Università degli Studi di Padova

Department of Information Engineering

MASTER THESIS IN COMPUTER ENGINEERING

A new scope for Open Cognition project: task and motion planning through its Neural-Symbolic Knowledge Store

Supervisor
EMANUELE MENEGATTI
Università di Padova

Co-supervisor
ENRICO PAGELLO
ELISA TOSELLO
Università di Padova

Master Candidate
Michele THIELLA

Date
.. October 2021

Academic Year 2020 - 2021

Abstract

Questo progetto propone diversi approcci per il riconoscimento dei gesti delle mani partendo da immagini e dati 3D che sfruttano tecniche di deep learning. L'algoritmo proposto inizia analizzando diversi tipi di dati: mappe di profondità, mappe di confidenza e mappe di colore (immagini RGB o Greyscale). Quindi i dati di input vengono inviati ad una Convolutional Neural Network, che per alcuni approcci è multi-branch mentre in altri single-branch. Ogni ramo della rete prende in input una delle mappe e produce un vettore di classificazione usando 4 strati convoluzionali di risoluzione progressivamente ridotta. Infine i vari vettori di classificazione passano attraverso un classificatore lineare che combina le uscite dei vari rami (nei casi multi-branch) e produce la classificazione finale. In conclusione vengono analizzati i risultati confrontando i vari approcci e le configurazioni della rete e dei dati di input.

Contents

List of Figures

List of Tables

Road to agi.

1 Introduzione

Il recupero e la classificazione di forme 3D sono stati oggetto di una grande quantità di lavori di ricerca che sfruttano sia la rappresentazione globale che i descrittori di forma locale. Un panoramica del campo può essere trovata in [1], [2], [3] o guardando le varie edizioni del SHREC 3D Retrieval Contest [4].

Questo progetto si focalizza sul riconoscimento di gesti delle mani, ambito di notevole interesse dovuto alle sue applicazioni in molti campi diversi come l'interazione uomo-computer, la robotica, i giochi per computer e l'interpretazione automatica della lingua dei segni. I recenti conseguimenti nel campo della Computer Vision hanno permesso di ottenere un miglioramento notevole in algoritmi di questo tipo. In particolare ci sono stati due avanzamenti chiave:

- Sviluppo di algoritmi di machine learning più potenti, specialmente tecniche di deep learning.
- Introduzione di telecamere di profondità, come Time-Of-Flight e Structured-light cameras [5].

La soluzione del problema di classificazione di forme 3D usando dati di immagini e video è sempre stata molto impegnativa, ma lo sviluppo di questi due campi ha permesso, sia di capire meglio il contenuto semantico delle immagini, che di utilizzare approcci basati su informazioni tridimensionali. Ad esempio i dati di profondità che si riescono ad ottenere da queste telecamere contengono una descrizione molto accurata della posa della mano e questo

è molto utile per le applicazioni di riconoscimento dei gesti. Il progetto si basa sullo studio di 5 approcci differenti che combinano diverse architetture della rete e vari tipi di dati di input. Infatti non ci si sofferma solo su dati di profondità ma si utilizzano anche mappe di confidenza e immagini RGB. L'unione di queste informazioni porta ai risultati presentati consultabili nelle tabelle alla fine del Capitolo **??**.

Questa tesi è composta da una parte introduttiva riguardante le Deep Neural Networks e i suoi algoritmi di ottimizzazione, una parte centrale che illustra il set di dati utilizzato e le sue elaborazioni, successivamente vengono spiegate diverse architetture della Convolutional Neural Network utilizzata per il riconoscimento di 11 gesti differenti, ed infine una parte conclusiva di analisi dei risultati.

2 Deep Neural Network

In questo progetto viene utilizzata Convolutional Neural Network (CNN). Una descrizione completa delle NN si può trovare in [6], [7, pp. 509–542], [8, pp. 37–121]. Le CNN verranno spiegate brevemente in seguito ma per maggiori dettagli si consulti [6, pp. 326–365], [9, pp. 106–119].

Invece ci si vuole soffermare sulla spiegazione delle ottimizzazioni che vengono applicate alla rete. I due algoritmi principali sfruttati sono: l'algoritmo di Backpropagation e lo Stochastic Gradien Descent.

2.1 Stochastic Gradient Descent

Si prende X per denotare un insieme di dati di training. Sarà conveniente considerare ogni input di training $\mathbf{x_i} \in X$ come un immagine e quindi con un vettore bidimensionale. L'output desiderato corrispondente è $\mathbf{y} = y(x)$, con $\mathbf{y}$ che invece rappresenta un vettore monodimensionale, i cui valori sono associati ciascuno ad ogni dato di input di training $\mathbf{x_i}$. Si cerca un algoritmo che ci permetta di trovare pesi e bias in modo che l'output dalla rete si avvicini a $y(x)$ per tutti gli input di training $\mathbf{x_i}$. Per quantificare quanto bene si sta raggiungendo questo obiettivo viene definita una funzione di costo:

$$C(w, b) \equiv \frac{1}{2n} \sum_x \|y(x) - a\|^2. \tag{2.1}$$

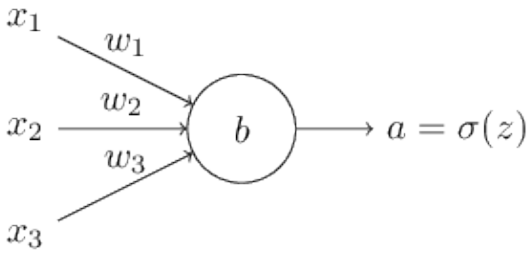

Figure 2.1: Rappresentazione di un neurone di bias b con tre variabili di input x_i, ciascuna avente un peso w_i e con output $\sigma(z)$.

Qui, **w** denota la raccolta di tutti i pesi nella rete, **b** tutti i bias, n è il numero totale di input di training, **a** è il vettore di uscite dalla rete quando X è input, e la somma è su tutti gli input di training $\mathbf{x_i}$. Ovviamente, l'output **a** dipende da x, w e b, ma per mantenere la notazione semplice non viene indicata esplicitamente questa dipendenza. La notazione $||\mathbf{v}||$ denota semplicemente la norma di un vettore **v**.

É chiamata C la funzione di costo quadratico. Ispezionando la sua forma si vede che $C(w, b)$ è non negativo, poiché ogni termine della somma è non negativo. Inoltre, il costo $C(w, b)$ diventa piccolo, cioè $C(w, b) \approx 0$, precisamente quando $y(x)$ è approssimativamente uguale all'uscita, a, per tutti gli input di training, x. Quindi l'algoritmo di training ha fatto un buon lavoro se è in grado di trovare pesi e bias in modo che $C(w, b) \approx 0$. Al contrario, non sta andando così bene quando $C(w, b)$ è grande, questo significherebbe che $y(x)$ non è vicino all'uscita a per un gran numero di ingressi. L'obiettivo dell'algoritmo di training sarà ridurre al minimo il costo $C(w, b)$ in funzione dei pesi e dei bias. Precisamente, si cercherà di trovare una serie di pesi e bias che rendano il costo il più piccolo possibile. E questo sarà possibile grazie ad un algoritmo noto come discesa del gradiente (Gradient Descent). La funzione però riscontra dei problemi riguardanti la velocità di apprendimento, ma si è voluto introdurla perchè, utilizzando una funzione di costo uniforme come quella quadratica, risulta facile capire come apportare piccole modifiche nei pesi e nei bias in modo da ottenere un miglioramento nella funzione di costo. Quindi viene scelta una funzione di costo diversa, nota come Cross-Entropy.

Si suppone di addestrare un neurone con diverse variabili di input, $x1$, $x2$,... , corrispondenti ai pesi $w1$, $w2$,... e un bias, b. Un esempio è dato in Figura **2.1**.

L'uscita dal neurone è, naturalmente, $a = \sigma(z)$, dove $z = \sum_j (w_j * x_j) + b$ è la somma ponderata degli input, mentre una descrizione completa della funzione di attivazione σ la si può trovare in [8, pp. 60–64].

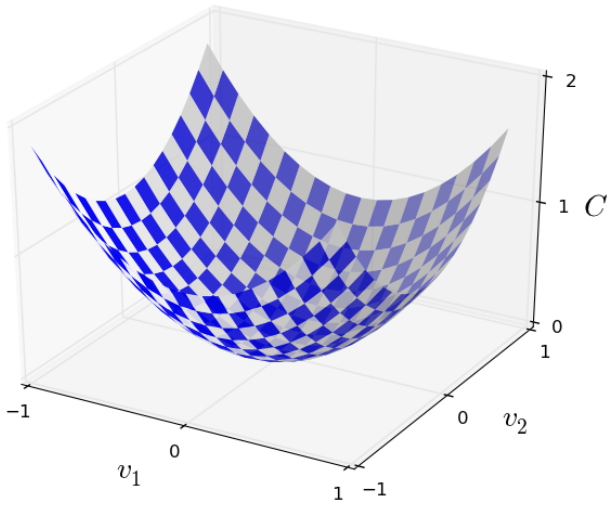

Figure 2.2: Funzione tridimensionale a forma di "valle".

Si definisce la funzione di costo Cross-Entropy per questo neurone come:

$$C = -\frac{1}{n}\sum_{x}[y \ln a + (1-y)\ln(1-a)] \tag{2.2}$$

Dove n è il numero totale di dati di training, la somma è fatta su tutti i file di training, $\mathbf{x_i}$, e $\mathbf{y}$ è l'output desiderato corrispondente.

Minimizzare la funzione è un problema noto, ma non è di facile risoluzione in quanto ci sono molti fattori da considerare come la ricerca dei valori di w e b che minimizzano la funzione, il σ funzione in background, la scelta dell'architettura di rete, i dati di input e così via. Il Gradient Descent è l'algoritmo che permetterà questa minimizzazione.

Nelle reti neurali spesso si hanno molte variabili, le reti neurali più grandi hanno funzioni di costo che dipendono da miliardi di pesi e bias in un modo estremamente complicato quindi usare il calcolo analitico per minimizzare non funzionerà. Viene sfruttata un'idea che sarà solo accennata in questa descrizione, vengono citati [6] e [8, pp. 91–94] per approfondimenti sull' argomento. Si supponga di voler minimizzare la funzione in Figura **2.2**.

Uno dei metodi più banali per risolvere il problema è appoggiarsi ad una interpretazione

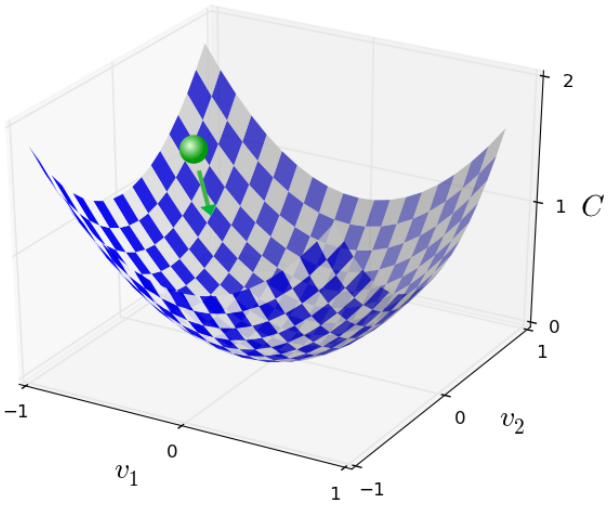

Figure 2.3: Viene sfruttata l'idea fisica tale per cui se si lascia andare una palla da un punto della funzione, essa rotolerà verso il fondo e si fermerà solo quando avrà trovato il punto di minimo.

fisica. Si immagini di prendere una sfera e di posizionarla in un punto qualasiasi della funzione, lasciandola liberà di muoversi come in Figura **2.3**. Man mano che essa farà uno spostamento verso il fondo della funzione, diminuirà anche la funzione di costo. Quindi, il modo in cui l'algoritmo di discesa del gradiente funziona consiste nel calcolare ripetutamente il gradiente ΔC, e quindi spostarsi nella direzione della pendenza della valle fino a raggiungere il punto minimo. L'algoritmo non funziona sempre, diverse cose possono andare storte e impedire alla discesa del gradiente di trovare il minimo globale di C. Ma, in pratica, la discesa del gradiente spesso funziona molto bene, e nelle reti neurali è un modo efficace per ridurre al minimo la funzione di costo, aiutando così la rete ad apprendere.

2.2 Algoritmo di Backpropagation

La Generalized Delta Rule [10], nota anche come algoritmo di Backpropagation, viene introdotta per le Feed-Forward Neural Networks.

In queste reti neurali le informazioni si muovono solo in una direzione, avanti, rispetto ai nodi d'ingresso, attraverso nodi nascosti fino ai nodi d'uscita. Nella rete non ci sono cicli. Le reti feed-forward non hanno memoria di input avvenuti a tempi precedenti, per cui l'output

è determinato solamente dall'attuale input.

L'addestramento di une rete neurale con backpropagation avviene in due diversi stadi: forward-pass e backward-pass. Nella prima fase i vettori in input sono applicati ai nodi in ingresso con una propagazione in avanti dei segnali attraverso ciascun livello della rete (forward-pass). Durante questa fase i valori dei pesi dei neuroni sono tutti fissati. Nella seconda fase la risposta della rete viene confrontata con l'uscita desiderata ottenendo il segnale d'errore. L'errore calcolato è propagato nella direzione inversa rispetto a quella delle connessioni tra i neuroni. I pesi infine sono modificati in modo da minimizzare la differenza tra l'uscita attuale e l'uscita desiderata (backward-pass).

La spiegazione seguente è destinata a fornire una descrizione del processo coinvolto nell' algoritmo di backpropagation usando, per semplicità, una rete neurale contente 3 livelli. Questi sono: livello di input, hidden e di output. Quindi, durante la fase di training (training phase), i dati di training (training data) vengono inseriti nel livello di input. I dati vengono propagati al livello hidden e quindi al livello di output. Questo è chiamato il passaggio in avanti dell'algoritmo di backpropagation (forward-pass). In questo passaggio, ogni nodo nel livello hidden ottiene input da tutti i nodi del livello di input, i quali vengono moltiplicati con pesi appropriati e quindi sommati. L'output del nodo hidden è la trasformazione non lineare della somma risultante. Allo stesso modo ogni nodo nel livello di output riceve input da tutti i nodi del livello hidden, che vengono moltiplicati con appropriati pesi e poi sommati. Nuovamente l'output di questo nodo è la trasformazione non lineare della somma risultante. I valori di uscita del layer di output vengono confrontati con i valori di output di target, che sono quelli che si cercano di insegnare alla rete. L'errore tra i valori di output effettivi e i valori di output di target viene calcolato e propagato di nuovo verso il livello hidden. Questo è chiamato backward-pass della rete. L'errore viene utilizzato per migliorare la forza delle connessioni tra i nodi, ad es. le matrici di peso tra i livelli input-hidden e i livelli hidden-output vengono aggiornate.

Durante la fase di test, non avviene alcun apprendimento, ad esempio, le matrici di peso non cambiano. Ogni vettore di prova viene inserito nel livello di input. L'avanzamento (feed-forward) dei dati di test è simile a quello dei dati nella fase di training ma termina con i valori di uscita del layer di output. Un breve riassunto viene mostrato in Figura **2.4**

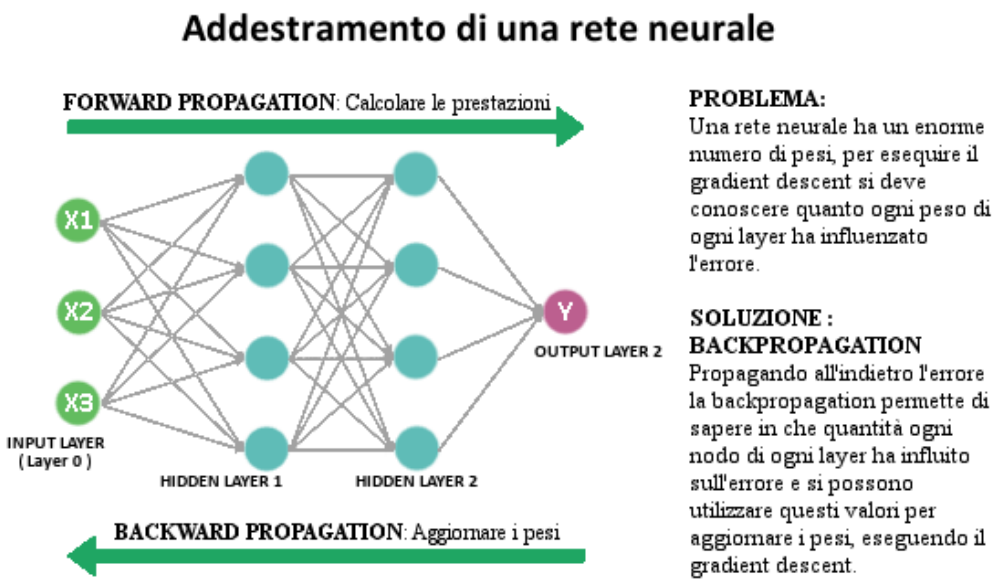

Figure 2.4: Schema riassuntivo del funzionamento dell'algoritmo di Backpropagation in una rete neurale feed-forward.

2.3 Convolution Neural Network

Infine per la conclusione del capitolo si vuole lasciare una descrizione delle Convolutional Neural Networks (CNN) usate per questo progetto. Il loro nome deriva da una delle operazioni più importanti della rete: la convoluzione. Le CNN sono un tipo di rete neurale artificiale feed-forward in cui il pattern di connettività tra i suoi neuroni è ispirato dall'organizzazione della corteccia visiva animale.

La ricerca svolta negli anni '50 e '60 di D.H Hubel e T.N Wiesel [11] ha suggerito un nuovo modello per descrivere come i mammiferi percepiscono visivamente il mondo. Hanno mostrato che le cortecce visive di gatto e scimmia includono neuroni che rispondono esclusivamente ai neuroni nel loro ambiente diretto. Nel loro articolo spiegano due tipi fondamentali di cellule del neurone visivo nel cervello che agiscono in modo diverso: cellule semplici (cellule S) e cellule complesse (cellule C). Le celle semplici si attivano, ad esempio, quando identificano le forme base come linee in un'area fissa o un angolo specifico. Le celle complesse hanno campi recettivi più grandi e il loro output non è sensibile alla posizione specifica nel campo. Le cellule complesse continuano a rispondere a un determinato stimolo, anche se la loro posizione assoluta sulla retina cambia. Inoltre, il concetto di gerarchia gioca un ruolo significativo nel cervello. Le informazioni sono memorizzate come modelli, in ordine sequen-

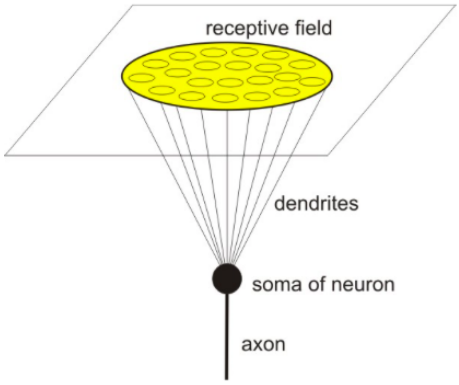

Figure 2.5: Il Campo Recettivo di un neurone - Sorgente: http://neuroclusterbrain.com/neuron_model.html feed-forward.

ziale. La neocorteccia, che è lo strato più esterno del cervello, immagazzina le informazioni gerarchicamente.

Nel 1980, Fukushima propose un modello gerarchico di rete neurale chiamato NeoCognitron [12]. Questo modello fu ispirato dai concetti delle cellule semplici e complesse ed è stato in grado di riconoscere i modelli apprendendo le forme degli oggetti.

Più tardi, nel 1998, le CNN furono introdotte in un articolo di Bengio, Le Cun, Bottou e Haffner [13]. La loro prima Convolutional Neural Network venne chiamata LeNet-5 ed era in grado di classificare cifre da numeri scritti a mano.

Tutti i metodi di riconoscimento degli oggetti all'avanguardia progettano le loro architetture di modelli attorno ad un concetto chiave per le CNN, il campo recettivo (Receptive Field). Il campo recettivo è definito come la regione nello spazio di input influenzata da una particolare caratteristica di una CNN (Figura **2.5**). Un campo recettivo di una caratteristica può essere descritto dalla sua posizione centrale e dalle sue dimensioni. Tuttavia, non tutti i pixel in un campo sono ugualmente importanti per la caratteristica corrispondente della CNN. All'interno di un campo recettivo i pesi di ogni pixel vengono appresi dalla rete nella fase di training. In principio potrebbe apprendere che i pixel più lontani pesino di più anche se di solito, come in questo caso, più vicino è un pixel al centro del campo, più contribuisce al calcolo della funzione di output. Ciò significa che una caratteristica non guarda solo una particolare regione (cioè il suo receptive field) nell'immagine di input, ma si concentra anche verso la parte centrale di essa.

In una CNN il campo recettivo può essere aumentato con metodi diversi, ad esempio: impilando più strati (profondità), Pooling (sottocampionamento), dilatazione del filtro, ecc.

Tuttavia, ad esempio, impilando più strati è possibile aumentare linearmente il campo recettivo, ma in pratica non è così semplice in quanto aumenterebbe di conseguenza la complessità della rete e la difficoltà di allenarla, come dimostrato da Luo, Wenjie in [14]. La struttura delle CNN sarà spiegata in seguito nel Capitolo **2.4**. Per ulteriori studi si rimanda a [6, pp. 326–366] e [15].

2.4 Struttura della rete neurale

La struttura e i moduli di base di una rete neurale convoluzionale sono: la convoluzione, il Pooling (Subsampling), la normalizzazione, il Fully Connected Layer e lo strato di apprendimento.

2.4.1 Il processo di convoluzione

caratteristico di questo tipo di rete neurale, è ispirato dai processi biologici per l'analisi visiva negli organismi viventi. Lo strato di neuroni che si occupa della convoluzione divide l'immagine in vari frammenti sovrapposti, che sono in seguito analizzati per individuare le particolarità che lo caratterizzano, trasferendo l'informazione allo strato seguente sottoforma di una "feature map" contenente le relazioni tra neuroni e particolarità.

2.4.2 Pooling

è un processo alquanto comune nelle reti neurali, che consiste nel generalizzare i dati per ridurne la dimensione, in modo da rendere più rapida l'analisi senza perdere troppa precisione. Nel caso di un'immagine, il processo è molto simile ad un sottocampionamento: una regione di pixel diventa un unico pixel al quale è assegnato un colore in base al massimo valore presente nella sua regione. (Figura **2.6**).

2.4.3 La normalizzazione

è anch'essa largamente utilizzata per evitare le anomalie in seguito ai vari passaggi negli strati della rete neurale. Le funzioni più efficenti sono la Rectified Linear Units (ReLU), la tangente iperbolica e la fuzione sigmoide, anche se la ReLU è la migliore poichè più rapida ed evita il Vanishing Gradients. Viene rimandata la spiegazione a [6] e [17].

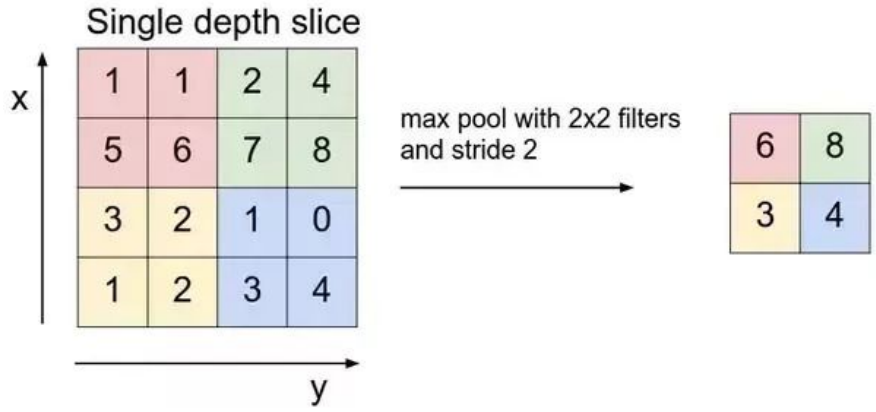

Figure 2.6: Il Max Pooling prende il valore più grande - Sorgente: [16].

2.4.4 Il Fully Connected Layer

nelle CNN rappresenta il vettore di caratteristiche per l'input. Questo strato di caratteristiche contiene informazioni molto utili per l'input. Quando la rete viene addestrata, esso viene utilizzato ad esempio per la classificazione, la regressione o l'input in altre reti, ecc. Durante la fase di allenamento, il fully connected layer determina la perdita e aiuta la rete a migliorare. I livelli di convoluzione prima di esso contengono informazioni sulle caratteristiche locali dell'immagine di input quali bordi, blob, forme, ecc. Ogni strato di convoluzione contiene diversi filtri che rappresentano una delle caratteristiche locali. In questo progetto il fully connected layer è l'ultimo strato della rete e contiene le informazioni composte e aggregate da tutti i gli strati di convoluzione.

2.4.5 Backpropagation

Infine, per le reti neurali c'è l'algoritmo di ottimizzazione che permette al sistema di modificare i pesi delle connessioni tra i neuroni sulla base della correttezza dei risultati emessi. Il metodo utilizzato è la Backpropagation spiegata nella sezione precedente (**2.2**).

3

Elaborazione dei dati di input

Per lo svolgimento del lavoro proposto viene utilizzato un set di dati costruito tramite un sensore 3D time-of-flight (noto anche come SoftKinetic DepthSense 325). Senz3D è un prodotto realizzato in partnership con Creative, una videocamera con sensore di profondità che può essere installata su notebook e PC basati su architettura Intel e che, di fatto, opera come una sorta di Kinect per dispositivi Windows. Il dispositivo ha sia una fotocamera RGB che un sensore di profondità Time-Of-Flight (ToF). La risoluzione di profondità è 320 per 240 pixel, con un frame rate compreso tra 6 e 60. La funzione automatica di soglia della confidenza del sensore è stata utilizzata anche per ottenere una mappa di profondità iniziale migliore.

Il set di dati contiene gesti prodotti da quattro persone diverse, ciascuna eseguendo undici diversi gesti ripetuti trenta volte ciascuno, per un totale di 1320 campioni. Sono usati i dati delle prime tre persone per la fase di allenamento della rete neurale (quindi 990 campioni). Mentre i restanti 330, relativi agli undici gesti dell'ultima persona, vengono utilizzati per la fase di test. Per ogni campione si hanno tre tipi di file. Una mappa di profondità (DEPTH), una mappa di confidenza (CONF) e un immagine che sarà utilizzata sia in formato di scala di grigi (GREY-SCALE) che in formato RGB. I gesti, numerati da 1 a 11, sono illustrati in Figura **3.1**. Si può notare come il set di dati sia molto variabile e naturale in quanto contiene gesti diversi con lo stesso numero di dita alzate, gesti con dita molto vicine tra di loro e con le punte delle dita che si toccano. Il dataset è disponibile all'indirizzo http://lttm.dei.unipd.it/downloads/gesture2.

Figure 3.1: Rappresentazione degli 11 gesti prendendo per ciascuno una delle 30 immagini RGB disponibili nel dataset relative alla prima persona.

Il lavoro proposto si compone di una fase iniziale di elaborazione dei dati di input. Ogni file viene letto e risalvato in formato NPY, poichè la rete neurale accetta in ingresso dati di questo formato. Un file NPY è un file di array NumPy creato dal pacchetto software di Python tramite la libreria NumPy. Esso contiene un array salvato nel formato di file NumPy (NPY), inoltre i file NPY memorizzano tutte le informazioni necessarie per ricostruire un array su qualsiasi computer, che include le informazioni su dtype e forma. Inizialmente è creato un ulteriore array di target che serve durante la fase di allenamento, ad esso, per ogni file NPY, sarà aggiunto il valore da 1 a 11 corrispondente al gesto che viene rappresentato.

Inoltre qui avviene la prima elaborazione dei dati di input, i quali vengono "puliti", riscalati e filtrati. Non ci si è soffermati sulla pulizia che consiste in una semplice rimozione dello sfondo, ad esempio tutti i campioni che rappresentano lo sfondo dell'immagine vengono impostati a zero mentre si lasciano invariati i numeri che descrivono il gesto della mano. Per la mappa di profondità viene utilizzato un filtro mediano. Questa funzione prende la mediana di tutti i pixel sotto l'area del kernel e l'elemento centrale viene sostituito dalla mediana nella finestra. Nella sfocatura mediana, l'elemento centrale viene sempre sostituito da qualche valore di pixel nell'immagine, in tal modo riesce a ridurre efficacemente il rumore impulsivo. Sono illustrati in Figura **3.2** alcuni gesti presi dai file in formato NPY e che quindi hanno già

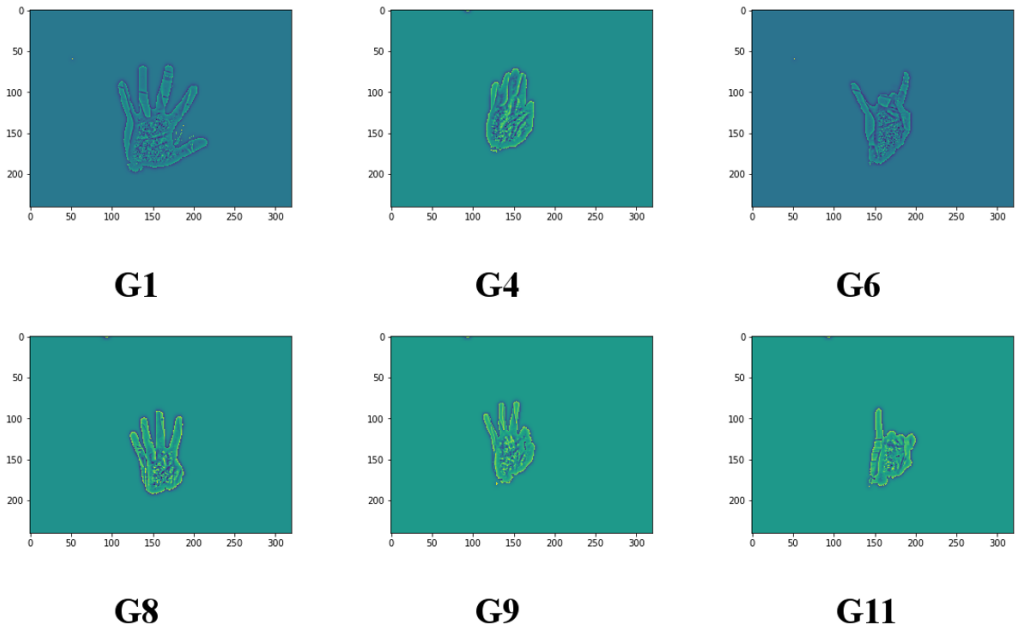

Figure 3.2: Rappresentazione grafica di vettori NPY in 2D di alcuni gesti, poco prima di essere elaborati dalla rete neurale.

subito tutto il processo di elaborazione (nell'immagine essi sono già stati anche normalizzati).

Questi file NPY successivamente vengono inviati alla rete neurale convoluzionale. La rete prende in input i file e produce un vettore di classificazione usando quattro strati convoluzionali di risoluzione progressivamente ridotta. I vari vettori di classificazione sono finalmente elaborati da un classificatore lineare che produce l'output finale. Questo è il funzionamento in generale, successivamente sarà spiegato nello specifico in quanto ci sono alcune variazioni in base all'approccio che viene scelto. Per la conclusione del paragrafo è mostrato un frammento del codice riguardante il filtraggio, la scalatura e la pulizia dei file (per quest'ultima è stato scelto un valore di soglia fisso di 400).

Codice: Esempio di elaborazione dei file di input.

```
depth=np.load(os.path.join(inpt_dir,'trainsource', id+'.npy'))
conf=np.load(os.path.join(inpt_dir,'trainsourceConf', id+'.npy'))
img=np.load(os.path.join(inpt_dir,'trainsourceColorRGB', id+'.npy'))

depth=cv2.medianBlur(depth,5)
```

```
imgR=pc.resize(img[:,:,0],(240,320),'linear').astype('float32')
imgG=pc.resize(img[:,:,1],(240,320),'linear').astype('float32')
imgB=pc.resize(img[:,:,2],(240,320),'linear').astype('float32')

for k in range(len(conf[0,:,:])):
    for q in range(len(conf[0,k,:])):
        if(conf[0,k,q] < 400):
         conf[0,k,q]= 0
            depth[0,k,q]=0
            conf[0,k,q]=0
            imgR[k,q]=0
            imgG[k,q]=0
            imgB[k,q]=0
```

4

Approcci proposti

La rete è riassunta nella Figura **4.1** ed è composta inizialmente da quattro strati convoluzionali (CONV), ciascuno avente dei filtri, una funzione di normalizzazione (ReLU) ed una funzione di pooling (MaxPOOL). Successivamente c'è una funzione di concatenazione, nel caso si usi un approccio con più canali contemporaneamente ed infine il Fully Connected Layer (LIN). Il reshape (RESHP) è un ri-arrangiamento dei vettori e viene usato per adattare la struttura dell'uscita di un livello all'ingresso del successivo. Infine si ha un classificatore Softmax (SOFT) seguido dal nodo AMAX, il quale sceglie la classe da ritornare basandosi su quella con probabilità più alta (l'output è descritto mediante una probabilità per ogni gesto).

I livelli vengono collegati secondo configurazioni diverse in base all'approccio seguito. In questo progetto di ricerca sono proposti cinque approcci differenti. In tal modo è stato possibile avere confronti tra di essi per un risultato finale più completo.

4.1 Approcci proposti

Gli approcci si differenziano in base ai dati di input utilizzati per ognuno di essi e qundi, di conseguenza, alle varie configurazioni della rete e del numero di canali. La Tabella **4.1** schematizza i vari approcci secondo i tipi di dati di input usati.

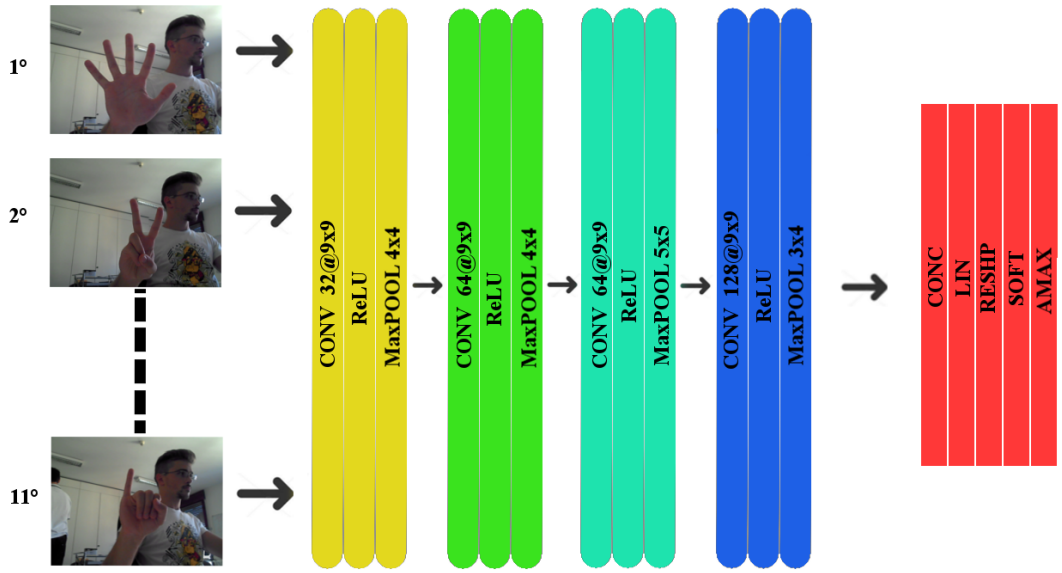

Figure 4.1: Si hanno in ingresso le immagini, elaborate precedentemente, relative agli unidici gesti. Esse passano attraverso quattro layer di convoluzione, in cui sono applicate le funzioni di filtraggio, di normalizzazione e di pooling. Infine, tramite le ultime funzioni vengono classificate.

Table 4.1: Tabella Approcci

	Approcci						
Dati di Input	**1°**	**1°**	**1°**	**2°**	**3°**	**4°**	**5°**
Depth	✓			✓		✓	✓
Conf		✓		✓		✓	✓
Color - R					✓	✓	✓
Color - G					✓	✓	✓
Color - B					✓	✓	✓
Grey-Scale			✓	✓			

4.1.1 Approccio unidimensionale 1 canale

Il 1° approccio e più semplice, viene ulteriormente suddiviso in tre parti in base ai dati di input scelti ed è caratterizzato da un unico canale unidimensionale. In questo approccio è applicato lo studio su dati di profondità (DEPTH), dati di confidenza (CONF) proporzionali all'ampiezza del segnale di uscita ToF del sensore di profondità e dati basati sul colore in scala di grigi (GREYSCALE). Ogni sottoapproccio viene eseguito in modo separato, cioè con cicli distinti di training/testing della rete. Quindi vengono riportati i risultati per ognuna delle tre categorie in modo indipendente una dell'altra.

4.1.2 Approccio unidimensionale 3 canali: Depth, Conf, Greyscale

Il 2° approccio utilizza le stesse tipologie di dati del primo. Però, invece di avere un unico canale e di fare uno studio indipendente per ogni classe di dati, sono creati 3 canali unidimensionali tramite un'architettura multi-branch (multicanale). Viene assegnato un canale per ogni tipologia di dati DEPTH, CONF e GREYSCALE. L'input della rete è la combinazione dei 3 canali in un unico ciclo di training e testing. Riguardo questo tipo di architettura si rimanda all'articolo [18] e alle sua bibliografia per una spiegazione dettagliata.

4.1.3 Approccio unidimensionale 3 canali: Color RGB

Ha la stessa struttura del secondo approccio. L'unica differenza è la tipologia di dati di input scelta. Ora lo studio è fatto su dati basati sul colore che seguono il modello RGB. Quindi è assegnato un canale unidimensionale per ciascuno dei tre colori (Rosso-Verde-Blu). Le restanti caratteristiche di questo 3° approccio coincidono con l'approccio precedente.

4.1.4 Approccio unidimensionale 5 canali: Depth, Conf, Color RGB

Composto da una combinazione del secondo e del terzo, in quanto il numero di canali unidimensionali è aumentato a cinque e sono assegnati ad essi i dati di input di profondità (DEPTH), di confidenza (CONF) e di colore RGB (quindi 3 canali unidimensionali per il colore). Anch'esso crea una combinazione di questi canali come input per cercare di migliorare l'apprendimento basandosi su più fattori contemporaneamente.

4.1.5 Approccio pentadimensionale 1 canale: Depth, Conf, Color RGB

Infine l'ultimo approccio riprende quasi tutti gli aspetti del quarto ma con una eccezione. Invece di avere canali unidimensionali per ognuna delle cinque tipologie di dati (DEPTH, CONF, RGB), si utilizza un unico canale multi-dimensionale (penta-dimensionale). Questo metodo comporta delle differenze sui dati i quali vengono combinati dall'inizio (rispetto al 4° approccio in cui questo avviene nell'ultimo layer). Quindi gli algoritmi che vengono utilizzati non saranno più eseguiti in modo distinto per ogni canale per poi unire i risultati successivamente, ma saranno applicati a tutti i dati insieme in quanto essi sono raggruppati in un canale unico. Non è scontato che questo approccio migliori l'apprendimento rispetto a quello precedente. Sicuramente si potranno avere situazioni in cui è preferibile usare questo approccio ed altri in cui funzionerà meglio un approccio unidimensionale.

In dettaglio, il primo approccio è quello base che utilizza un unico canale unidimensionale. In questo caso la rete neurale è anch'essa unidimensionale composta da quattro blocchi principali ciascuno contenente tre strati (Figura **4.1**). Ogni blocco è composto da uno strato convoluzionale (CONV) seguito da una funzione di normalizzazione (ReLU) e completato da un layer Max-Pooling finale (MaxPOOL) mentre il classificatore pixel-wise Softmax è applicato sopra l'ultimo strato convoluzionale. Per ridurre il tempo di calcolo, le immagini di input hanno una risoluzione ridotta di 320x240. Gli strati convoluzionali hanno rispettivamente 32, 64, 64, 128 filtri dal primo al quarto, tutti di dimensione 9x9 pixel, mentre il classificatore Softmax ha una matrice di peso della dimensione 128x11 e nessun bias. Viene applicata, la normalizzazione del contrasto locale a ciascun canale di input indipendentemente, consentendo ai pesi del filtro nel primo strato convoluzionale di convergere più velocemente. Nel primo approccio non viene usata la funzione di concatenazione (CONC). Negli approcci successivi la costruzione della rete neurale è simile ma si hanno più canali di input contemporaneamente e quindi un'architettura multi-branch. Per ognuno di essi, la rete ha i quattro blocchi principali con tutte le relative funzione spiegate sopra. In questi casi viene poi applicata la funzione di concatenazione e restano invariate le funzioni successive. L'eccezione si trova nel quinto approccio in cui si torna a lavorare con un solo canale, ma stavolta multi-dimensionale. Viene mostrato un pezzo del codice per l'assegnazione del vettore penta-dimensionale contenente i dati di DEPTH, CONF, RGB. La composizione della rete neurale è invariata rispetto ai casi del primo approccio.

Codice: Assegnazione del vettore di input pentadimensionale nel 5° approccio.

```
inpt=np.zeros(shape=(5,240,320),dtype=np.float32)

inpt1=pc.resize(depth[0,:,:],(240, 320),'linear').astype('float32')
inpt1=pc.normalize(inpt1,'local',1.8,1.8)
inpt[0,:,:]=inpt1

inpt2=pc.resize(conf[0,:,:],(240, 320),'linear').astype('float32')
inpt2=pc.normalize(inpt2, 'local',1.8,1.8)
inpt2=np.expand_dims(inpt2, axis=0)
inpt[1,:,:]= inpt2

img=pc.resize(img[:,:,:],(240,320,3),'linear').astype('float32')
inpt3=pc.normalize(img,'local',1.8,1.8)

inpt[2,:,:]=inpt3[:,:,2]
inpt[3,:,:]=inpt3[:,:,1]
inpt[4,:,:]=inpt3[:,:,0]
```

5 Conclusione

5.1 Conclusione

In questo progetto è stata proposta un'architettura di rete profonda per il riconoscimento di gesti della mano da dati 3D. Sono stati usati diversi approcci, che progressivamente analizzavano i dati a risoluzione sempre inferiore fino ad ottenere un singolo risultato di classificazione per ciascuno. Per lo studio di tale problema le possibilità erano molte quindi si è voluto mirare ad avere un quadro complessimo più ampio, in modo da poter confrontare molti risultati. Cercando combinazioni differenti tra architetture single-branch e multi-branch e mescolando in vari modi i dati di input a disposizione si è ottenuta la Tabella **??**. Essa contiene un riassunto incrociato tra gli unidici gesti e i vari approcci utilizzati, così da ottenere un'ampia visione complessiva sia per ogni gesto in relazione a tutti gli approcci che per ogni approccio in funzione di tutti i gesti. Tutti gli approcci hanno ottenuto buoni risultati, partendo dal 4° e dal 5° che combinando più dati sono riusciti ad ottenere più informazioni ed imparare meglio. Fino al 1° approccio che, nonostante utilizzasse solo le depth map, ha comunque raggiunto l'85%. Infine si vuole dare importanza anche ai risultati dei rimanenti approcci, essi potrebbero dimostrare la loro efficacia in altre situazioni in cui, ad esempio, i dati di input siano molto diversi rispetto a quelli contenuti in questo dataset.

Ci sono ancora molti aspetti da analizzare meglio, ad esempio nel modo di pulire i dati di input cambiando da una soglia fissa ad una variabile cosicchè si possa ottenere una cancel-

lazione dello sfondo più accurata. Oppure nella costruzione della rete neurale e nell'inizializzazione dei suoi parametri. Aggiungendo o rimuovendo layer, cambiando le dimensioni dei filtri, ecc. Ma, nonostante i problemi sopracitati, si può vedere come la rete abbia ottenuto buoni risultati pur in mancanza di ottimizzazioni molto più accurate.

Bibliography

[1] Johan W. H. Tangelder and Remco C. Veltkamp. "A survey of content based 3D shape retrieval methods". In: *Multimedia Tools and Applications* 39.3 (Dec. 2007), pp. 441–471. DOI: 10.1007/s11042-007-0181-0. URL: https://doi.org/10.1007/s11042-007-0181-0.

[2] Yulan Guo et al. "3D Object Recognition in Cluttered Scenes with Local Surface Features: A Survey". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 36.11 (Nov. 2014), pp. 2270–2287. DOI: 10.1109/tpami.2014.2316828. URL: https://doi.org/10.1109/tpami.2014.2316828.

[3] Bo Li et al. "A comparison of 3D shape retrieval methods based on a large-scale benchmark supporting multimodal queries". In: *Computer vision and image understanding* 131 (2015), pp. 1–27. ISSN: 1077-3142. DOI: 10.1016/j.cviu.2014.10.006.

[4] *SHREC 3D Retrieval Contest*. http://cs231n.github.io/convolutional-networks/. Accessed: 13-9-2018.

[5] Carlo Dal Mutto, Pietro Zanuttigh, and Guido M Cortelazzo. *Time-of-Flight Cameras and Microsoft Kinect™*. Springer US, 2012. DOI: 10.1007/978-1-4614-3807-6. URL: https://doi.org/10.1007/978-1-4614-3807-6.

[6] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. URL: http://www.deeplearningbook.org.

[7] Paul V. Biron. "Backpropagation: Theory, Architectures, and Applications, edited by Yves Chauvin and David E. Rumelhart." In: *JASIS* 48.1 (1997). URL: http://dblp.uni-trier.de/db/journals/jasis/jasis48.html#Biron97.

[8] David Kriesel. *A Brief Introduction to Neural Networks*. 2007. URL: available%20at%20http://www.dkriesel.com.

[9] Hamed Habibi Aghdam and Elnaz Jahani Heravi. *Guide to Convolutional Neural Networks*. Springer International Publishing, 2017, pp. 106–119. ISBN: 978-3-319-57550-6. DOI: 10.1007/978-3-319-57550-6. URL: https://doi.org/10.1007/978-3-319-57550-6.

[10] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. *Leaning Internal Representaions by Ewvr Propagation in Rumelhart, D. E. and McClelland, J. L., Pamllel Distributed Processing: Explorations in the Microstructure of Cognition*. Cambridge Massachusette: MIT Press, 1986.

[11] D. H. Hubel and T. N. Wiesel. "Receptive fields and functional architecture of monkey striate cortex". In: *The Journal of Physiology* 195.1 (Mar. 1968), pp. 215–243. DOI: 10.1113/jphysiol.1968.sp008455. URL: https://doi.org/10.1113/jphysiol.1968.sp008455.

[12] Kunihiko Fukushima. "Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position". In: *Biological Cybernetics* 36.4 (Apr. 1980), pp. 193–202. DOI: 10.1007/bf00344251. URL: https://doi.org/10.1007/bf00344251.

[13] Yann Lecun et al. "Gradient-based learning applied to document recognition". In: *Proceedings of the IEEE*. 1998, pp. 2278–2324.

[14] Wenjie Luo et al. "Understanding the Effective Receptive Field in Deep Convolutional Neural Networks". In: *Advances in Neural Information Processing Systems 29*. Ed. by D. D. Lee et al. Curran Associates, Inc., 2016. URL: http://papers.nips.cc/paper/6203-understanding-the-effective-receptive-field-in-deep-convolutional-neural-networks.pdf.

[15] Vincent Dumoulin and Francesco Visin. "A guide to convolution arithmetic for deep learning". In: *CoRR* abs/1603.07285 (2016).

[16] *CS231n Convolutional Neural Networks for Visual Recognition*. http://cs231n.github.io/convolutional-networks/.

[17] George E. Dahl, Tara N. Sainath, and Geoffrey E. Hinton. "Improving deep neural networks for LVCSR using rectified linear units and dropout". In: *2013 IEEE International Conference on Acoustics, Speech and Signal Processing*. IEEE, May 2013. DOI: 10.1109/icassp.2013.6639346. URL: https://doi.org/10.1109/icassp.2013.6639346.

[18] Hongyang Zhang, Junru Shao, and Ruslan Salakhutdinov. "Deep Neural Networks with Multi-Branch Architectures Are Less Non-Convex". In: *CoRR* abs/1806.01845 (2018).

[19] *ANKIVIL Machine Learning Experiments - Making Theano Faster with CuDNN and CNMeM on Windows 10*. https://ankivil.com/making-theano-faster-with-cudnn-and-cnmem-on-windows-10/.