training per allenare i modelli nel lungo periodo.

Al fine di effettuare un confronto omogeneo tra i modelli, il training è stato effettuato sotto le stesse condizioni:

1. batch-size uguale a 8;
2. frame sampling uguale a 10.

L'analisi dei risultati verrà affrontata partendo dal sotto-dataset più piccolo fino ad arrivare al dataset completo, rispecchiando quello che è stato il percorso seguito in pratica. Prima di procedere, si ricordi che, i sotto-dataset sono organizzati per ordine di samples. Ad esempio, il dataset WLALS100 conterrà le prime 100 gloss a cui sono associate il più alto numero di video.

6.1 WLASL 100

Il primo sotto-dataset considerato è stato WLASL100, cioè quello composto dalle prime 100 gloss. Le caratteristiche di tale dataset sono riportate nella seguente tabella:

	#Video	#Train	#Validation	#Test
		70%	17%	13%
WLASL100	2.038	1.442	338	258

Tabella 6.1: WLASL100.

Il modello proposto per tale dataset è stato allenato per 120 epoche in circa 8 ore, producendo, per loss e accuracy, le seguenti learning curves:

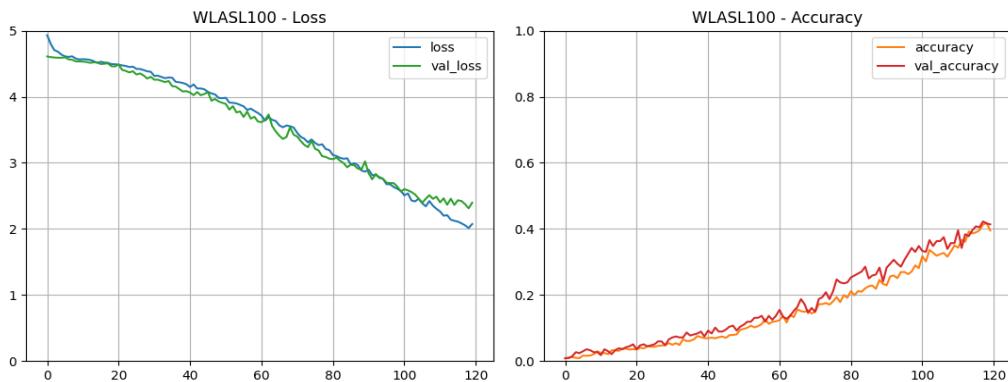


Figura 6.1: Learning curves del training effettuato sul dataset WLASL100.

Nello specifico, il training ha consentito al modello di raggiungere un'accuracy del 33,70% sul test set a fronte del 25,97% raggiunto in [25].

Le learning curves raffigurate in Figura 6.1 riportano un storico generale su quello che è stato il training del modello posposto per questo sotto-dataset. In particolare, si è ritenuto accettabile terminare il training a circa 120 epoche in quanto da quel momento in poi il modello avrebbe cominciato ad overfittare il train set (si noti come la validation loss si inizi a distaccarsi drasticamente dalla loss).

Infine, è bene fare una precisazione in questa fase: sebbene in tasks di questo genere l'early stopping [33] sia ampiamente utilizzato, per questo mo-

dello e per tutti gli altri proposti, si è deciso di procedere in maniera manuale alla selezione del modello e, quindi, al numero di epoche. Questa decisione è stata dettata dal fatto che, nella pratica, durante i training iniziali sul dataset [25] è stato riscontrato un andamento molto altalenante della loss, vanificando l'applicazione della tecnica citata.

6.2 WLASL 300

Il secondo sotto-dataset analizzato è stato WLASL300, cioè quello composto dalle prime 300 gloss. Le caratteristiche di tale dataset sono riportate nella seguente tabella:

	#Video	#Train	#Validation	#Test
WLASL300	5.117	3.549	69% 900	18% 13% 668

Tabella 6.2: WLASL300.

Il modello proposto per tale dataset è stato allenato per 55 epoche in circa 9 ore, producendo, per loss e accuracy, le seguenti learning curves:

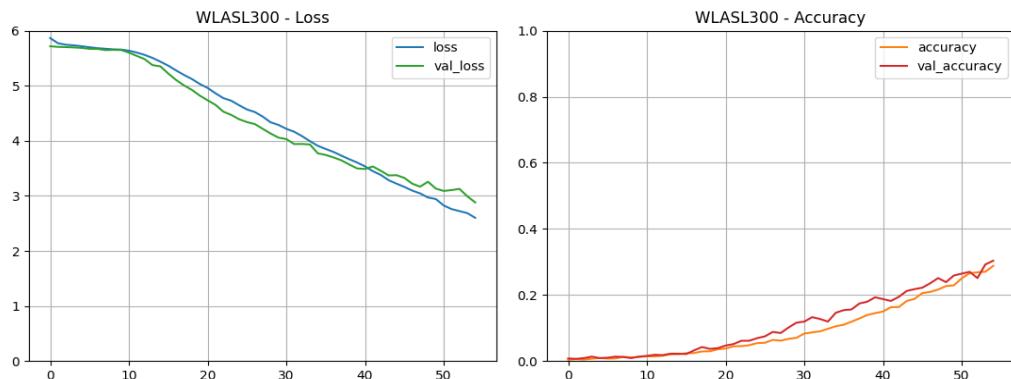


Figura 6.2: Learning curves del training effettuato sul dataset WLASL300.

Nello specifico, il training ha consentito al modello di raggiungere un'accuracy del 28,15% sul test set a fronte del 19,31% raggiunto in [25].

Anche in questo caso, le learning curves raffigurate in Figura 6.2 riportano un storico generale su quello che è stato il training del modello posposto per questo sotto-dataset. In particolare, salta subito all'occhio come il numero di epoche sia sceso drasticamente. Questo è dovuto al fatto che, al crescere del dataset, l'accuracy da raggiungere e superare diventa sempre più bassa. Nel caso specifico si è allentato il modello fino alla sessantesima epoca per evitare l'imminente overfitting (come dimostra l'andamento curvilineo della validation loss).

6.3 WLASL 1000

Il terzo sotto-dataset analizzato è stato WLASL1000, cioè quello composto dalle prime 1.000 gloss. Le caratteristiche di tale dataset sono riportate nella seguente tabella:

	#Video	#Train	#Validation	#Test	
WLASL1000	13.168	13.168	8.974	2.318	1.876

Tabella 6.3: WLASL1000.

Il modello proposto per tale dataset è stato allenato per 25 epoche in circa 11 ore, producendo, per loss e accuracy, le seguenti learning curves:

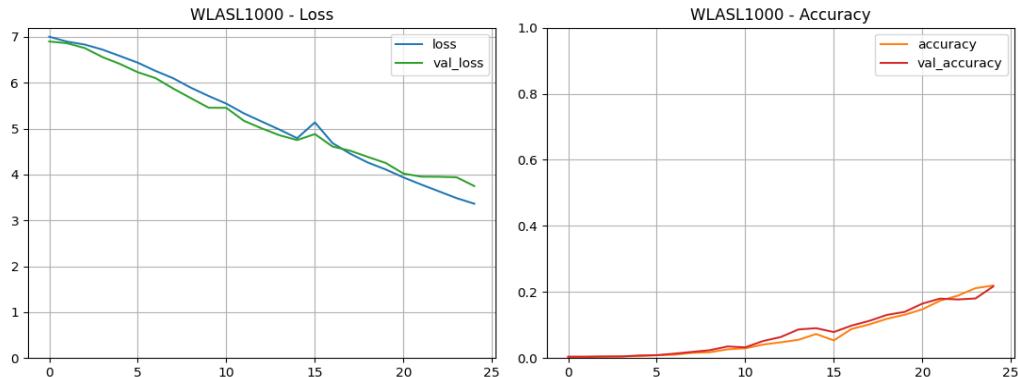


Figura 6.3: Learning curves del training effettuato sul dataset WLASL1000.

Nello specifico, il training ha consentito al modello di raggiungere un'accuracy del 19,01% sul test set a fronte del 14,66% raggiunto in [25].

Come prima, le learning curves raffigurate in Figura 6.3 riportano un storico generale su quello che è stato il training del modello posposto per questo sotto-dataset. In particolare, si è ritenuto accettabile terminare il training a circa 25 epoche per evitare che il modello iniziasse la sua tendenza all'overfitting. Si noti, inoltre, che le epoche sono diminuite ancora, ma il tempo di training è aumentato drasticamente, sottolineando la pesantezza dell'addestramento di una rete end-to-end su un dataset che comincia ad essere estremamente corposo.

Per questo dataset bisogna, però, fare una considerazione particolare. Il metodo di campionamento utilizzato può essere applicato solo se il video ha una lunghezza di almeno 12 frame. In questo dataset, i video 59958.mp4 e 18223.mp4 hanno una lunghezza inferiore. Siccome i video da trattare sono solo due si è proceduto ad estenderli, fino alla lunghezza minima, con uno script custom, senza modificare il modulo di campionamento.

6.4 WLASL 2000

Il quarto ed ultimo sotto-dataset analizzato è stato WLASL2000, ovvero il dataset completo. Esso è composto da tutte le 2.000 gloss. Le caratteristiche di tale dataset sono riportate nella seguente tabella:

	#Video	#Train	#Validation	#Test				
WLASL2000	21.083	21.083	68%	14.289	18%	3.916	2.878	14%

Tabella 6.4: WLASL2000.

Il modello proposto per tale dataset è stato allenato per 25 epoche in circa 17 ore, producendo, per loss e accuracy, le learning curves presenti nella Figura 6.4.

Il training ha consentito al modello di raggiungere un'accuracy del 10,07% sul test set a fronte del 8,44% raggiunto in [25].

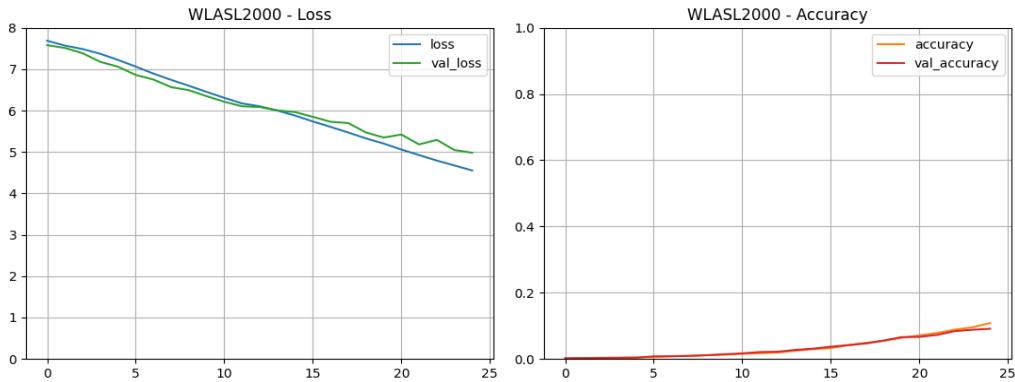


Figura 6.4: Learning curves del training effettuato sul dataset WLASL2000.

Le learning curves raffigurate in Figura 6.4 riportano un storico generale di quello che è stato il training del modello posposto per questo sotto-dataset. In questo caso il numero di epoche è rimasto invariato rispetto al dataset precedente, raggiungendo però un'accuracy molto minore. Questo è dovuto al fatto che, sebbene il numero di glosses sia raddoppiato, non può essere detto lo stesso per il numero di samples (si ricordi che i dataset sono stati organizzati in base alla frequenza dei video, quindi il dataset WLASL2000 presenta moltissime alle quali sono associate poche istanze di riferimento). Tutto ciò ha contribuito ad avere un'accuracy molto bassa e ad un precoce inizio di overfitting, infatti, il distaccamento delle due loss avviene già all'epoca 15.

Infine, anche per il suddetto dataset bisogna fare le stesse considerazioni fatte per il WLASL1000. In questo caso oltre ai due video precedenti, sono stati estesi i video 15144.mp4, 02914.mp4 e 55325.mp4. Anche in questo caso si è proceduto manualmente.

6.5 Risultati Finali

Al fine di raggruppare e mettere a confronto i risultati ottenuti con quelli presentati in [25], si può osservare la tabella 6.5.

La Tabella 6.5 mette in evidenza il superamento delle performance in tutti e quattro i sotto-dataset proposti per il WLASL. In particolare, per i primi due casi si è avuto un miglioramento di circa 8~10% in termini di accuracy.

	WLASL100	WLASL300	WLASL1000	WLASL2000
Modello [25]	25,97%	19,31%	14,66%	8,44%
Modello proposto	33,70%	28,15%	19,01%	10,07%

Tabella 6.5: Accuratezza del modello proposto e del modello presentato in [25] sui dataset considerati.

Mano mano che il dataset crescesse l'aumento è stato sempre minore, fino ad arrivare a circa 2% sul dataset completo.

I risultati ottenuti mettono ben in evidenza come il task affrontato sia molto complesso. Infatti, come salta subito all'occhio, sui primi due dataset l'aumento delle prestazioni è netto, mentre, man mano che si procede, le prestazioni tendono a non migliorare più di tanto. Questo è dovuto principalmente alla struttura del dataset perché, seppur vero che i video sono molti, è anche vero che quando si utilizza l'intero dataset i samples per singola gloss diventa molto basso (range 2-4). Tutto questo comporta che sull'intero dataset il modello fa più fatica a riconoscere i segni, proprio perché il numero non elevato di samples non gli consente un buon addestramento.

6.6 WLASL 20 Custom

Sebbene i risultati ottenuti superino quelli ottenuti in [25], essi non raggiungono un'accuratezza tale da garantire un buon riconoscimento all'interno di un sistema real-time.

Per arginare tale inconveniente, si è deciso di lavorare su un dataset ridotto, contenente solo 20 gloss. Per tale dataset sono state scelte le parole più significative per l'uso quotidiano, cercando, comunque, di mantenere un numero discreto di samples. La lista delle parole viene riportata di seguito:

<i>Book</i>	<i>Chair</i>	<i>Clothes</i>	<i>Computer</i>	<i>Drink</i>
<i>Drum</i>	<i>Family</i>	<i>Footbal</i>	<i>Go</i>	<i>Hat</i>
<i>Hello</i>	<i>Kiss</i>	<i>Like</i>	<i>Play</i>	<i>School</i>
<i>Street</i>	<i>Table</i>	<i>University</i>	<i>Violin</i>	<i>Wall</i>

Data la presenza di sole 20 parole si è deciso di chiamare il dataset WLASL20custom. Le caratteristiche di tale dataset sono riportate di seguito:

	#Video	#Train	#Validation	#Test
			70%	17%
WLASL20 custom	403	285	67	51

Tabella 6.6: WLASL20custom.

Il modello proposto per tale dataset è stato allenato per 260 epoche in circa 3,5 ore, producendo, per loss e accuracy, le seguenti learning curves:

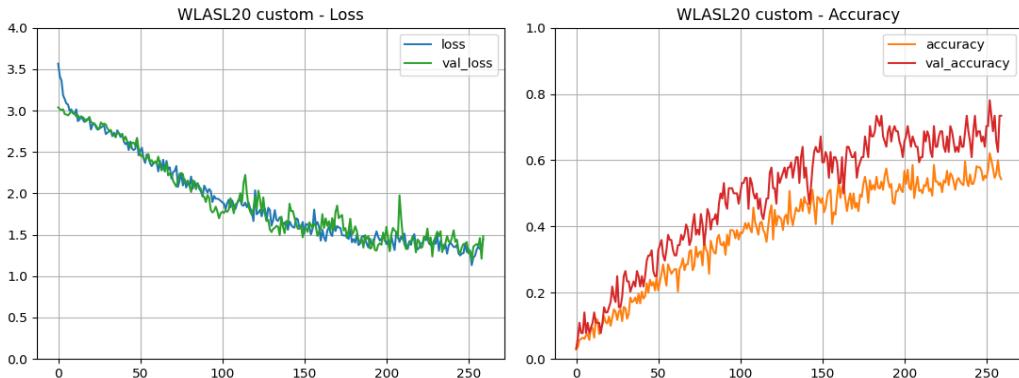


Figura 6.5: Learning curves del training effettuato sul dataset WLASL20 custom.

Il training ha consentito al modello di raggiungere un'accuracy del 63,36% sul test set.

Un discorso completamente diverso da quelli fatti in precedenza, può essere attribuito alle learning curves raffigurate in Figura 6.5. Esse riportano un storico generale di quello che è stato il training del modello posposto per il dataset custom. In questo caso il training si è protratto per un numero di epoche molto più elevato rispetto a tutti gli altri modelli. Questo è dovuto al

fatto che per le poche glosses scelte il numero di samples è abbastanza elevato, consentendo al modello di apprenderne le caratteristiche senza cadere nella trappola dell'overfitting.

In particolare, il totale delle epoche mostrate, come detto in precedenza, si aggira in torno alle 260 e fino al quel momento non hanno manifestato nessun tipo di overfitting. A maggior prova della validità del modello si può guardare anche alle learning curves dell'accuracy (destra dell'immagine): l'accuracy sul validation si comparata in maniera migliore di quella del training set, rafforzando la validità del modello stesso.

Infine, grazie all'accuracy raggiunta, il sistema di riconoscimento dei segni ha ottenuto delle buone performance. La discussione dei dettagli, però, viene lasciata al prossimo capitolo.

Capitolo 7

Web App

Al fine di mettere alla prova il modello allenato sul dataset WLASL20custom, è stata implementata un'applicazione di tipo client-server in grado di tradurre i gesti acquisiti tramite webcam nel corrispettivo significato in lingua americana, il tutto in maniera real-time.

Con il seguente schema si cerca di far chiarezza sull'architettura che la web app deve avere:

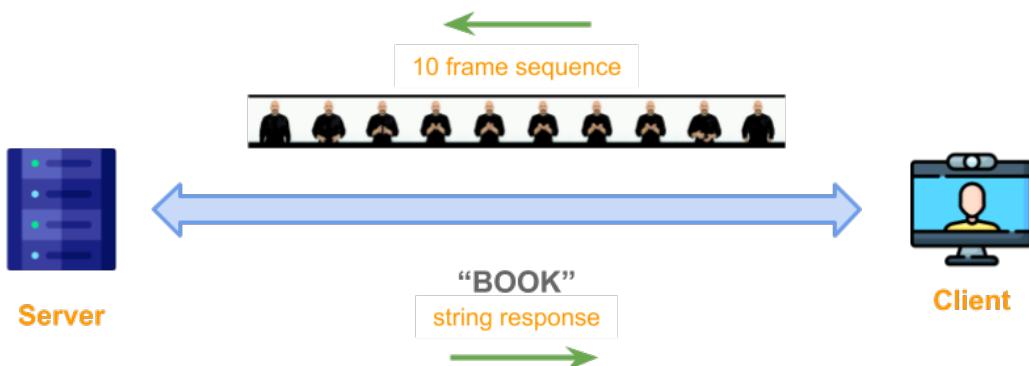


Figura 7.1: Architettura della Web Application.

Come suggerisce il nome, sono presenti due agenti: il client e il server. Lato client viene avviato uno streaming video, attraverso una webcam, dal quale si estraggono gruppi di 10 frame in maniera ciclica. Una volta riempito il buffer di 10 frame, il client invia tale sequenza al server, il quale, sfruttando il modello addestrato, risponde con la gloss predetta.

7.1 Server

Il lavoro più oneroso, come ci si può immaginare, è svolto dal server che deve essere in grado di predire, data una sequenza di 10 frame, la gloss esatta.

Per l'implementazione del server ci si affida a un framework capace di gestire codice python: Flask.

7.1.1 Flask

Flask [32] è un framework web scritto in python leggero ed estendibile. Grazie alle sue caratteristiche si presta bene per l'implementazione di web app non troppo complesse, ma che richiedano molta flessibilità. Se lo scopo, però, è quello di creare un prodotto professionale fin da subito, allora è il caso di affidarsi a frameworks che offrano solide funzionalità, come Django [10]. Ciò nonostante, per l'obiettivo della tesi è più che sufficiente Flask e ciò che offre.

In una classica applicazione client-server la conversazione tra i due avviene attraverso lo scambio di chiamate HTTP. Questo tipo di approccio risulta un collo di bottiglia per applicazioni di tipo in real-time: essendo la richiesta HTTP di tipo bloccante, il server accetterà una sola richiesta alla volta bloccando qualsiasi altro tipo di interazione con il client. Tutto questo, per l'applicazione proposta, si tradurrebbe in un lungo delay tra il gesto effettuato e la gloss predetta dal server.

L'obiettivo da raggiungere, invece, è quello di riuscire a tradurre in tempo reale i segni che vengono segnati di fronte alla webcam. Per fare questo, dunque, si avrà il bisogno di inviare molteplici frame dal client verso il server, in maniera continua, senza inondarlo con una massiccia quantità di richieste http. Per risolvere tale problema l'unico modo è sfruttare una WebSocket.

7.1.2 Flask-SocketIO

Una WebSocket è un protocollo, funzionante su HTTP, in grado di creare una connessione persistente tra client e server, consentendo una comunicazione bidirezionale tra i due.

Il protocollo HTTP è di tipo *half-duplex*, ciò significa che i dati possono o essere inviati dal client o dal server, ma non contemporaneamente. Inoltre, la richiesta è sempre fatta dal client e il server si limita solo a rispondere. Facendo un'analogia, l'HTTP può essere paragonato ad un walkie-talkie. Di contro, una WebSocket è di tipo *full-duplex*, ovvero dopo una richiesta iniziale unidirezionale, il client e il server possono ‘parlare’ simultaneamente e in modo persistente. Per fare anche in questo caso un’analogia, basti pensare ai telefoni cellulari.

Riassumendo, per il sistema proposto, la WebSocket vince contro il protocollo HTTP per questi 2 motivi:

- La connessione rimane persistente nel tempo: una volta stabilita la connessione il client può inviare in maniera continua sequenze di 10 frame senza nessun tipo di blocco;
- Connessione bidirezionale tra client e server: il server alla ricezione del buffer di frame inizia l’inferenza e solo dopo averla conclusa invia un messaggio di risposta al client; tutto questo senza preoccuparsi delle ulteriori richieste ricevute in contemporanea dal client. Dunque, l’unico delay tra i segno effettuato e la risposta dal server sarà il tempo di inferenza del modello e nient’altro.

Flask può facilmente utilizzare comunicazioni bidirezionali tra client e server sfruttando la libreria Flask-SocketIO [16].

L’esempio di codice seguente mostra come implementare una WebSocket, sfruttando Flask-SocketIO, in un’applicazione Flask:

```

1 from flask import Flask, render_template
2 from flask_socketio import SocketIO
3
4 app = Flask(__name__)
5 socketio = SocketIO(app)
6
7 if __name__ == '__main__':
8     socketio.run(app, host='127.0.0.1', port=5000)

```

Listing 7.1: Implementazione di una WebSocket in Flask.

Queste semplici righe di codice sono sufficienti per creare ed avviare il server. Si noti, però, come il web server è avviato: la funzione `socketio.run()` incapsula lo start-up del web server, sostituendo la funzione standard di Flask `app.run()`. Questo fa sì che il server possa sfruttare le comunicazioni bidirezionali con il client; in altre parole, è stata creata la WebSocket e il server è avviato.

7.2 Client

Il client, rispetto al server, presenta molte meno responsabilità. Infatti, esso non fa altro che inviare tramite la socket, in maniera continua e a gruppi di 10, i frame ricavati dallo streaming della webcam.



Figura 7.2: Client.

Come si vede dalla Figura 7.2, l'Interfaccia è costituita da una pagina HTML molto minimale. Essa consente solo di essere ripresi, sotto consenso dell'attivazione della webcam, senza alcuna possibilità di inserire input di qualsiasi genere.

Prima di iniziare a inviare la sequenza di frame al server, bisogna stabilire la connessione con quest'ultimo attraverso la socket. Il codice per fare ciò è molto semplice ed è riportato nelle seguenti linee:

```

1 <script src="//cdnjs.cloudflare.com/ajax/libs/socket.io
2   /2.2.0/socket.io.js"></script>
3 <script type="text/javascript" charset="utf-8">
4   var socket = io();
5   socket.on("connect", function() {
6     socket.emit("my event", {data: "I'm connected!"});
7   });
8 </script>

```

Listing 7.2: Connessione tra client e server attraverso la socket.

Quando si utilizza SocketIO, i messaggi vengono ricevuti da entrambe le parti come eventi. Infatti, alla riga 4 il client rimane in ascolto sull'evento *connect* e appena ottiene la connessione con il server, invia, attraverso la funzione *emit()*, un messaggio in risposta (riga 5).

Una volta stabilità la connessione, il client deve campionare, in maniera continua, lo streaming video in buffer di 10 frame e inviarli al server per effettuare l'inferenza. Il primo passo, dunque, è il campionamento:

```

1 const video = document.querySelector("#videoElement");
2 ...
3 const FPS = 25;
4 var seq = [];
5 setInterval(() => {
6   const canvas = document.createElement("canvas");
7   canvas.getContext("2d").drawImage(video, 0, 0);
8   const data = canvas.toDataURL("image/jpeg");
9
10  seq.push(data);
11  if (seq.length == 10){
12    socket.emit("image", seq);
13    seq.shift();
14  }
15 }, 10000/FPS);

```

Listing 7.3: Campionamento continuo di 10 frame dallo streaming video.

Una volta recuperato l'oggetto video nella pagina HTML (riga 1), si inizializza alla riga 4 il buffer che conterrà i frame. Siccome scambiare delle immagini tra client e server non è una cosa così immediata come potrebbe

essere un file JSON, alla riga 8 l'immagine viene convertita in una stringa. Alla riga 10 il buffer è riempito con il frame appena convertito. Alla riga 11 si controlla che il buffer abbia lunghezza 10 e alla riga 12 il client lo invia al server. Infine, alla riga 13, il buffer viene shiftato verso sinistra al fine di liberare uno slot per il prossimo frame.

Inviata la sequenza di 10 frame al server, esso deve effettuare l'inferenza e inviare al client la gloss predetta:

```

1 @socketio.on('image')
2 def image(data_image):
3     # define empty frame sequence
4     sequence_frame = np.empty((0, 224, 224, 3))
5     # unpack request
6     for data in data_image:
7         # decode and convert the string into an image
8         frame = Image.open(io.BytesIO(base64.b64decode(data
9             [23:])))
10        . . .
11        . . .
12    # predict the gloss from model
13    output = model.predict(sequence_frame)[0]
14    predict_ind = np.argmax(output)
15    # check if the probability exceeds the threshold
16    if output[predict_ind] > threshold:
17        word = class_text[predict_ind]
18        text_show = word
19    else:
20        text_show = 'none'
21    # return the gloss to the client
22    label = text_show.upper()
23    emit('response_back', label)

```

Listing 7.4: Inferenza del server.

Il server, prima di tutto, effettua il decode dei frame inviati come stringhe (righe 6-8). Ottenuto i frame nel giusto formato, si procede, alla riga 12, ad effettuare la prediction sfruttando il modello precedentemente allenato. Alla riga 15 viene stabilito cosa ritornare in base alla probabilità ottenuta: se la probabilità supera la soglia stabilita allora viene ritornata la gloss predetta,

altrimenti viene ritornata la stringa ‘none’. Infine, alla riga 22, il server invia un messaggio al client contenente la label corretta.

A questo punto al client non resta che intercettare la risposta del server restando in ascolto sull’evento *response_back* (riga 2):

```

1 var sentence = "";
2 socket.on('response_back', function(label){
3     if (!sentence.includes(label) && label != "NONE") {
4         sentence += " - " + label;
5         document.getElementById("label").innerHTML = sentence
6         .substring(3, sentence.length);
7     }
7 });

```

Listing 7.5: Ricezione lato client della risposta del server.

Infine, alle righe 4 e 5 viene aggiornata la frase da mostrare nella pagina web. Si noti che in questo caso la frase non è altro che la sequenza di parole, segnate dalla persona, una di seguito all’altra. Infatti, come detto all’inizio, il problema è stato affrontato focalizzandosi sulle singole parole e non su intere frasi.

Per avere un quadro completo dell’intero processo, si propone il seguente schema grafico:

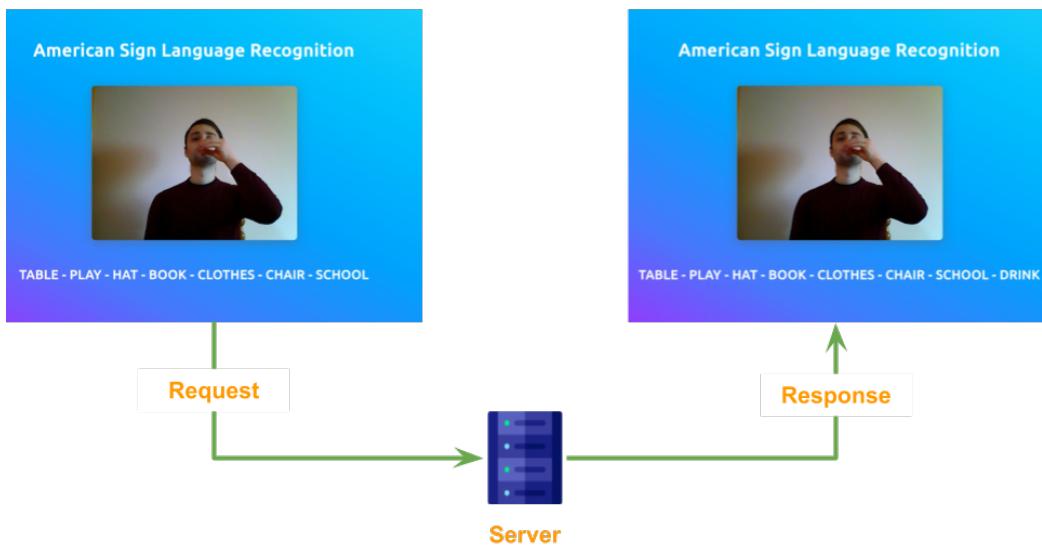


Figura 7.3: Esempio pratico della web app.

Come si vede dalla Figura 7.3, durante il processo di inferenza, si riportano tutte le gloss segnate l’una di seguito all’altra. In questo caso specifico, nello screen di sinistra si sta iniziando a segnare il gesto che indica la parola drink; a questo punto buffers di 10 frame sono inviati al server. Terminata l’inferenza, il server ritorna la gloss drink al client, al quale non resta che mettere in coda alle altre parole quella appena ricevuta (screen sulla destra).

Come accennato in precedenza, tra la fase di segnatura e inferenza ci potrebbe essere un minimo di delay, dato che l’inferenza, seppur breve, implica del tempo che non può essere risparmiato. Questo però non può essere definito come un “fallimento” dell’applicazione, perché, nella realtà, ci si aspetta che la traduzione compaia solo qualche istante dopo l’inzio della segnatura.

Infine, si è preferito lasciare le varie parole segnate in sequenza, sotto lo streaming video, proprio per sottolineare che la traduzione di intere frasi non è poi una cosa così lontana dalla realtà.

Capitolo 8

Conclusioni

In questo studio di tesi è stato proposto un applicativo real-time client-server in grado di riconoscere e tradurre la lingua dei segni americana a livello di parola nel corrispettivo testo scritto.

Attraverso i risultati riportati si è visto come l'approccio proposto ha superato le performance del modello presentato in [25], sottolineandone la fattibilità. Inoltre, l'implementazione dell'applicativo client-server non ha fatto altro che aumentare le aspettative generali, facendo volgere lo sguardo verso una traduzione a livello di frase.

Sebbene, ora come ora, lo studio effettuato in questa tesi non abbia dei diretti sbocchi in ambito pratico, esso vuole fungere da punto di partenza per gettare le basi per un sistema molto più completo.

Investire dell'effort in questo tipo di problematiche non è una cosa scontata e richiede molto studio e ricerca. Nonostante tutto, i benefici che le persone non udenti, ma anche quelle udenti, potrebbero ricavarne sarebbero molteplici:

- migliorare l'interazione tra persone udenti e non udenti in qualsiasi luogo. Infatti, basterebbe che singola una telecamera sia presente a riprendere i gesti del segnante;
- l'integrazione di tali sistemi in applicazioni di video conferenza porterebbe ad un maggior coinvolgimento delle persone non udenti;

- inserendo tali funzionalità in assistenti virtuali o di tipo robotici, le persone non udenti potrebbero essere in grado di relazionarsi anche in contesti non del tutto usuali.

Riportate qui sopra, sono solo una piccola parte delle problematiche che un sistema di traduzione real-time della lingua dei segni potrebbe risolvere.

Quindi, in maniera automatica, quello che ci si aspetta dal futuro è il raggiungimento della possibilità di riuscire a tradurre, in maniera real-time e senza la necessità di particolari strumenti, intere frasi e dialoghi segnati da persone non udenti.

Bibliografia

- [1] Anna Lisa Antonucci. *Nel mondo 72 milioni di persone usano la lingua dei segni*. URL: https://www.ansa.it/sito/notizie/politica/2018/09/10/nel-mondo-72-milioni-di-persone-usano-la-lingua-dei-segni_eb1551e4-f5f1-4e1d-9383-2aed0663601e.html. (accessed: 29-12-2020).
- [2] Vassilis Athitsos et al. “The American Sign Language Lexicon Video Dataset”. In: *2012 IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops* 0 (giu. 2008), pp. 1–8. DOI: [10.1109/CVPRW.2008.4563181](https://doi.org/10.1109/CVPRW.2008.4563181).
- [3] Y. Bengio, P. Simard e P. Frasconi. “Learning long-term dependencies with gradient descent is difficult”. In: *IEEE Transactions on Neural Networks* 5.2 (1994), pp. 157–166. DOI: [10.1109/72.279181](https://doi.org/10.1109/72.279181).
- [4] Helene Brashear et al. “Using multiple sensors for mobile sign language recognition”. In: nov. 2005, pp. 45–52. ISBN: 0-7695-2034-0. DOI: [10.1109/ISWC.2003.1241392](https://doi.org/10.1109/ISWC.2003.1241392).
- [5] C. Charayaphan e A.E. Marble. “Image processing system for interpreting motion in American Sign Language”. In: *Journal of Biomedical Engineering* 14.5 (1992), pp. 419–425. ISSN: 0141-5425. DOI: [https://doi.org/10.1016/0141-5425\(92\)90088-3](https://doi.org/10.1016/0141-5425(92)90088-3). URL: <http://www.sciencedirect.com/science/article/pii/0141542592900883>.
- [6] Kyunghyun Cho et al. “Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation”. In: (giu. 2014). DOI: [10.3115/v1/D14-1179](https://doi.org/10.3115/v1/D14-1179).

- [7] François Chollet. *Keras: Deep Learning for Python*. URL: <https://keras.io/>. (accessed: 30-12-2020).
- [8] Jia Deng et al. “Imagenet: A large-scale hierarchical image database”. In: *In CVPR*. 2009.
- [9] S. Sidney Fels e Geoffrey E. Hinton. *Glove-Talk: A neural network interface between a data-glove and a speech synthesizer*. 1993.
- [10] Django Software Foundation. *Django*. URL: <https://flask.palletsprojects.com/en/1.1.x/>. (accessed: 29-12-2020).
- [11] A. Géron. “Avoiding Overfitting Through Regularization”. In: *Hands-On Machine Learning with Scikit-Learn & TensorFlow*. O'Reilly, 2017, pp. 302–310.
- [12] A. Géron. “Convolutional Neural Networks”. In: *Hands-On Machine Learning with Scikit-Learn & TensorFlow*. O'Reilly, 2017. Cap. 13, pp. 353–378.
- [13] Glottolog. *Sign Language: American Sign Language*. URL: <https://glottolog.org/resource/languoid/id/amer1248>. (accessed: 29-12-2020).
- [14] Google. *Colaboratory*. URL: <https://colab.research.google.com/>. (accessed: 30-12-2020).
- [15] Gary J. Grimes. “Digital data entry glove interface device”. In: nov. 1983.
- [16] Miguel Grinberg. *Flask-SocketIO*. URL: <https://flask-socketio.readthedocs.io/en/latest/>. (accessed: 29-12-2020).
- [17] Kaiming He et al. *Deep Residual Learning for Image Recognition*. 2015. arXiv: 1512.03385 [cs.CV].
- [18] Sepp Hochreiter e Jürgen Schmidhuber. “Long Short-Term Memory”. In: *Neural Comput.* 9.8 (nov. 1997), pp. 1735–1780. ISSN: 0899-7667. DOI: 10.1162/neco.1997.9.8.1735. URL: <https://doi.org/10.1162/neco.1997.9.8.1735>.

- [19] Jie Huang et al. “Sign Language Recognition using 3D convolutional neural networks”. In: giu. 2015, pp. 1–6. DOI: 10.1109/ICME.2015.7177428.
- [20] Hamid Reza Jozé e Oscar Koller. *MS-ASL: A Large-Scale Data Set and Benchmark for Understanding American Sign Language*. 2019. arXiv: 1812.01053 [cs.CV].
- [21] Andrej Karpathy. *The Unreasonable Effectiveness of Recurrent Neural Networks*. URL: <http://karpathy.github.io/2015/05/21/rnn-effectiveness/>. (accessed: 16-01-2021).
- [22] Keras. *TimeDistributed layer*. URL: https://keras.io/api/layers/recurrent_layers/time_distributed/. (accessed: 30-12-2020).
- [23] Alina Kuznetsova, Laura Leal-Taixé e Bodo Rosenhahn. “Real-Time Sign Language Recognition Using a Consumer Depth Camera”. In: *Proceedings of the 2013 IEEE International Conference on Computer Vision Workshops*. ICCVW ’13. USA: IEEE Computer Society, 2013, pp. 83–90. ISBN: 9781479930227. DOI: 10.1109/ICCVW.2013.18. URL: <https://doi.org/10.1109/ICCVW.2013.18>.
- [24] Yann LeCun et al. “Object Recognition with Gradient-Based Learning”. In: *Shape, Contour and Grouping in Computer Vision*. Berlin, Heidelberg: Springer Berlin Heidelberg, 1999, pp. 319–345. ISBN: 978-3-540-46805-9. DOI: 10.1007/3-540-46805-6_19. URL: https://doi.org/10.1007/3-540-46805-6_19.
- [25] D. Li et al. “Word-level Deep Sign Language Recognition from Video: A New Large-scale Dataset and Methods Comparison”. In: *2020 IEEE Winter Conference on Applications of Computer Vision (WACV)*. 2020, pp. 1448–1458. DOI: 10.1109/WACV45572.2020.9093512
.
- [26] Wei Liu et al. “SSD: Single Shot MultiBox Detector”. In: vol. 9905. Ott. 2016, pp. 21–37. ISBN: 978-3-319-46447-3. DOI: 10.1007/978-3-319-46448-0_2.

- [27] A. M. Martinez et al. “Purdue RVL-SLLL ASL database for automatic recognition of American Sign Language”. In: *Proceedings. Fourth IEEE International Conference on Multimodal Interfaces*. 2002, pp. 167–172. DOI: 10.1109/ICMI.2002.1166987.
- [28] Metal3d. *Keras Sequence Video generators*. URL: <https://pypi.org/project/keras-video-generators/>. (accessed: 30-12-2020).
- [29] National Institute on Deafness and Other Communication Disorders - NIDCD. *American Sign Language*. URL: <https://www.nidcd.nih.gov/health/american-sign-language>. (accessed: 29-12-2020).
- [30] Yali Nie. *A Multi-stage Convolution Machine with Scaling and Dilation for Human Pose Estimation*. URL: https://www.researchgate.net/figure/Layers-and-their-abstraction-in-deep-learning-Image-recognition-as-measured-by-ImageNet_fig17_326531654. (accessed: 16-01-2021).
- [31] Christopher Olah. *Understanding LSTM Networks*. URL: <http://colah.github.io/posts/2015-08-Understanding-LSTMs/>. (accessed: 16-01-2021).
- [32] Pallets. *Flask web development, one drop at a time*. URL: <https://flask.palletsprojects.com/en/1.1.x/>. (accessed: 29-12-2020).
- [33] Lutz Prechelt. “Early Stopping - But When?” In: *Neural Networks: Tricks of the Trade*. A cura di Genevieve B. Orr e Klaus-Robert Müller. Berlin, Heidelberg: Springer Berlin Heidelberg, 1998, pp. 55–69. ISBN: 978-3-540-49430-0. DOI: 10.1007/3-540-49430-8_3. URL: https://doi.org/10.1007/3-540-49430-8_3.
- [34] Razieh Rastgoo, Kourosh Kiani e Sergio Escalera. “Video-based isolated hand sign language recognition using a deep cascaded model”. In: *Multimedia Tools and Applications* 79 (ago. 2020). DOI: 10.1007/s11042-020-09048-5.
- [35] Joseph Redmon e Ali Farhadi. “YOLOv3: An Incremental Improvement”. In: (apr. 2018).

- [36] D. E. Rumelhart, G. E. Hinton e R. J. Williams. “Learning Internal Representations by Error Propagation”. In: *Parallel Distributed Processing – Explorations in the Microstructure of Cognition*. MIT Press, 1986. Cap. 8, pp. 318–362.
- [37] M. Sandler et al. “MobileNetV2: Inverted Residuals and Linear Bottlenecks”. In: *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2018, pp. 4510–4520. DOI: 10.1109/CVPR.2018.00474.
- [38] Mark Sandler e Andrew Howard. *MobileNetV2: The Next Generation of On-Device Computer Vision Networks*. URL: <https://ai.googleblog.com/2018/04/mobilenetv2-next-generation-of-on.html>. (accessed: 16-01-2021).
- [39] Florian Schroff, Dmitry Kalenichenko e James Philbin. “FaceNet: A unified embedding for face recognition and clustering”. In: *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (giu. 2015). DOI: 10.1109/cvpr.2015.7298682. URL: <http://dx.doi.org/10.1109/CVPR.2015.7298682>.
- [40] K. Simonyan e A. Zisserman. *Very Deep Convolutional Networks for Large-Scale Image Recognition*. 2015. arXiv: 1409.1556 [cs.CV].
- [41] Christian Szegedy et al. *Rethinking the Inception Architecture for Computer Vision*. 2015. arXiv: 1512.00567 [cs.CV].
- [42] Shinichi Tamura e Shingo Kawasaki. “Recognition of sign language motion images”. In: *Pattern Recognition* 21.4 (1988), pp. 343–353. ISSN: 0031-3203. DOI: [https://doi.org/10.1016/0031-3203\(88\)90048-9](https://doi.org/10.1016/0031-3203(88)90048-9). URL: <http://www.sciencedirect.com/science/article/pii/0031320388900489>.
- [43] Kristen Tcherneshoff. *Answering Your Most Common Questions About Sign Languages*. URL: <https://medium.com/wikitongues/answering-your-most-common-questions-about-sign-languages-d206bb06853c>. (accessed: 29-12-2020).

- [44] D. Uebersax et al. “Real-time sign language letter and word recognition from depth data”. In: *2011 IEEE International Conference on Computer Vision Workshops (ICCV Workshops)*. 2011, pp. 383–390. DOI: 10.1109/ICCVW.2011.6130267.
- [45] Robert Y. Wang e Jovan Popović. “Real-Time Hand-Tracking with a Color Glove”. In: *ACM Trans. Graph.* 28.3 (lug. 2009). ISSN: 0730-0301. DOI: 10.1145/1531326.1531369. URL: <https://doi.org/10.1145/1531326.1531369>.
- [46] Wikipedia contributors. *List of sign languages — Wikipedia, The Free Encyclopedia*. [Online; accessed 29-December-2020]. 2020. URL: https://en.wikipedia.org/w/index.php?title=List_of_sign_languages&oldid=996341280.
- [47] Zahoor Zafrulla et al. “American Sign Language Recognition with the Kinect”. In: *Proceedings of the 13th International Conference on Multi-modal Interfaces*. ICMI ’11. Alicante, Spain: Association for Computing Machinery, 2011, pp. 279–286. ISBN: 9781450306416. DOI: 10.1145/2070481.2070532. URL: <https://doi.org/10.1145/2070481.2070532>.
- [48] Morteza Zahedi et al. “Combination of Tangent Distance and an Image Distortion Model for Appearance-Based Sign Language Recognition”. In: *Pattern Recognition*. A cura di Walter G. Kropatsch, Robert Sablatnig e Allan Hanbury. Berlin, Heidelberg: Springer Berlin Heidelberg, 2005, pp. 401–408. ISBN: 978-3-540-31942-9.
- [49] Fuzhen Zhuang et al. *A Comprehensive Survey on Transfer Learning*. 2020. arXiv: 1911.02685 [cs.LG].