Riconoscimento delle parti della mano mediante deep learning

Favaro Marzia

Relatore: Zanuttigh Pietro

Correlatore: Michieli Umberto

Corso di laurea in Ingegneria dell'Informazione

15 Luglio 2019

Anno Accademico 2018-2019

Marzia Favaro Pietro Zanuttigh Umberto Michieli

Abstract

Questa tesi pone le basi ad un più ampio progetto di riconoscimento gestuale per il miglioramento dell'interfaccia uomo-macchina. L'obiettivo finale è quello di interpretare le informazioni combinate di una videocamera e di un sensore di profondità e ricavarne la volontà dell'utente che comunica mediante gesti della mano.

Per migliorare la capacità di riconoscimento gestuale, si può affiancare al processo di identificazione l'apprendimento della disposizione delle diverse componenti della mano, al fine di aumentare la qualità della risposta della macchina. Si discuterà di una componente dell'apprendimento complessivo che consiste nella divisione di un'immagine in diverse regioni corrispondenti alle differenti aree della mano. Per lo scopo verrà utilizzata una rete neurale ottimizzata per il problema specifico e qui ne verranno descritte le motivazioni e la costruzione.

I dati impiegati per l'allenamento saranno sintetici, generati automaticamente come perturbazioni dei gesti di interesse, per creare un set di immagini allo stesso tempo robusto e facile da generare. Questi saranno l'input fornito ad una U-Net con metrica di apprendimento cross entropy dice loss e ottimizzatore Adam.

1. Introduzione

Il progetto di segmentazione di una mano vede il suo fine in un più ampio contesto di riconoscimento gestuale, mirato al miglioramento dell'interazione uomo-macchina. Questo obiettivo richiede di passare attraverso l'identificazione delle diverse componenti della mano, per generare un input più stabile da fornire alla rete neurale per l'identificazione del gesto. La tesi si occuperà di classificare i pixel delle immagini fornite per distinguere le diverse componenti della mano, creando una rete neurale che si potrà affiancare alla predizione del gesto.

Immagine rgb della mano | Depth map della mano | Immagine classificata

Dall'input, fornito tramite immagini RGB e depth map, si vuole passare ad un'immagine della mano partizionata nelle sue diverse componenti: le singole dita (distinte tra loro), il palmo, e il polso, separando il tutto dallo sfondo.

La trattazione sarà in primo luogo generale, mirata a descrivere il generico approccio ad un problema di segmentazione tramite reti neurali convoluzionali, e nella seconda parte si andranno a descrivere più nel dettaglio le scelte implementative specifiche del caso.

Si partirà con la discussione del problema della segmentazione semantica in generale, accennando agli specifici elementi che compongono le reti per poter discutere la scelta della tipologia della rete. Si dedicherà poi spazio alla discussione dei dati: le problematiche legate alla loro acquisizione e soluzioni alternative per una generazione

di grandi quantità di dati. Infine verrà la trattazione più dettagliata delle scelte effettuate per risolvere il problema specifico, e un commento sui risultati ottenuti.

Il progetto sarà quindi ottimizzato e pronto perchè alla rete così costruita si associ la componente di riconoscimento gestuale apportandone alcune modifiche.

2. Reti neurali per la segmentazione semantica

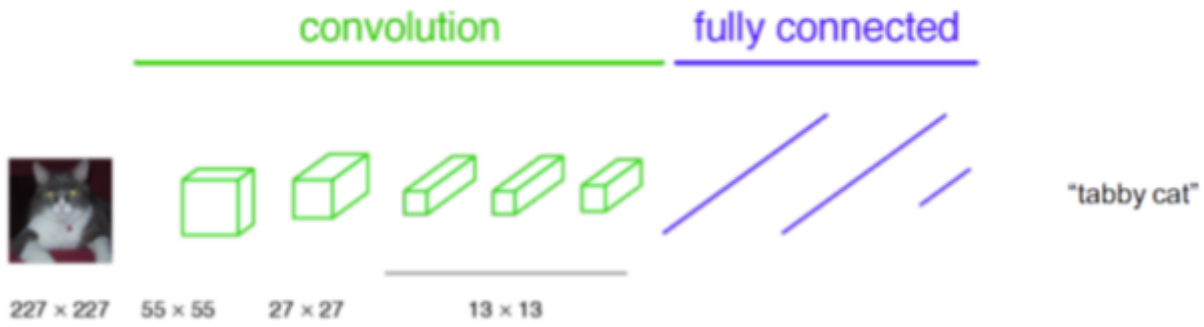
Il problema in questione rientra in ciò che in machine learning è definito *supervised learning*. Questo consiste nell'allenare una rete neurale fornendole campioni di input e corrispondenti output attesi fino a renderla in grado di lavorare con dati analoghi ma nuovi. La procedura viene effettuata in due fasi temporalmente distinte: la prima di allenamento, computazionalmente onerosa e in cui la rete viene regolata per essere in grado di risolvere il problema, e la seconda di *inference*, in cui data la rete allenata è possibile ottenere l'output desiderato.

2.1. Il problema della segmentazione semantica

La prima fase di definizione del problema consiste nella descrizione dei suoi input e output.

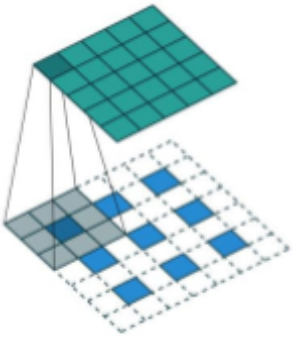
L'input della rete è stabilito: si tratta di un'immagine di una mano fornita mediante le componenti RGB e una depth map, di cui si discuterà più nel dettaglio nel seguito. L'output della rete può essere rappresentato in diversi modi, ma la scelta più naturale consiste nell'etichettare ogni singolo pixel e *classificarlo* assegnandogli un valore che porti con sé le informazioni desiderate. Si rientra quindi in un problema di segmentazione semantica[3].

In generale il problema di classificazione consiste nel ridurre un vettore di ingresso ad una informazione di dimensione minore (come uno scalare). Un esempio è il problema di classificazione di cifre scritte a mano: data un'immagine la rete deve essere in grado di riconoscere quale cifra rappresenta, fornendo un valore in uscita secondo una convenzione da stabilire (un intero, un vettore booleano...). Il ridimensionamento, e con esso il concentramento dell'informazione, avviene in diversi livelli consecutivi.



Un esempio di problema di classificazione

Questa procedura, tuttavia, non è adatta allo scopo qui trattato così come è stata presentata: infatti l'informazione a cui si vuol giungere non è di dimensione inferiore all'input e tantomeno scalare. Si può ricorrere quindi all'*upsampling*, che permette di ridistribuire l'informazione dopo averla concentrata.



Rappresentazione grafica di upsampling

Prima di trattare nello specifico la rete adottata, è necessario conoscere i blocchi e le operazioni di cui si compone, che verranno quindi brevemente esposti.

2.2. Componenti di una CNN

2.2.1. Convoluzione

L'input dell'operazione di convoluzione consiste in un tensore (un volume) di taglia $n_{in} * n_{in} * n_{channels}$. A compiere la manipolazione dell'input sono k filtri $f * f * n_{channels}$: la

convoluzione non avviene infatti su tutta la matrice nella sua interezza, ma avviene localmente in k regioni.

La funzione dei filtri, collezioni *kernel*, è quella di scorrere lungo la matrice in input, compiendo l'operazione di convoluzione e generando in output una matrice riportante una forma di compressione locale delle informazioni originali. Ogni kernel agisce in modo indipendente sul relativo canale dell'immagine e l'output del filtro è la combinazione di questi.

L'output che viene generato ha dimensione $n_{out} * n_{out} * k$, in cui n_{out} risulta $n_{out} = (n_{in} + 2p - k)/s + 1$, dove p rappresenta la dimensione di *padding* della convoluzione ed s lo *stride*. Il padding consiste nell'aggiungere un margine attorno all'immagine per non perdere l'informazione di bordo. Lo stride invece è l'indicatore di "velocità" di spostamento dei filtri, ossia di quanto si spostano ad ogni passaggio. Minore è lo stride, maggiore è la sovrapposizione dei filtri.

2.2.2. Max pooling

L'informazione, così come fornita in input, non è esaminabile direttamente, in quanto non è possibile avere una visione d'insieme di uno volume così ampio. L'idea di fondo è quella di ridurre le informazioni da analizzare, mantenendo solo le più importanti (nel caso del max pooling, i pixel con i valori massimi) per ogni regione.

Esempio di max pooling

Mediante il *pooling*, ad ogni livello il campo ricettivo dei filtri si allarga, in quanto l'immagine è sempre più concentrata in poco spazio e diventa analizzabile da filtri

sempre più piccoli. Grazie a questa procedura, quindi, è possibile analizzare l'immagine nella sua interezza, rendendo più chiaro *cosa* rappresenta, ma perdendo l'informazione sul *dove* l'informazione si trovasse rispetto all'input. Le procedure che vengono eseguite sono caratterizzate dall'invarianza alla traslazione delle componenti elementari di cui si compongono: non è la posizione assoluta a contare, ma quella relativa.

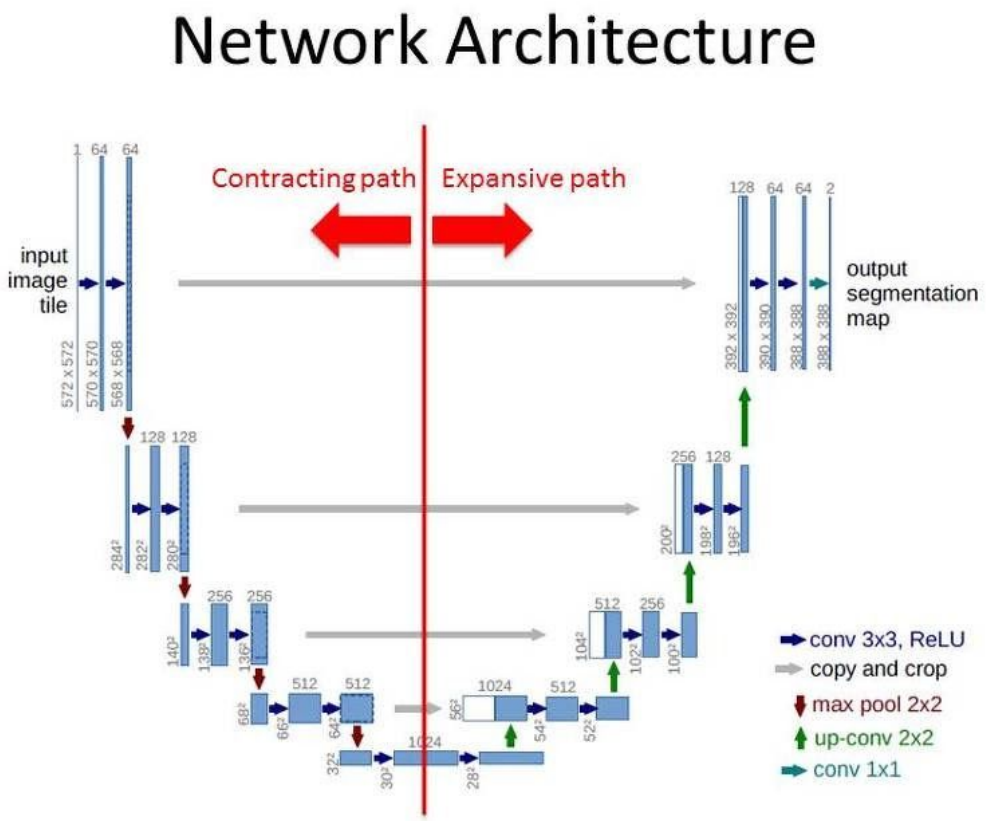
Si deve osservare come però, per il problema di segmentazione, l'uso dello stride e del pooling porti ad una riduzione della risoluzione spaziale. A questo scopo interviene l'*upsampling*, per invertire la procedura di condensamento e riportare i risultati alle coordinate originali. Una tecnica che discende naturalmente dalla convoluzione è la *deconvoluzione*, o *backwards convolution*.

2.2.3. Funzioni di attivazione

Per spiegare in cosa consistono i livelli di attivazione è necessario comprendere le componenti "atomiche" di ogni rete neurale: i *neuroni*, che saranno però discussi più nel dettaglio in seguito. Ogni neurone possiede un insieme di ingressi provenienti da altri neuroni, e un'uscita che a sua volta è passata in input ad altri neuroni. Questi hanno un livello di attivazione che definisce lo stato del neurone come acceso o spento dipendentemente dal tipo di funzione di attivazione che li caratterizza e dai suoi parametri. Mediante l'impostazione di questi ultimi, infatti, si determina il comportamento dei neuroni e quindi la risposta della rete agli input.

2.3. U-Net

La struttura che è stata scelta per la rete è una U-Net, sviluppata da Olaf Ronneberger[4] per l'analisi di immagini biomediche. La rete si presta bene al problema per via della sua struttura a encoder-decoder che permette prima di comprimere e successivamente espandere il tensore in ingresso per le finalità sopra citate.



2.3.1. Contrazione ed espansione[2]

La prima parte della rete ad essere attraversata è quella di contrazione. Qui 4 blocchi codificatori (*encoder*) si susseguono concatenati uno all'altro e trasformano i tensori diminuendone lunghezza e larghezza e aumentandone la profondità. Un encoder è costituito da diversi livelli, tra cui di convoluzione, attivazione e pooling.

Successivamente alla contrazione, dopo aver attraversato un nodo centrale, il tensore attraversa il secondo ramo della rete, in cui la sua dimensione viene ripristinata a quella iniziale passando attraverso i *decoder*.

La simmetria della rete rende superflua una discussione approfondita dei decoder. Tuttavia va osservato come questi ricevono in input non solo il tensore rappresentativo dell'immagine: infatti avviene anche una concatenazione tra encoder e decoder corrispondente (operanti allo stesso "livello"). Queste connessioni, *skip*

connections, aiutano ad inglobare anche l'informazione riguardante il contesto e a portare l'informazione a risoluzioni più elevate, aumentando la risoluzione dei contorni.

L'aspetto finale della U-net, ossia il *summary* del modello sviluppato, poiché troppo lungo per essere inserito qui, si trova su github[1].

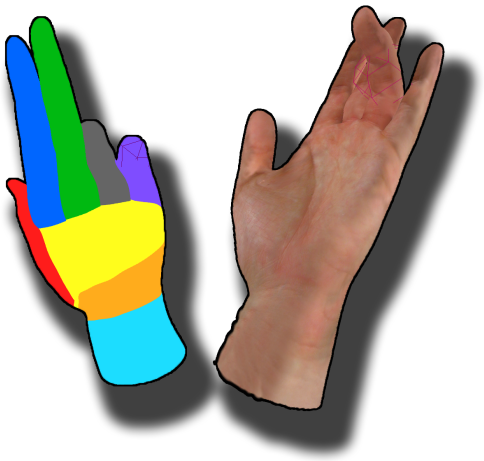
[1] https://bit.ly/2S5chUl

3. Dataset per gesture recognition

L'allenamento della rete richiede una grande mole di dati in input. Questo è un ostacolo, in quanto ad ogni immagine deve essere associata una maschera rappresentante le etichette non generabile in automatico (altrimenti si avrebbe già la soluzione della segmentazione e la costruzione della rete sarebbe inutile).

Il numero di dati necessari per il corretto allenamento è elevato (nel caso in questione nell'ordine delle migliaia), questo per evitare che la rete finisca in *overfitting* e per renderla flessibile ai diversi input. L'*overfitting* avviene quando i dati di allenamento della rete sono troppo pochi o troppo simili e quindi la rete impara a rispondere in modo pressoché perfetto a casi già visti, ma non è in grado di gestirne di nuovi.

Poiché non è possibile generare manualmente le maschere con le categorie per le migliaia di immagini necessarie, viene utilizzato un generatore sintetico[2][9]. Questo è in grado di applicare una texture ad una mano della quale si possono controllare i movimenti.



In questo modo si riescono a generare molti dati con uno sforzo umano indipendente dalla dimensione del set desiderato. Tuttavia questo metodo porta degli svantaggi: i

[2] http://lttm.dei.unipd.it/downloads/handposegenerator

dati sintetici sono privi di rumore, variazioni di luminosità, ombre e altri disturbi che invece sono presenti nelle immagini reali.

Altro aspetto piuttosto limitante, da non sottovalutare, è la dimensione dei dati generati. Qui si è utilizzato un set di 11 gesti, per ciascuno ci sono 200 immagini 256x256 sia RGB+depth (4 canali), sia di classificazione (ad un canale), per un totale di circa 5 GB. Seppur possa sembrare una quantità non troppo eccessiva, va tenuto conto che questa si deve sommare alla dimensione della rete e allo spazio utilizzato nei calcoli, e che queste dimensioni eccedono la normale capacità delle ram dei computer domestici. Questi problemi hanno contribuito in modo non indifferente a rallentare il progetto ed evidenziano la necessità di caricare porzioni più ridotte di dati nella ram.

3.1. Generazione dei dati sintetici

La generazione dei dati avviene a partire da gesti predefiniti, quelli di interesse per il problema finale di riconoscimento gestuale. Questi vengono perturbati casualmente per creare più varietà possibile di immagini. Ad ogni posizione della mano corrispondono tre immagini generate:

- Due immagini RGB: una a cui viene applicata la texture della pelle umana e una a cui corrispondono i colori rappresentanti le classi
- Un'immagine a un canale con l'informazione sulla distanza (la profondità)

Le immagini così generate, però, non sono ancora pronte per essere fornite alla rete: avviene ora la fase di assemblamento di RGB e depth in un unico tensore e di remapping del tensore delle classi in una matrice a valori interi.

3.2. Adattare i dati sintetici al modello reale

Ora i dati contengono le informazioni essenziali per il riconoscimento di gesti, ma presentano tutti delle caratteristiche comuni molto forti e innaturali.

Un problema subito evidente è l'orientamento: tutte le mani hanno il polso rivolto a sinistra. Per risolvere il problema è sufficiente applicare una rotazione casuale alle immagini, di un numero di gradi nel range 0-360 poiché sono tutte rotazioni plausibili. Oltretutto tutte le mani generate sono sinistre, quindi alcune saranno specchiate per diventare destre.

Anche la centralità dell'immagine è poco naturale, ma basta applicare piccole traslazioni per ovviare al problema.

Ci sono molte altre modifiche possibili per migliorare le immagini, come riscalarle casualmente, applicare filtri per modificare la luminosità, aggiungere rumore e disturbi, cambiare colori, applicare piccole distorsioni.

In questo caso è stata effettuata una procedura di pre-processing che comprende rotazioni, traslazioni e riflessioni.

3.3. La depth map

Ulteriore vantaggio dell'uso del generatore è la generazione contestuale delle *depth map* delle immagini, ossia la rappresentazione delle informazioni sulla tridimensionalità della mano mediante un'immagine ad un canale. Questa è un'informazione ulteriore che non viene fornita dalle normali fotocamere, ma che può essere generata comunque anche nella realtà mediante sensori appositi.

Un esempio di depth map

4. Allenamento della rete

È adesso possibile iniziare l'allenamento della rete e quindi ottenere i primi risultati.

4.1. Divisione dei dati

La rete necessita di tre diversi set di dati, che sono estratti casualmente tra quelli generati precedentemente, secondo proporzioni prestabilite.

Lo scopo dell'allenamento è impostare i parametri della rete affinché performi al meglio. Questi vengono impostati inviando l'input alla rete, che in output fornirà un tensore dipendente dagli stessi parametri, contenente la stima sull'assegnazione dei pixel alle varie parti della mano. Inizialmente questi assumono valori pseudo-casuali, che vengono aggiustati nella fase di *backpropagation* sfruttando l'informazione sul *ground truth* (il tensore delle classi) fornite insieme alle immagini di input. Il primo set, di *training*, (il più ampio, che rappresenta il 70% dei dati), viene impiegato a questo proposito.

Nella fase successiva vengono valutate le performance della rete con i nuovi parametri e a questo scopo si impiega il set di *validation*, (20% del set originario) sul quale si calcolano i parametri di misura dell'efficienza per monitorare l'apprendimento ed eventualmente fermarlo in caso di mancato miglioramento.

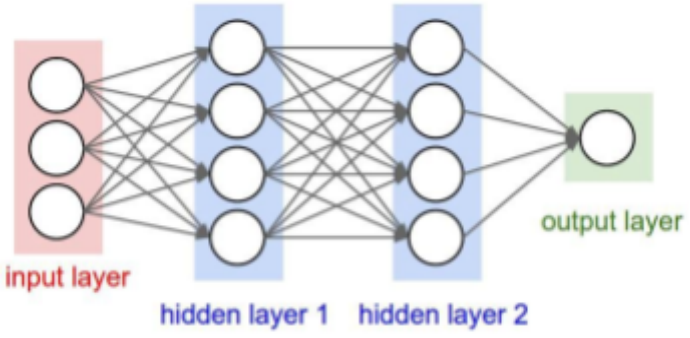
L'ultimo set, di *test* (il restante 10%), viene utilizzato per la verifica finale dei risultati dell'allenamento. È distinto dai precedenti per evitare che le stesse immagini su cui la rete è stata allenata vengano utilizzate per verificarne le prestazioni: sarà infatti normale che l'output corrispondente ad un'immagine di allenamento sia migliore di uno esterno.

4.2. Inizializzazione dei parametri e funzioni di attivazione

I *parametri* della rete sono valori associati ad ogni neurone di ogni livello. I neuroni sono le entità atomiche della rete e ne determinano il funzionamento. Ogni neurone ha

il compito di restituire un output fornito da una funzione che viene scelta in base alle esigenze.

Con *livello* si intende un insieme di neuroni che operano in parallelo. Ognuno di essi riceve gli stessi input (nel caso di una rete completamente connessa) e determina un output definito da una stessa *funzione di attivazione* che viene applicata ad una trasformazione dell'input, che viene prima pesato e a cui si somma un bias. Questi parametri di trasformazione sono dipendenti dal singolo neurone. La rete è composta di diversi livelli, collegati tra loro in quanto l'output del livello precedente è l'input del successivo, e che si dividono in tre categorie: *input*, *output* e *hidden*. Su questi ultimi c'è un maggior margine di libertà per quanto riguarda la scelta della dimensione (numero di neuroni) e della funzione di attivazione, mentre l'input e l'output sono vincolati dalle dimensioni dei tensori di ingresso e uscita e dalla loro tipologia; inoltre si trovano agli estremi della rete.



Rappresentazione grafica di una piccola rete neurale

La scelta della funzione di attivazione non è solo vincolata alle sue caratteristiche matematiche, ma anche all'efficienza della sua computazione: si deve ricordare infatti che il numero di neuroni è estremamente elevato, e di conseguenza anche i calcoli sono temporalmente dispendiosi (nella rete utilizzata per la soluzione di questo problema vengono impiegati 31 milioni di parametri da ottimizzare).

La facilità di calcolo e la rapidità nella convergenza sono la ragione per cui per negli hidden layers (che nella rete utilizzata in questo caso sono incapsulati in encoder e decoder) viene utilizzata la *ReLU*, *rectified linear unit*, con il comportamento da funzione identità per valori in ingresso positivi e che invece appiattisce a 0 i valori

negativi. Nella sua semplicità, non si potrebbe ricondurre a una funzione completamente lineare, che ridurrebbe le prestazioni. Nel livello di output invece viene impiegata la *softmax*, o esponenziale normalizzata. Questa viene impiegata nei problemi di classificazione non binari e il suo output rappresenta la probabilità di appartenenza alla classe. Grazie alla sua caratteristica esponenziale le differenze tra i valori di input vengono amplificate in uscita.

I neuroni sono quindi funzioni i cui parametri devono essere ottimizzati per riuscire ad ottenere l'output desiderato e che inizialmente vengono impostati a valori casuali tra 0 e 1.

4.3. Feed forward e Back-propagation

La *forward propagation* è così detta in quanto è la fase in cui i dati attraversano la rete nella sua direzione "naturale", ossia input-output.

Durante la forward propagation ogni unità (neurone) pesa l'input e vi aggiunge la componente di bias, per poi applicare al valore ottenuto la funzione di attivazione.

Avviene successivamente la fase di correzione degli errori per approssimazioni successive: la back-propagation. Interviene a questo proposito la *loss function*.

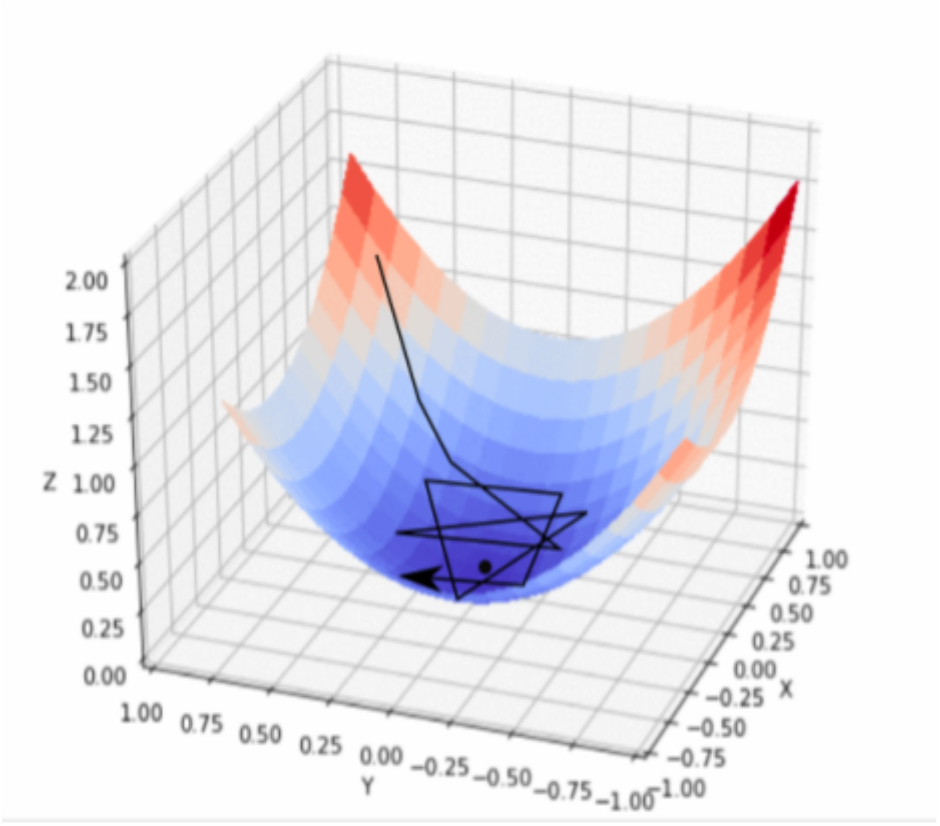
La loss function che è stata scelta in questo caso è la *Cross-Entropy dice loss*[7]. Questa deriva dalla *Cross Entropy* (CE) loss, funzione che misura le performance di una rete analizzando distribuzioni di probabilità ed è quindi adatta alla rete qui utilizzata, in quanto il livello di output utilizza la softmax activation function. Questa loss misura l'entropia mutua tra il risultato ottenuto e quello atteso. Quello che ci si aspetta è che il valore dell'entropia mutua decresca ad ogni epoca (ciclo di allenamento della rete) e che quindi i risultati ottenuti siano sempre più fedeli a quelli attesi. Tuttavia questa metrica potrebbe portare a risultati falsati in questo caso, essendo le classi sbilanciate: tutte le immagini presentano la classe "sfondo" dominante su altre classi come "mignolo", in quanto lo sfondo occuperà molti più pixel nell'immagine rispetto ad altre classi. Utilizzare la CE loss potrebbe portare quindi a risultati sbilanciati, perciò occorre un fattore di correzione, che viene introdotto dai

coefficienti della dice loss[5,6]. Naturalmente questa scelta ha un prezzo: così facendo il calcolo del gradiente si complica. Tuttavia come si discuterà analizzando i risultati, è importante non sottovalutare il problema dello sbilanciamento delle classi.

4.3.1. Gradient descent

A prescindere dalla scelta della funzione di loss, l'obiettivo sarà quello di minimizzarla per ridurre la discrepanza tra valore ottenuto e desiderato. Per far questo viene calcolato il gradiente della loss[1] (calcolato rispetto ai suoi parametri) al fine di avvicinarsi al minimo della funzione. Il metodo della discesa del gradiente consiste nello spostarsi nella curva (della funzione) lungo la direzione di decrescita e permette di evitare il calcolo della derivata globale della funzione per l'individuazione dei punti di minimo. Lo spostamento lungo la curva rappresenta la modifica dei parametri, che avviene in modo proporzionale alla derivata parziale della loss. Questo sistema presenta però degli svantaggi: non c'è garanzia di trovare il minimo globale, infatti con questo sistema una volta situati in un punto di minimo locale non è possibile uscirne e vi sono difficoltà anche nell'attraversamento di punti di sella.

La velocità di discesa può essere regolata mediante il *learning rate*, coefficiente che permette di velocizzare o rallentare l'apprendimento. Questo parametro ha anch'esso un suo peso e influisce sulle prestazioni della rete. Un learning rate alto aumenta la velocità di apprendimento, ma potrebbe mancare il minimo e creare oscillazioni intorno ad esso. Per evitare questo fenomeno si può gradualmente ridurre il learning rate durante l'apprendimento. Se il suo valore è basso invece si ottiene maggior precisione, ma il tempo per raggiungere il minimo è maggiore.



rappresentazione del percorso compiuto durante il gradient descent

La discesa del gradiente si rappresenta come $\omega \rightarrow \omega' - \eta \nabla L$ in cui ω rappresenta le coordinate nello spazio dei pesi, ω' la nuova posizione, L è la funzione da minimizzare (loss) e η il learning rate. Il segno meno è dovuto al fatto che la direzione di spostamento è quella indicata dall'opposto del gradiente, per opporsi alla direzione di massimo.

Nonostante il metodo della discesa del gradiente sia già una grande semplificazione rispetto al calcolo analitico dei punti di minimo, resta computazionalmente oneroso se il dataset è molto grande, perchè richiede l'analisi di tutti i campioni. Lo si può ancora ottimizzare con metodi di *stochastic gradient descent* (SGD). Quando la funzione L si può esprimere come $L(\omega) = 1/n \sum_{1}^{n} L_i$, allora il gradiente si approssima con il gradiente calcolato su un singolo addendo $\omega \rightarrow \omega' - \eta \nabla L_i$.

Vi sono alcune estensioni di questo concetto che vengono qui accennate.

4.3.2. Varianti

AdaGrad (*adaptive gradient*) è un metodo che utilizza un tasso di apprendimento indipendente per ogni parametro. I suoi vantaggi sono la sua convergenza generalmente più rapida rispetto a SGD e la capacità di adattare in automatico il learning rate, tuttavia questo parametro tende a cancellarsi ad ogni iterazione, con conseguente arresto dell'apprendimento.

RMSProp (*root mean square propagation*) è anch'esso un metodo a tasso di apprendimento adattivo, che però riesce a superare il problema dell'arresto dell'apprendimento introdotto da AdaGrad: il tasso di apprendimento viene controllato dalla media quadratica del gradiente. *AdaDelta* è invece un metodo simile con lo stesso punto di forza di RMSProp.

Adam (*adaptive movement estimation*) è un'estensione di RMSProp, ed è un algoritmo di ottimizzazione del secondo ordine che incorpora anche l'Hessiano[8].

Le ottimizzazioni che si fermano al primo ordine sono più facili da calcolare e impiegano meno tempo, al contrario gli ottimizzatori del secondo ordine sono più lenti. Si è scelto di analizzare due di queste forme di ottimizzazione per il calcolo dei punti di minimo: SGD e Adam. Il confronto tra le due verificherà e confermerà le migliori prestazioni di Adam.

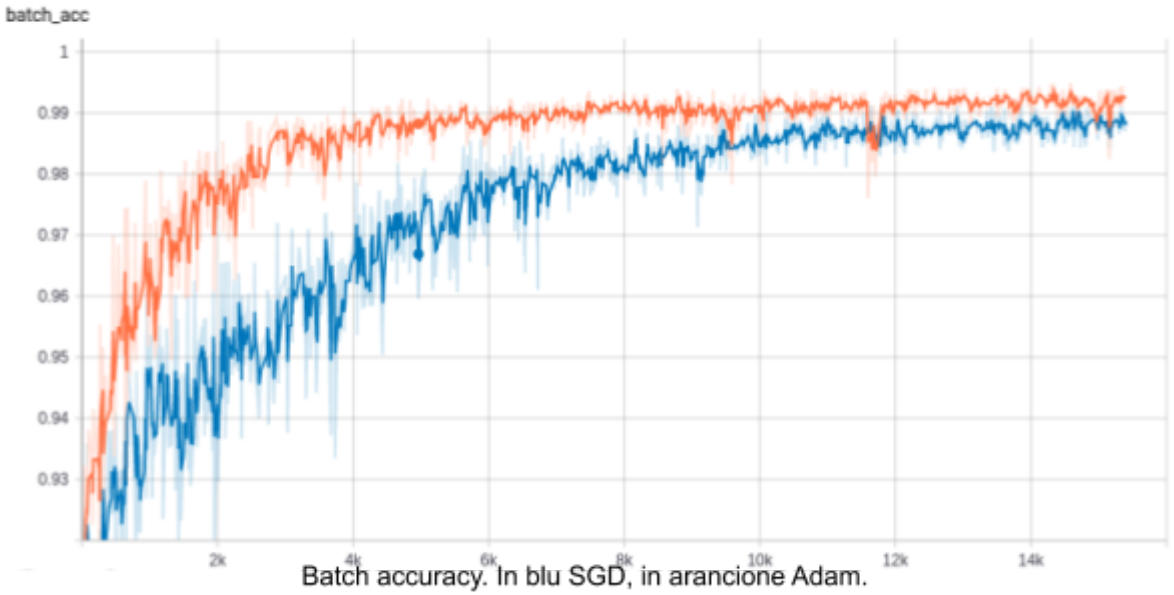
5. Risultati

Per l'analisi della rete si sono analizzati i comportamenti della stessa in corrispondenza a variazione delle funzioni e dei parametri che la contraddistinguono.

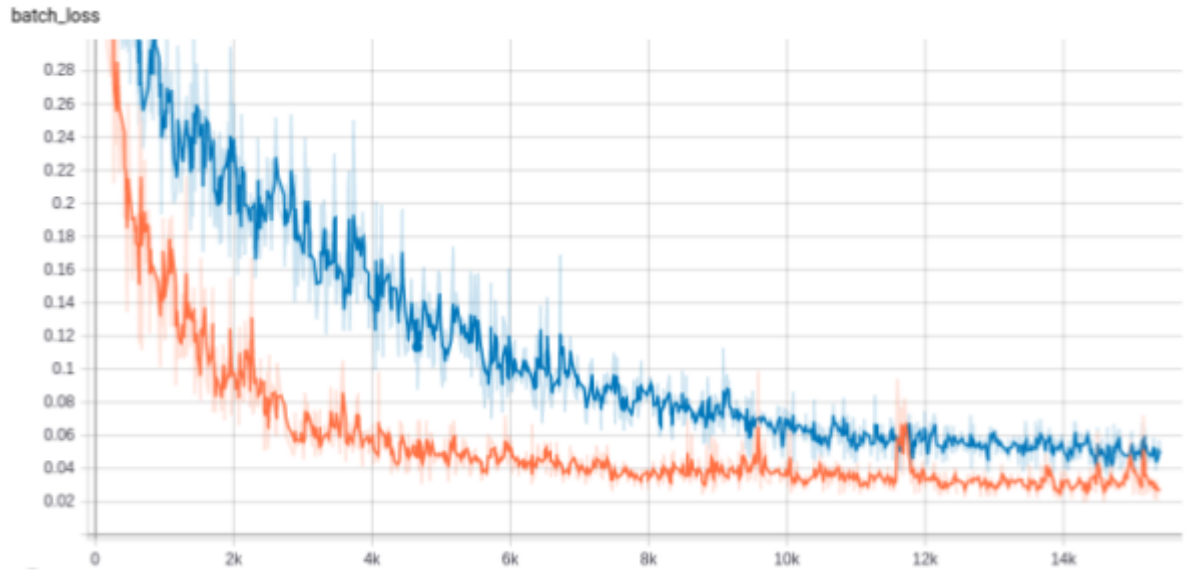
Il codice sorgente sviluppato è consultabile su github:
https://github.com/marziaf/hand-segmentation

5.1. SGD vs Adam

Un primo confronto è stato effettuato tra due metodi di ottimizzazione del gradiente: stochastic gradient descent e Adam. L'allenamento si è svolto sul 50% dei dati (770 immagini di allenamento, 220 per la validazione) a causa di problemi di memoria che hanno impedito l'utilizzo dell'intero set, con metrica di valutazione CE-dice loss, allenando la rete su 20 epoche. I risultati confermano la teoria: Adam è un algoritmo più performante di SGD. Questo si vede nei dati di allenamento, di cui sono riportati i grafici dell'accuratezza e della loss in funzione dei batch. Si vede come le curve blu, che rappresentano il training con SGD, rappresentino risultati più deboli rispetto alle arancioni di Adam.

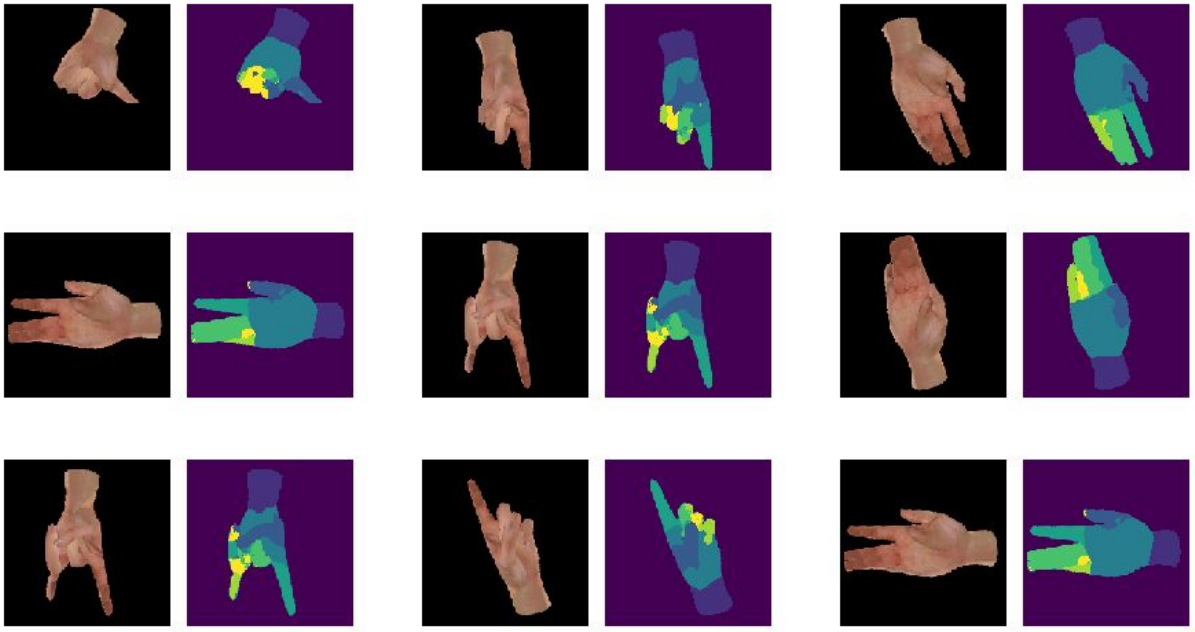


Batch accuracy. In blu SGD, in arancione Adam.
Nell'asse delle ascisse gli step (periodo di aggiornamento dei parametri) e nelle ordinate l'accuratezza



Batch loss. In blu SGD, in arancione Adam.
Nell'asse delle ascisse gli step e nelle ordinate la loss

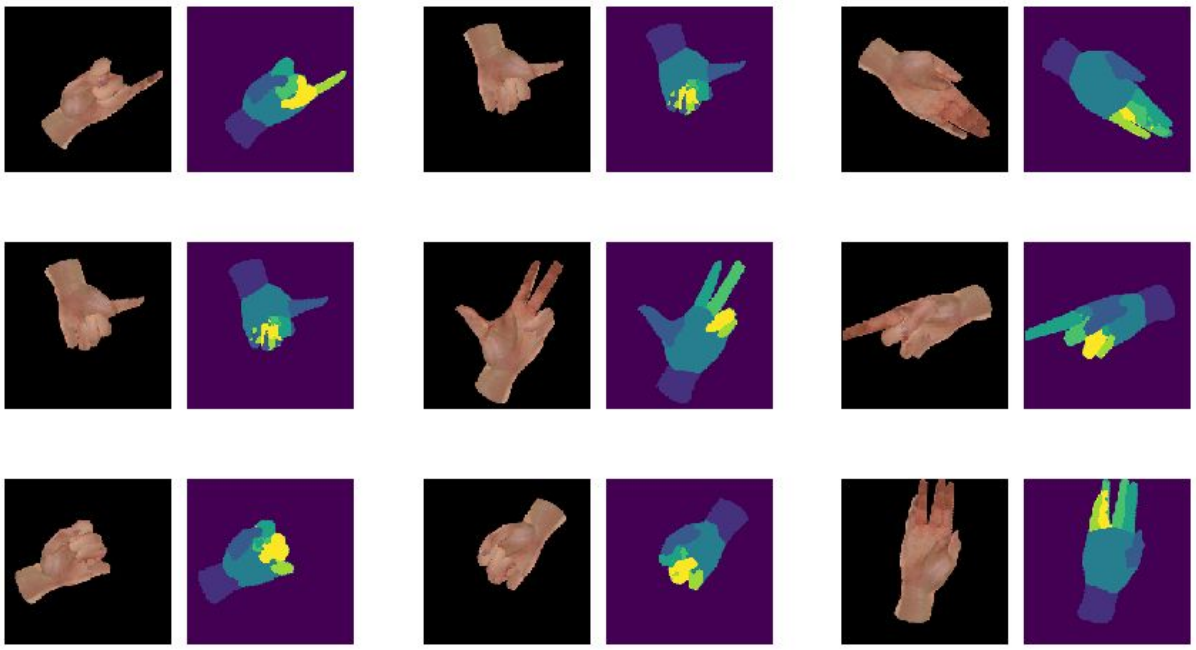
Anche visivamente, utilizzando il set di test per controllare l'output della rete con il solo ingresso dell'immagine RGBD, si nota la differenza.



Risultati ottenuti utilizzando Stochastic Gradient Descent

Per quanto riguarda SGD si osserva come sfondo, polso e palmo siano generalmente ben riconosciuti, ma è evidente la difficoltà nell'identificare le singole dita e

distinguerle tra loro, specialmente se non è presente dello spazio intermedio. Questo risultato non si può ritenere definitivo, e richiede miglioramenti.

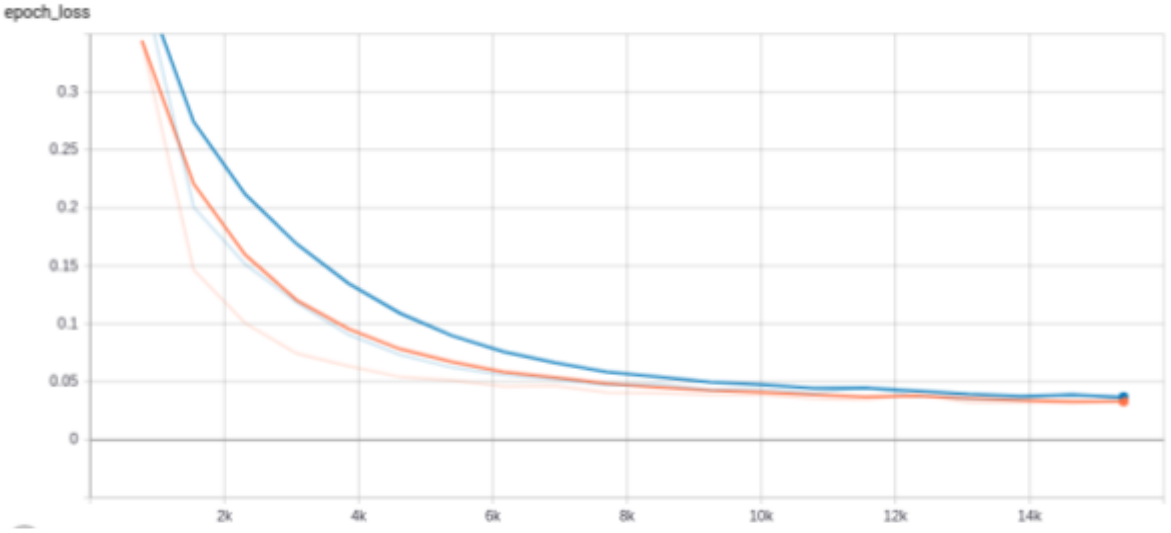


Risultati ottenuti con Adam

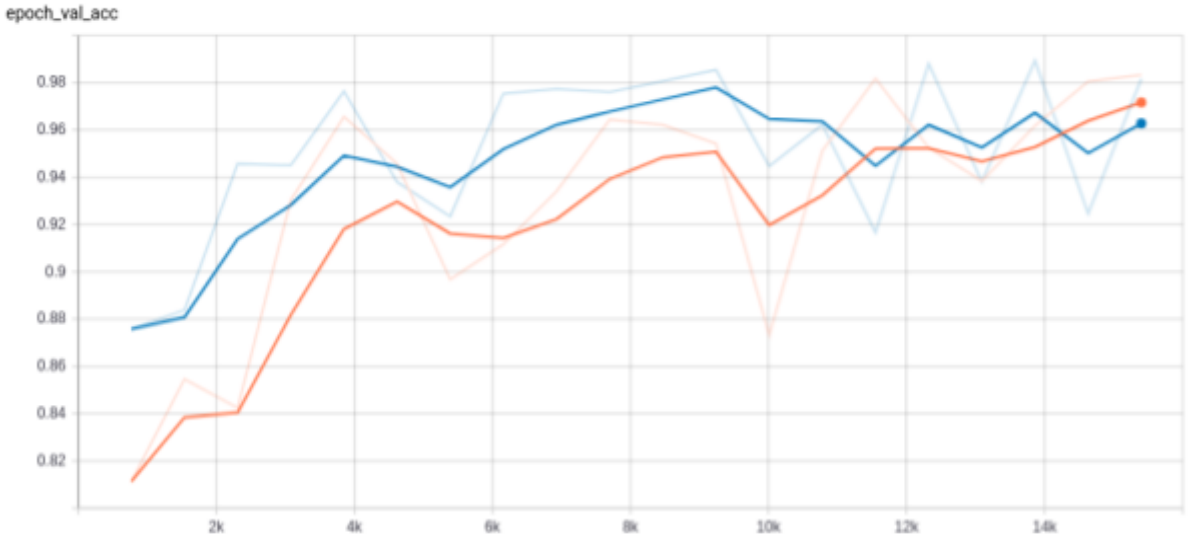
Con Adam invece si osservano dei miglioramenti: nonostante alcune criticità e una parziale fusione delle dita, che non sono ancora ben distinte, il risultato è migliore.

5.2. Scelta del batch size

La dimensione dei batch è stata condizionata molto dalle risorse a disposizione, in quanto un suo incremento comporta un maggior consumo di memoria. Questa grandezza indica il numero di immagini utilizzate ad ogni iterazione, ossia quanti dati vengono presi in considerazione dalla rete prima di aggiornare i suoi parametri. Un buon punto di partenza sarebbe stato di 4 o 8, ma potenzialmente anche maggiore. Purtroppo con il crescere di questo valore aumenta la complessità computazionale e l'occupazione di memoria, risorse di cui non c'è stata disponibilità. Si è fatto però un confronto tra l'allenamento tra batch di dimensione 1 e 2, e i risultati sono stati quelli attesi: la convergenza avviene più velocemente aumentando questo valore.



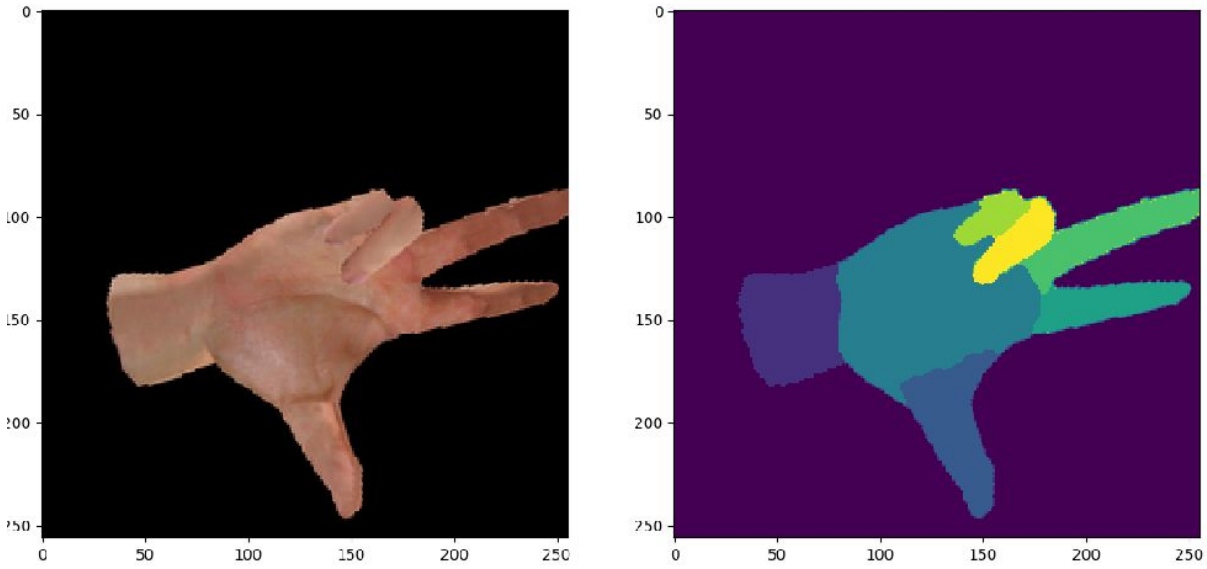
La discesa della loss nel corso delle epoche. In arancione batch size pari a 2, in blu 1



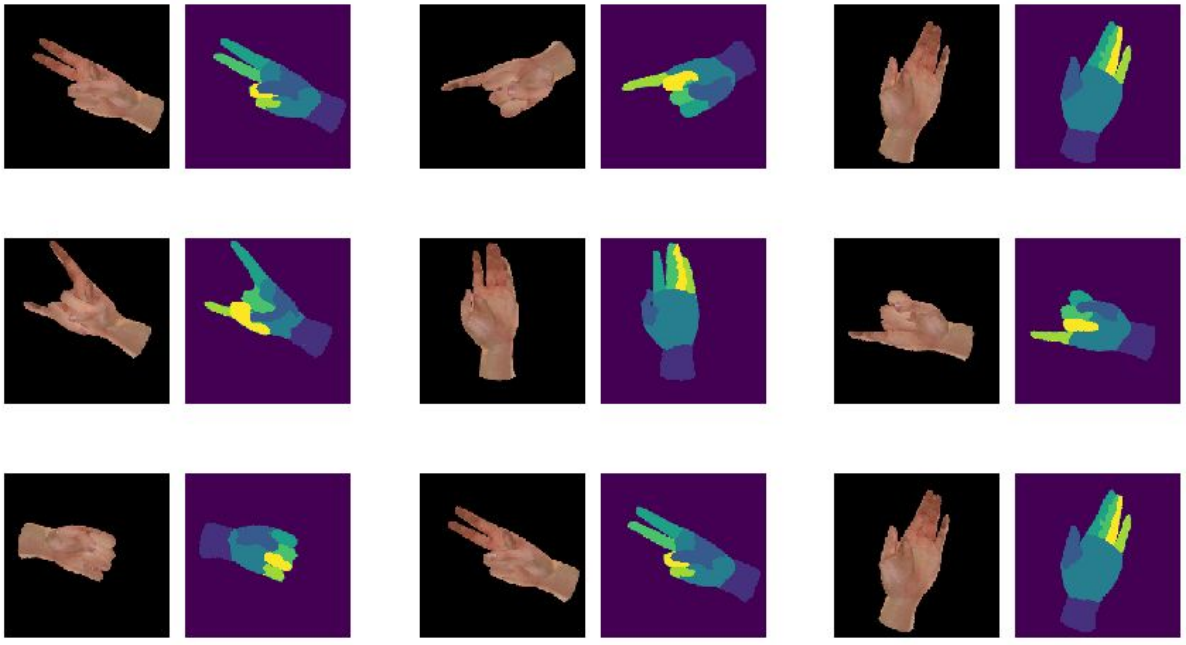
Accuratezza nel corso delle epoche di allenamento. In arancione batch size di 1, in blu di 2

5.3. Valutazione

Il miglior risultato ottenuto è frutto dell'allenamento su mille immagini di una U-net con 5 livelli di encoder-decoder, con aggiornamento dei parametri ogni 2 immagini e come metrica di interesse cross entropy dice loss e ottimizzazione Adam.



Un esempio specifico frutto del miglior modello



Un set di esempi per mostrare i risultati del miglior allenamento

Confrontando queste immagini con le precedenti, si vede come ora le dita vengano definite in modo decisamente più preciso.

Per la valutazione di questa rete sono state usate tre metriche oggettive per la definizione della qualità dei risultati: l'accuratezza di classificazione, l'accuratezza dei pixel e la mean intersection over union (mIoU).

L'accuratezza della previsione dei pixel è una statistica semplice ma che può portare a trarre conclusioni errate. La sua descrizione è banale, in quanto è la percentuale di pixel classificati correttamente sul totale, ma non è una buona descrizione dei risultati in contesti di sbilanciamento delle classi, come nel caso in questione di segmentazione della mano, in cui lo sfondo domina sulle altre classi. Questa metrica, come le successive, è stata misurata sulle singole coppie previsione-aspettativa appartenenti ad un subset del test, e i risultati riportati sono la loro media. La pixel accuracy misurata è pari a 98.02%, risultato che sembra ottimo, ma che verrà smentito dalle altre metriche.

Sebbene la pixel accuracy non sia uno strumento ottimo, si può migliorare eliminando la classe dominante, che in quanto tale, ha una buona precisione. Si ottiene quindi il 94.05% di precisione eliminando l'analisi dello sfondo.

L'accuratezza di classificazione è la statistica sull'errore di previsione delle classi e per ogni label è definita come il rapporto $TP/(TP+FN)$, dove T (true) indica il successo, F (false) l'errore, P il caso positivo di appartenenza alla label, N, negativo, se il pixel è stato mappato altrove. La mean classification accuracy è quindi la media tra la precisione delle singole classi e restituisce il 92% di accuratezza.

Infine la intersection over union, la più precisa delle statistiche, misura il numero di pixel in comune tra le maschere target e di previsione, dividendo il totale per i pixel appartenenti all'unione delle due. Mediando i risultati delle maschere di ogni classe si ottiene la mIou. Questa, come la pixel accuracy, è stata misurata comprendendo lo sfondo, ottenendo 86.75%, ed escludendolo scendendo così a 85.08%.

Da questo studio si capisce l'importanza nello scegliere metriche che tengano conto dello sbilanciamento delle classi per l'allenamento.

6. Conclusioni

In questa tesi si è discussa la costruzione di una rete neurale per la segmentazione di una mano. Nonostante l'allenamento di questa non si sia svolto nell'ambiente migliore, e quindi non si siano sfruttate al massimo le potenzialità della rete, i risultati ottenuti sono molto buoni. Tuttavia sono migliorabili a partire dai dati: delle trasformazioni citate solo alcune sono state implementate, ma è importante generare dati ancora più vari perché la rete sia in grado di gestire la varietà di mani e gesti che può incontrare nelle immagini reali. Anche le metriche di riferimento per l'allenamento non sono state ottimizzate al massimo: si è visto nell'ultima parte come pesi lo sbilanciamento delle classi e come quindi sia fondamentale per ottenere un buon risultato utilizzare metriche che ne tengano conto, perciò potrebbe essere opportuno sostituire alla CE-dice loss una metrica come la mIoU.

L'accuratezza dei risultati sperimentali, misurata sia in termini di mIoU, sia con il riscontro visivo, è incoraggiante e suggerisce che i risultati ottenuti abbiano una stabilità tale da permettere di ottimizzare il riconoscimento gestuale. Per l'implementazione di questa nuova abilità della rete si può effettuare la stima del gesto in parallelo, modificando il livello di output sostituendolo a uno densamente connesso.

Dopo aver apportato le opportune modifiche e aver verificato l'abilità di predizione del gesto si potrà passare alla fase di test sui dati reali e la conseguente applicazione su dispositivi per ottenere un vero riscontro pratico dell'abilità della macchina di interpretare i gesti dell'uomo.

7. Bibliografia

[1] Natural Networks and Deep Learning - Michael Nielsen - Determination Press - 2015

[2] Encoder-Decoder with Atrous Separable Convolution for Semantic Image Segmentation - Liang-Chieh Chen, Yukun Zhu, George Papandreou, Florian Schroff, and Hartwig Adam - arXiv - 2018

[3] Fully Convolutional Networks for Semantic Segmentation - Jonathan Long, Evan Shelhamer, Trevor Darrell, UC Berkeley - The IEEE Conference on Computer Vision and Pattern Recognition (CVPR) - 2015

[4] U-Net: Convolutional Networks for Biomedical Image Segmentation - Olaf Ronneberger, Philipp Fischer, and Thomas Brox - Part of Lecture Notes in Computer Science book series - 2015

[5] Generalized Wasserstein Dice Score for Imbalanced Multi-class Segmentation using Holistic Convolutional Networks - Lucas Fidon, Wenqi Li, Luis C. Garcia-Peraza-Herrera, Jinendra Ekanayake, Neil Kitchen, Sébastien Ourselin, and Tom Vercauteren - arXiv - 2017

[6] EddyNet: A Deep Neural Network For Pixel-Wise Classification of Oceanic Eddies - Redouane Lguensat, Miao Sun, Ronan Fablet, Evan Mason, Pierre Tandeo, and Ge Chen - arXiv - 2017

[7] Dice Loss Function for Image Segmentation Using Dense Dilation Spatial Pooling Network - Qiuhua Liu, Min Fu - 2018

[8] Adam: A Method for Stochastic Optimization - Diederik P. Kingma, Jimmy Lei Ba - arXiv - 2017

[9] Head-Mounted Gesture Controlled Interface for Human-Computer Interaction - Alvise Memo, Pietro Zanuttigh - Multimedia Tools and Applications - 2018