



UNIVERSITÀ DEGLI STUDI DI FIRENZE
SCUOLA DI SCIENZE MATEMATICHE FISICHE E NATURALI
DIPARTIMENTO DI INFORMATICA

Tesi di laurea Magistrale in Informatica - Data Science

RICONOSCIMENTO DI AZIONI UMANE USANDO TECNICHE DI APPRENDIMENTO PROFONDO PER LA STIMA DELLA POSA

Candidato
Andrea Moscatelli

Relatore
Marco Bertini

Correlatore
Correlatore 1

ANNO ACCADEMICO 2018 - 2019

Indice

Prefazione	iii
Introduzione	iv
1 L'apprendimento automatico	1
1.1 Deep learning	3
1.2 Le reti neurali	4
1.3 Le reti neurali ricorrenti	7
1.4 Le reti LSTM	10
2 Lavori precedenti	14
2.1 Recognizing Human Actions as the Evolution of Pose Estimation Maps (2018)	15
2.2 Actional-Structural Graph Convolutional Networks for Skeleton-based Action Recognition (2019)	20
2.3 Vertex feature encoding and hierarchical temporal modelling in a spatial-temporal graph convolutional network for action recognition (2019)	20
3 Stima della posa	21
3.1 PoseNet	23
3.1.1 Stima dei key-points	24
3.1.2 Raggruppamento dei key-points in istanze di persona . . .	26
3.2 Detectron2	27
3.2.1 Mask R-CNN	30

3.2.2	Predizione della posa con Mask R-CNN	32
4	Metodo proposto	33
4.1	Estrazione delle pose	33
4.2	Assegnazione delle pose	35
4.3	Rimozione degli zeri	37
4.4	39
5	Il dataset NTU RGB+D	40
5.1	NTU RGB+D in dettaglio	41
5.2	Criteri di valutazione	44
5.2.1	Valutazione Cross-subject	44
5.2.2	Valutazione Cross-view	45
6	Esperimenti e risultati	46
7	Conclusioni e sviluppi futuri	47
	Bibliografia	48

Prefazione

prefazione

Introduzione

Introduzione

Capitolo 1

L'apprendimento automatico

Quando parliamo di apprendimento automatico (in inglese *machine learning*), ovvero *l'apprendimento delle macchine*, viene spontaneo domandarsi “Come impara una macchina?”.

L'apprendimento viene definito come un processo iterativo che permette di mutare le proprie conoscenze a seconda delle informazioni che vengono raccolte.

Nel machine learning l'apprendimento segue esattamente le stesse dinamiche ma a seconda di come gestiamo le informazioni a nostra disposizione vengono definite 4 macro classi.

- **Apprendimento supervisionato (Supervised Learning)** - questo primo caso, può essere paragonato al processo di apprendimento che si avrebbe con un'insegnante *onnisciente* che supervisiona il suo alunno e che interromperà l'apprendimento solo quando quest'ultimo raggiungerà un livello accettabile del compito da imparare.

L'apprendimento supervisionato lo si ha cioè quando nel nostro *dataset* di allenamento sono a disposizione sia i dati di input X che quelli di output Y e vogliamo insegnare al nostro algoritmo quella funzione f tale che

$$Y = f(X).$$

L'obiettivo è quello di approssimare sufficientemente bene la funzione f di modo da poter prevedere correttamente il valore Y_i per un futuro X_i non compreso nel nostro dataset di allenamento.

I problemi di apprendimento supervisionato si possono dividere in problemi di:

1. *Classificazione*, ovvero insegnare ad una macchina a categorizzare. In questo scenario, nel nostro dataset di allenamento ad ogni dato di input sarà associata un'etichetta che ne indica la categoria appartenente. Più grande sarà il dataset e maggiore sarà l'informazione a disposizione dell'algoritmo per imparare.
 2. *Regressione* - ovvero insegnare ad una macchina a predire il valore di ciò che sta analizzando. A differenza della classificazione, in questo caso il risultato in output sarà un valore continuo e non un valore categorico. Ad esempio, dati in input le ore di studio di uno studente per la preparazione di un esame e le rispettive ore di sonno, prevedere la probabilità di superamento dell'esame in questione, oppure date in input la superficie e la posizione di un appartamento, prevederne il valore di mercato.
- **Apprendimento non supervisionato (Unsupervised Learning)** - in questo tipo di apprendimento, al contrario di quello supervisionato, non viene fornita nessuna etichetta di output Y per i nostri dati X . L'obiettivo che quindi ci si pone in questo tipo di situazione è scovare delle relazioni tra i dati analizzati. In questo caso non c'è nessun "insegnante" a guidare l'apprendimento e non ci sono risposte corrette o sbagliate, l'algoritmo deve cercare di scoprire "da solo" se ci sono delle strutture decifrabili nei dati.

I problemi di apprendimento non supervisionato si possono dividere in:

1. *Raggruppamento* - detto anche *clustering*, si utilizzano quando è necessario raggruppare i dati che presentano caratteristiche simili. In questo caso l'algoritmo non fa uso di dati categorizzati, come visto in precedenza, ma estrae una regola di raggruppamento secondo caratteristiche che ricava dai dati stessi.
2. *Associazione* - strettamente legata al *Data Mining*, questa classe di problemi è utile in tutti i casi per i quali siamo interessati a scoprire

regole induttive nei dati analizzati, ad esempio la tendenza nei consumatori di un certo supermercato ad acquistare il prodotto *A* dopo aver acquistato il prodotto *B*. Ci si pone quindi l'obbiettivo di identificare schemi frequenti, associazioni, correlazioni o strutture casuali fra gli *item* di un database relazionale cercando di scoprire le regole che predicono l'evento di un certo item in base agli eventi degli item ad esso legati.

- **Apprendimento parzialmente supervisionato (Semi-Supervised Learning)** - questo tipo di apprendimento è una sorta di via di mezzo dei primi due, ovvero un apprendimento utilizzato quando solo alcuni dei nostri dati in input *X* sono etichettati, mentre la maggior parte di essi non lo è.

In questa categoria di apprendimento ricadono la stragrande maggioranza dei problemi di machine learning e questo perchè etichettare dati è un processo lungo e costoso soprattutto in ambito *Big Data*, campo ideale per il machine learning.

- **Apprendimento con rinforzo (Reinforcement Learning)** - questa metodologia di apprendimento simula il processo di apprendimento umano per tentativi ed errori. L'algoritmo si adatta gradualmente tramite un sistema di valutazione basato su ricompense e penalità a seconda della decisione presa. L'obbiettivo è quello di evolvere l'algoritmo massimizzando le ricompense ricevute.

1.1 Deep learning

Il *deep learning* (o *apprendimento profondo* in italiano) è una delle tecniche di machine learning che “insegna” ai computer una cosa estremamente naturale al cervello umano, ovvero *imparare per esempi*. Il deep learning è la tecnologia chiave grazie alla quale abbiamo oggi automobili che si guidano da sole, riconoscimento e controllo vocale dei device, riconoscimento di tumori, traduzioni automatiche, colorazione automatica di vecchi filmati in bianco e nero e molte altre cose ritenute impossibili solo fino a pochi anni fa.

Nel deep learning un computer impara, ad esempio, un task di classificazione direttamente da immagini, testi e suoni e può raggiungere prestazioni allo stato dell'arte superando talvolta persino l'accuratezza umana. I modelli vengono allenati usando grandi dataset etichettati seguendo un'apprendimento di tipo supervisionato ed il sistema centrale sul quale è basato il deep learning è la *rete neurale*.

1.2 Le reti neurali

Nel campo dell'apprendimento automatico, una rete neurale (*neural network* o *NN*) è un modello di calcolo ispirato al sistema di elaborazione delle informazioni tipico del cervello degli esseri viventi nel quale tanti piccoli elementi base, denominati *neuroni* (figura 1.1), sono interconnessi e collaborano tra loro per poter eseguire un determinato compito.

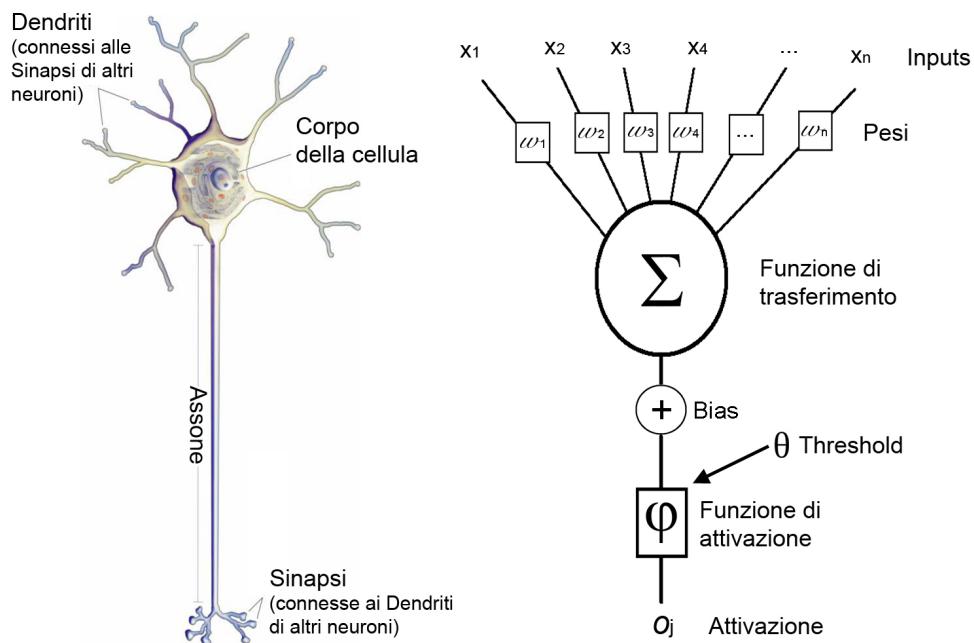


Figura 1.1: Un neurone biologico (sinistra) ed un neurone artificiale (destra) a confronto. La struttura del neurone artificiale si ispira molto a quella del neurone biologico.

Queste reti costituiscono ad oggi la miglior soluzione per una vasta gamma di problemi grazie alla loro capacità di approssimazioni di pressoché qualsiasi funzione $f : \mathbb{R}^n \rightarrow \mathbb{R}$, a patto di scegliere la giusta quantità di neuroni in fase di progettazione.

La struttura di una rete neurale si sviluppa per livelli ed ogni livello, eccetto quello di *input* e quello di *output*, viene chiamato *hidden layer*. Ogni hidden layer può essere composto da un qualsiasi numero di neuroni, che prendono il nome di *hidden units* e maggiore sarà il numero di hidden units, maggiore sarà la capacità espressiva della rete, ovvero la capacità di approssimare correttamente una qualsiasi funzione.

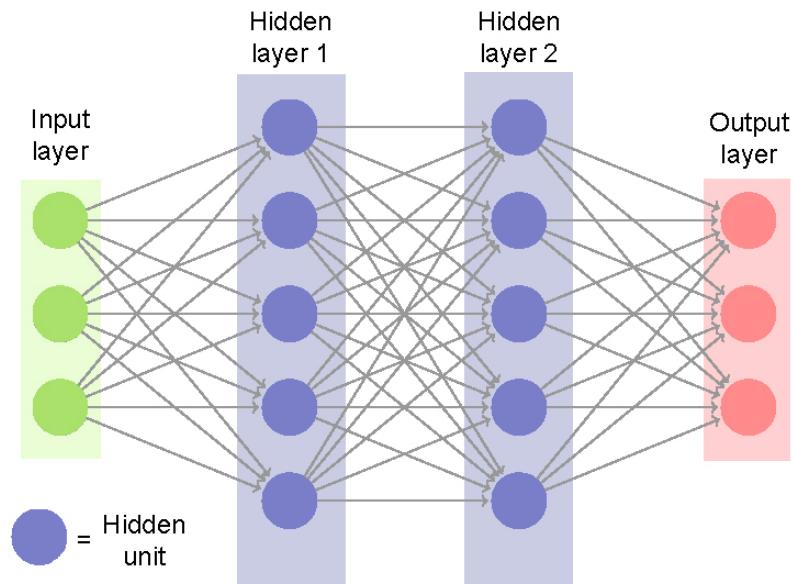


Figura 1.2: Struttura di una rete neurale. Ogni livello della rete, eccetto il primo e l'ultimo, viene chiamato *hidden layer* ed i neuroni che lo compongono prendono il nome di *hidden units*. Maggiore sarà il numero di hidden units e maggiore sarà la capacità espressiva della rete.

Sono proprio gli hidden layer a dare *profondità* alla rete ed è per questo che si parla sia di *apprendimento profondo* che di *reti neurali profonde* (o *deep neural network* in inglese).

Negli ultimi anni le deep neural network hanno avuto una diffusione sorprendente non solo grazie alla loro adattabilità ad una vasta gamma di problemi, ma anche alla rapida evoluzione che hanno avuto le GPU, che hanno reso parallelizzabile l'addestramento di queste reti.

I livelli che possiamo trovare all'interno di una rete neurale possono essere di diverso tipo, come diverse possono essere le connessioni dei neuroni fra un livello e l'altro. Sono proprio queste diversità a determinare il tipo di rete neurale e le più famose tipologie sono *w*:

- **fully connected network** - in questo tipo di reti ogni neurone è connesso con tutti quelli del livello precedente (come in figura 1.2). Come si può facilmente intuire, il numero dei parametri (ovvero la capacità espressiva) della rete cresce esponenzialmente al crescere della sua struttura ed il rischio che si corre è quello di creare una rete talmente efficiente nell'ottimizzare la funzione generatrice dei dati di allenamento da essere troppo specifica per essa, non riuscendo a predire correttamente valori al di fuori di quel dataset. Questa problematica, ben nota nel mondo del deep learning, prende il nome di *overfitting*.
- **convolutional neural network (CNN)** - queste reti sono composte principalmente da *layer convoluzionali*, ovvero layer nei quali ogni neurone è connesso solo ad una piccola regione localizzata del livello precedente (figura 1.3). Questa particolare tipologia di connessione è estremamente efficiente nell'ambito della visione computazionale ed è in grado di *sintetizzare* correttamente un'immagine senza perdita d'informazione. Questa sintetizzazione viene indicata in gergo col termine di *features extraction*, ovvero l'estrazione delle caratteristiche principali.

Le reti neurali sopra citate, seppur molto efficienti nei loro ambiti, presentano però un problema di *memoria* ovvero, non sono state ideate per processare una serie temporale di dati. L'unica soluzione che hanno è quella di trattare tutta la serie temporale come un unico input e processarlo interamente in un unico ciclo. Questa primordiale soluzione non ha ovviamente preso campo vista la mancanza di generalità.

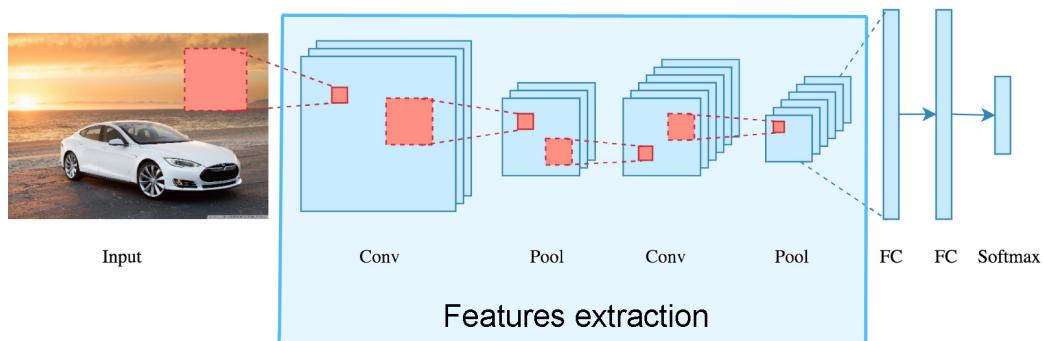


Figura 1.3: Un esempio di rete convoluzionale per l’analisi di un’immagine. *Conv* = “Convolution layer”, moltiplicazione vettoriale della porzione di layer al quale ogni neurone è collegato; *Pool* = “Pooling layer”, sintetizzazione della porzione di layer connessa; *FC* = “Fully connected layer”; *Softmax* = operazione conclusiva per la decisione del valore di output.

Esistono però delle reti neurali che, a differenza dalle precedenti, riescono ad analizzare una sequenza temporale in maniera più simile al ragionamento umano, ovvero mantenendo una memoria del “passato” ed in base a questa adattare la valutazione dello stato corrente. Queste particolari reti vengono chiamate *reti neurali ricorrenti* (o *recurrent neural networks*).

1.3 Le reti neurali ricorrenti

Le reti neurali ricorrenti (*recurrent neural network* o *RNN*) sono una classe di rete neurali artificiali caratterizzate da una struttura ciclica, ovvero una struttura dove i layer possono essere connessi con loro stessi o con layer precedenti (figura 1.4).

Questa struttura ciclica permette di mantenere una sorta di *stato di memoria* della rete utilizzato per processare ogni elemento della sequenza in input che si è rivelato ideale per compiti di analisi predittiva su sequenze temporali, quali possono essere ad esempio il riconoscimento della grafia, l’analisi di video, il riconoscimento vocale o l’analisi di testi.

Per quanto riguarda l’addestramento di queste reti, esso avviene con la stessa modalità delle reti neurali standard ovvero, i pesi dei neuroni che compongono la

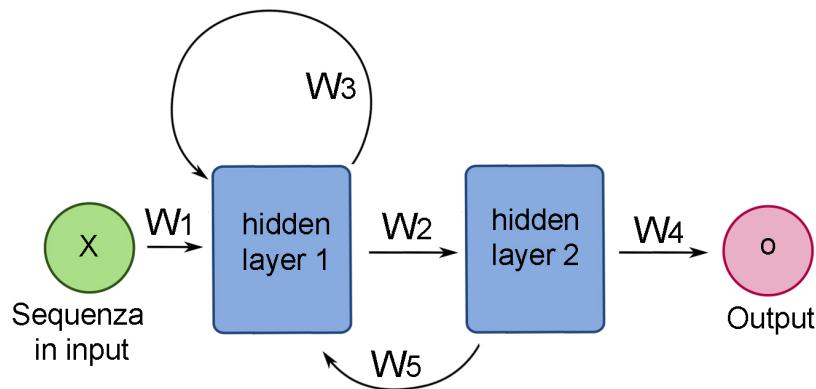


Figura 1.4: Un esempio di rete ricorrente con due hidden layers.

rete vengono regolati tramite un processo iterativo con lo scopo di minimizzare l'errore di classificazione prodotto sul dataset di allenamento.

Tipicamente, l'algoritmo che regola l'apprendimento delle RNN viene chiamato *Backpropagation Throw Time (BPTT)* e non è altro che la versione "ricorrente" del classico *backpropagation algorithm* usato per le reti neurali standard.

L'applicazione del BPTT può essere vista come l'applicazione del backpropagation algorithm allo "srotolamento" della RNN (figura 1.5).

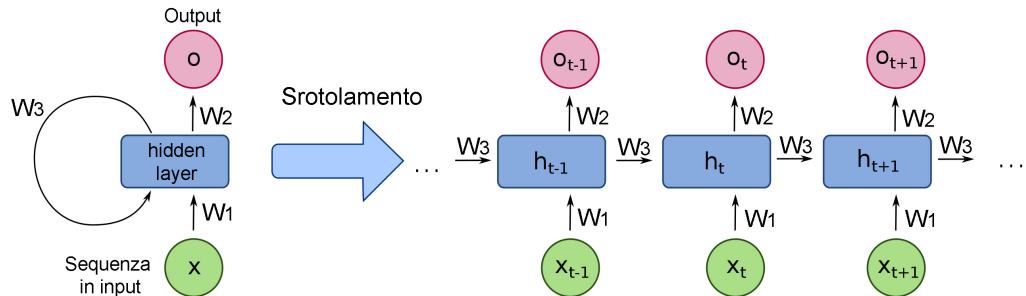


Figura 1.5: Esempio di "srotolamento" di una rete ricorrente, ovvero la trasformazione da una rete ricorrente ad una combinazione equivalente di reti neurali semplici.

La lunghezza delle sequenze che queste reti sono in grado di analizzare è teoricamente infinita, ma di fatto per sequenze troppo lunghe si presenta un

problema legato alla precisione finita dei calcolatori, ovvero la *scomparsa del gradiente* (o in inglese *vanishing gradient problem*).

Il concetto alla base del backpropagation algorithm è infatti quello di aggiornare iterativamente ogni parametro del modello in maniera proporzionale alla derivata parziale della *loss function* (ovvero la funzione d'errore di classificazione) rispetto al parametro stesso.

Poiché ogni gradiente sarà un valore compreso fra 0 e 1 e l'aggiornamento dei parametri ai vari livelli verrà propagato tramite la *regola della catena*, ovvero

$$\frac{\partial a}{\partial b} = \frac{\partial a}{\partial c} \cdot \frac{\partial c}{\partial b},$$

il prodotto dei gradienti decrescerà esponenzialmente al crescere della profondità della rete, col rischio di sparire se dovesse diventare più piccolo della precisione minima del calcolatore.

La profondità delle reti ricorrenti è strettamente legata alla lunghezza della sequenza analizzata e raggiunge velocemente un numero di livelli intrattabile per l'algoritmo di backpropagation, rendendole di fatto inutilizzabili per i casi nel mondo reale.

Per ovviare a questo problema, nel 1997 Sepp Hochreiter e Jürgen Schmidhuber proposero un nuovo tipo di rete ricorrente chiamato *Long short-term memory (LSTM)* [20].

1.4 Le reti LSTM

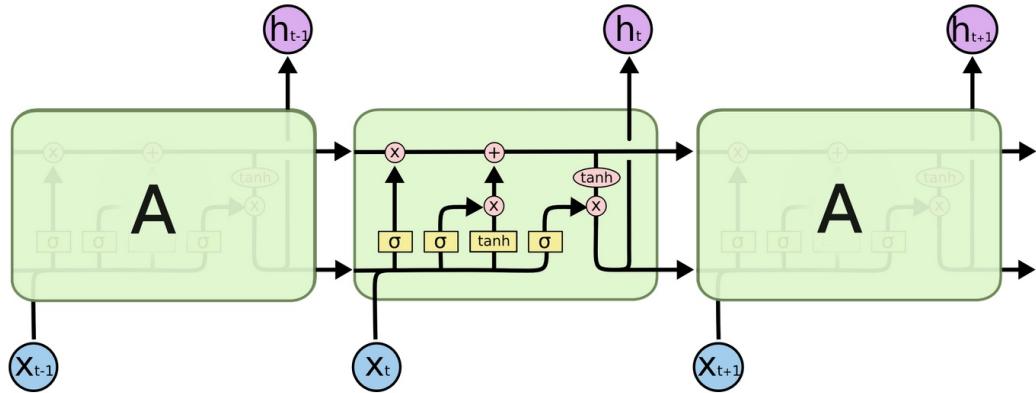


Figura 1.6: Esempio di “srotolamento” di una rete LSTM. In questa figura ogni unità di rete “A” prende in input al tempo t un elemento X_t della sequenza temporale X e restituisce uno stato in output h_t . \tanh e σ sono le funzioni di tangente iperbolica e sigmoidale rispettivamente, mentre “+” e “ \times ” sono rispettivamente le operazioni vettoriali di somma e moltiplicazione.

Le reti Long Short-Term Memory, normalmente chiamate LSTM, sono delle particolari reti ricorrenti in grado, non solo di imparare le dipendenze *a lunga distanza* presenti in una sequenza in input, ma anche di risolvere il problema della scomparsa del gradiente.

Come ogni altra rete ricorrente anche per le LSTM è possibile srotolarne la struttura ed ottenere una catena di “pezzi di rete” chiamati anche *unità* (figura 1.6). Il funzionamento di un’unità ricorrente LSTM ruota tutto intorno ai concetti di *cell state* (figura 1.7a) e di *gate* (figura 1.7b), ovvero:

- **cell state:** funziona come un nastro trasportatore in grado di memorizzare informazioni, rendendole disponibili durante tutta l’analisi della sequenza;
- **gate:** sono delle NN più piccole che stabiliscono, attraverso una funzione sigmoidale, quali informazioni possono continuare ad essere memorizzate nel cell state e quali invece devono essere “dimenticate”.

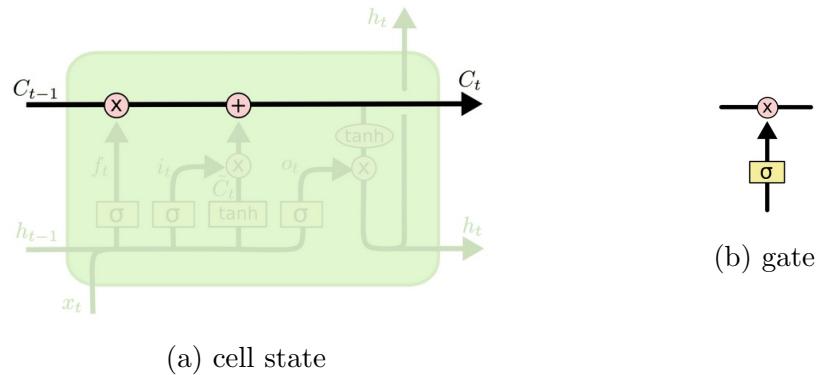


Figura 1.7: I concetti chiave di una LSTM.

Esistono principalmente tre diverse tipologie di gate che regolano il flusso informativo all'interno di una unità LSTM: *forget gate*, *input gate* e *output gate*.

Forget gate - Il forget gate (figura 1.8) decide quali informazioni mantenere all'interno del cell state e quali invece saranno “dimenticate”.

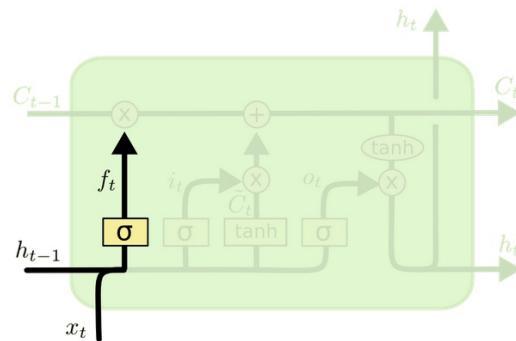


Figura 1.8: Forget gate di una LSTM

In formula, al t -esimo passo, il forget gate sarà

$$f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f)$$

dove:

- h_{t-1} è lo stato della LSTM al passo precedente
- x_t è il t -esimo elemento della sequenza in input

- W_f e b_f sono rispettivamente la matrice dei pesi e il bias del forget gate
- σ è la funzione sigmoidale.

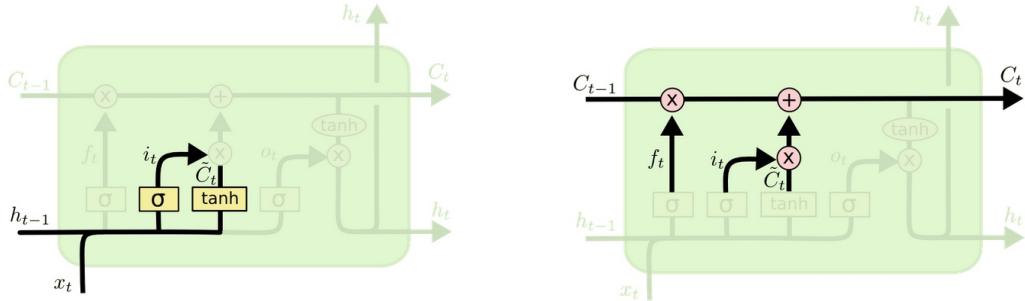
Input gate - Questo gate decide quali nuove informazioni vogliamo registrare nel nostro cell state e la sua funzione può essere suddivisa in due parti (figura 1.9a):

1. attraverso un'operazione sigmoidale, seleziona cosa dell'input corrente deve essere mantenuto. In formula,

$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i).$$

2. trasforma il precedente stato della LSTM con un'operazione $tanh$, ovvero

$$\tilde{C} = \tanh(W_C \cdot [h_{t-1}, x_t] + b_C)$$



(a) Selezione dell'input corrente e trasformazione dello stato LSTM precedente.

(b) Combinazione col forget gate per il calcolo del nuovo stato del cell state.

Figura 1.9: Input gate.

I due vettori risultanti vengono dapprima moltiplicati fra loro e successivamente sommati al cell state combinandoli col forget gate (figura 1.9b). In formula

$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t.$$

Output gate - Questo ultimo gate si occupa infine di decidere cosa vogliamo restituire in output ad ogni passo t (figura 1.10). Il suo compito è quello di combinare una versione filtrata dello stato precedente della LSTM con una trasformazione del cell state. L'operazione applicata al filtraggio dello stato precedente è quella sigmoidale, di modo da selezionarne solo i componenti desiderati, ovvero

$$o_t = \sigma(W_o[h_{t-1}, x_t] + b_o)$$

mentre quella applicata al cell state è \tanh di modo da proiettarne i valori nel range (-1,1), ottenendo

$$h_t = o_t * \tanh(C_t)$$

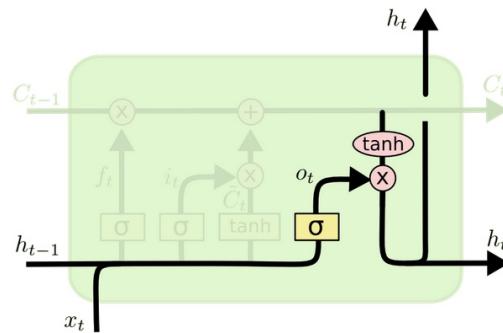


Figura 1.10: Output gate di una LSTM

Il valore h_t così calcolato sarà l'output al passo t della rete LSTM e l'input del passo successivo, insieme allo stato corrente del cell state C_t .

Quella appena descritta è una classica LSTM ma in alcuni articoli scientifici è possibile trovare delle leggere varianti, come le “peephole” LSTM [21] o le Gated Recurrent Unit (GRU) [22]. Ad ogni modo il concetto alla base di queste varianti resta lo stesso.

Capitolo 2

Lavori precedenti

Il riconoscimento di azioni corporee tramite l'analisi della *posa* umana o *skeleton* (vedremo nel prossimo capitolo più in dettagli di cosa si tratta) sta attraendo recentemente una notevole attenzione nel mondo della visione computerizzata. Il suo successo è senz'altro dovuto non solo agli ottimi risultati ottenuti, ma anche alla sua efficiente semplificazione della struttura umana riducendo di fatto i costi computazionali e le risorse necessarie allo stoccaggio dati. Normalmente la posa viene calcolata utilizzando sensori 2D o 3D posizionati in prossimità dei giunti corporei (gomiti, ginocchi, spalle,...), oppure con algoritmi appositi in grado di estrarla da normali video RGB.

Normalmente ci sono due approcci per il riconoscimento di azioni umane facendo uso dalla posa: gli approcci “*a mano*” e quelli *deep learning*.

Negli approcci *a mano* si cerca di categorizzare le azioni umane seguendo pattern fisici intuitivi, come le posizioni degli arti o le correlazioni temporali fra essi, come ad esempio il movimento ondulatorio delle gambe durante una camminata o una corsa.

Per quanto riguarda invece gli approcci *deep learning* le tipologie di azioni corporee vengono categorizzate in maniera automatica direttamente dai dati stessi. Reti neurali ricorrenti, come le LSTM, o quelle di tipo *convoluzionali-temporali*, come quella in [26], si sono rivelate essere estremamente efficaci per il riconoscimento di pattern temporali in un video, ottenendo ottimi risultati nella categorizzazione delle azioni.

Esiste infine un terzo approccio che recentemente sta riscuotendo un discreto interesse, ovvero quello basato su *grafi*, che punta ad esprimere contemporaneamente sia le relazioni temporali che quello nello spazio dei giunti corporei. Questa architettura rappresenta la sequenza di pose in un video come un grafo composto da archi temporali e spaziali, considerando cioè rispettivamente le relazioni inter ed intra-frame.

Ad oggi i migliori risultati sulla categorizzazione dei movimenti corporei possono essere riassunti nei seguenti articoli scientifici:

- **Recognizing Human Actions as the Evolution of Pose Estimation Maps** - Mengyuan Liu, Junsong Yuan - 2018 [23]
- **Actional-Structural Graph Convolutional Networks for Skeleton-based Action Recognition** - Maosen Li, Siheng Chen, Xu Chen, Ya Zhang, Yanfeng Wang, and Qi Tian - 2019 [24]
- **Vertex feature encoding and hierarchical temporal modeling in a spatial-temporal graph convolutional network for action recognition** - Konstantinos Papadopoulos, Enjie Ghorbel, Djamila Aouada, Bjorn Ottersten - 2019 [25]

Vediamo adesso più nel dettaglio come questi lavori hanno affrontato il problema del riconoscimenti di azioni umane.

2.1 Recognizing Human Actions as the Evolution of Pose Estimation Maps (2018)

L'idea alla base di questo lavoro è quella di creare delle *mappe di stima della posa* umana dalle quali estrarre delle *heatmaps* globali e delle *predizioni di pose* e una volta ottenute queste due componenti, combinare le loro caratteristiche temporali e spaziali di modo da identificare l'azione svolta.

Più precisamente, indicando con $\mathcal{Y}_k \in \{x, y\}$ le coordinate del k -esimo giunto corporeo, possiamo definire un'intera struttura corporea come $\mathcal{Y} = \{\mathcal{Y}_1, \dots, \mathcal{Y}_k, \dots, \mathcal{Y}_K\}$, dove K è il numero totale dei tipi di giunti corporei considerati.

Addestrando un classificatore multiclasso g_t^k per predire la posizione del k -esimo giunto corporeo al passo t , otteniamo per ogni punto \mathbf{z} dell'immagine una mappa di stima relativa al giunto k ,

$$\mathbf{B}_t^k(\mathcal{Y}_k = \mathbf{z}) = g_t^k \left(\mathbf{f}_z; \bigcup_{i=1,\dots,K} \psi(\mathbf{z}, \mathbf{B}_{t-1}^i) \right)$$

dove \mathbf{f}_z è la *color feature* per il punto \mathbf{z} , \mathbf{B}_{t-1}^i è la mappa stimata da g_{t-1}^i al passo precedente, \bigcup è l'operatore di concatenazione vettoriale e ψ è la funzione di estrazione della *feature* interessata partendo dalla mappa ottenuta al passo precedente. Dopo T passi, le mappe ottenute vengono utilizzate per stimare la posizione dei giunti.

Partendo dalle K mappe così ottenute, per ogni frame n del video, ovvero $\{\mathbf{B}_T^{1,n}, \dots, \mathbf{B}_T^{K,n}\}$, vengono calcolate una heatmap \mathbf{G}_n ed una posa \mathcal{L}_n che rappresentano globalmente la figura umana nell'immagine (figura 2.1). La heatmap \mathbf{G}_n sarà

$$\mathbf{G}_n = \frac{1}{K} \sum_{k=1}^K \mathbf{B}_T^{k,n},$$

mentre la posa \mathcal{L}_n sarà quell'insieme di punti dell'immagine $\{\mathbf{z}^{k,n}\}_{k=1}^K$ per i quali

$$\mathbf{z}^{k,n} = \underset{\mathbf{z} \in \mathcal{Z}}{\operatorname{argmax}} \{ \mathbf{B}_T^{k,n}(\mathcal{Y}_k = \mathbf{z}) \},$$

dove $\mathcal{Z} \in \mathbb{R}^2$ è l'insieme di tutti i punti dell'immagine.

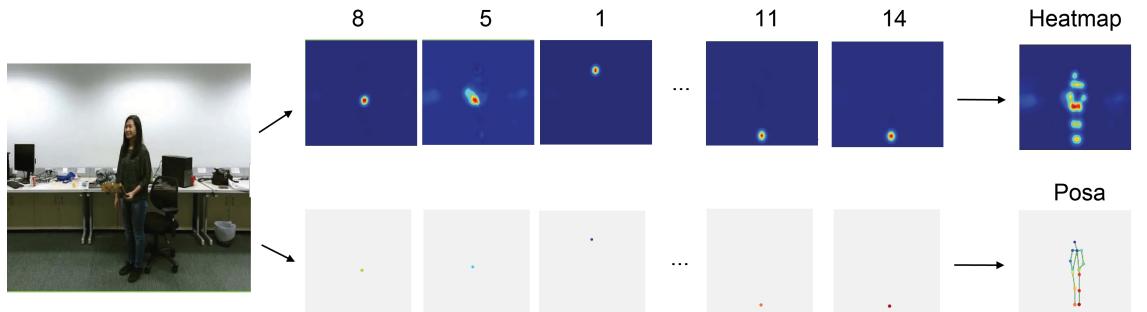


Figura 2.1: Processo di creazione della heatmap e della posa a partire da un singolo frame del video. Ogni giunto corporeo viene stimato singolarmente per poi essere successivamente combinato agli altri.

Quello che abbiamo fatto è stato cioè trasformare un video in un sequenza di heatmaps e pose, a questo punto dobbiamo però sintetizzarle in un unico elemento rappresentativo di più facile manipolazione.

Gli autori del lavoro mettono a confronto 3 diversi tipi di sintetizzazione:

- **sintetizzazione temporale delle heatmaps** - Data una sequenza di N heatmaps $\mathcal{V}_G = \{\mathbf{G}_1, \dots, \mathbf{G}_n, \dots, \mathbf{G}_N\}$ dove $\mathbf{G}_n \in \mathbb{R}^{P \times Q}$ è quella relativa all' n -esimo frame con P righe e Q colonne, allora la sequenza $\mathbf{G}_{1:n}$ può essere mappata in un vettore

$$\mathbf{v}_n = V\left(\frac{1}{n} \sum_{i=1}^n \mathbf{G}_i\right),$$

dove la funzione V trasforma una matrice in un vettore $\mathbf{v}_n \in \mathbb{R}^{(P \cdot Q) \times 1}$ (figura 2.2).

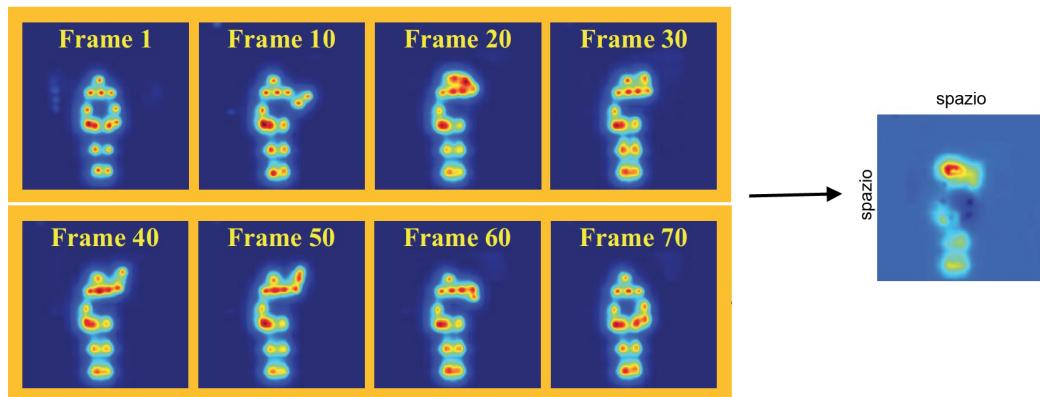


Figura 2.2: Sintetizzazione temporale delle heatmaps.

- **sintetizzazione spazio-temporale delle heatmaps** - Partizionando ogni heatmap \mathbf{G}_n in P righe, ovvero $\mathbf{G}_n = [(\mathbf{p}_1^T, \dots, \mathbf{p}_s^T, \dots, \mathbf{p}_P^T)^T]$ o in Q colonne, ovvero $\mathbf{G}_n = [(\mathbf{q}_1, \dots, \mathbf{q}_s, \dots, \mathbf{q}_Q)]$ ed usando la precedente funzione V per mappare $(\mathbf{p}_{1:s})^T$ e $\mathbf{q}_{1:s}$ in \mathbf{v}_s^P e \mathbf{v}_s^Q rispetivamente, si può ottenere due vettori $\mathbf{u}^P, \mathbf{u}^Q$ rappresentativi del frame attraverso la seguente funzione

obbiettivo:

$$\begin{aligned} \arg \min_{\mathbf{u}^{\eta}} \frac{1}{2} \|\mathbf{u}^{\eta}\|^2 + W \sum_{\forall i,j} v_{s_i}^{\eta} \succ v_{s_j}^{\eta} \epsilon_{ij}, \\ \text{s.t. } (\mathbf{u}^{\eta})^T \cdot (\mathbf{v}_{s_i}^{\eta} - \mathbf{v}_{s_j}^{\eta}) \geq 1 - \epsilon_{ij} \\ \epsilon_{ij} \geq 0 \end{aligned}$$

dove $\eta \in \{\mathbf{p}, \mathbf{q}\}$, $v_{s_i}^{\eta} \succ v_{s_j}^{\eta}$ indica la successione temporale fra $v_{s_i}^{\eta}$ e $v_{s_j}^{\eta}$, $\mathbf{u}^{\mathbf{p}} \in \mathbb{R}^Q$ e $\mathbf{u}^{\mathbf{q}} \in \mathbb{R}^P$. Per gli N frame vengono concatenati in ordine temporale i vettori così calcolati, ottenendo $\mathbf{U}^{\mathbf{p}} \in \mathbb{R}^{Q \times N}$ e $\mathbf{U}^{\mathbf{q}} \in \mathbb{R}^{P \times N}$. La matrice finale \mathbf{U} sarà l'unione di queste concatenazioni, ovvero $[(\mathbf{U}^{\mathbf{p}})^T, (\mathbf{U}^{\mathbf{q}})^T]^T$ ovvero una matrice che manterrà sia le caratteristiche temporali che spaziali del movimento ripreso nel video. L'intero processo è schematizzato in figura 2.3.

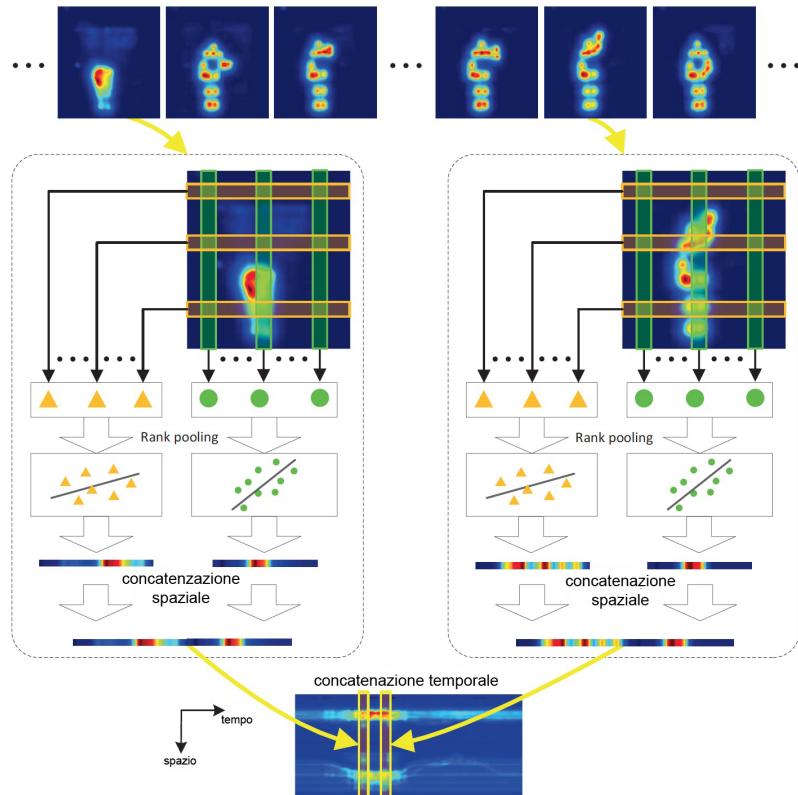


Figura 2.3: Sintetizzazione spazio-temporale delle heatmaps.

- **sintetizzazione spazio-temporale delle pose** - Per comprendere come viene creata questa sintetizzazione, supponiamo di avere una sequenza di pose $\mathcal{V}_L = \{\mathcal{L}_1, \dots, \mathcal{L}_n, \dots, \mathcal{L}_N\}$ dove $\mathcal{L}_n = \{z^{k,n}\}_{k=1}^K$ e $z^{k,n} = (x^{k,n}, y^{k,n})$, ovvero le coordinate orizzontali e verticali del k -esimo giunto corporeo. Per codificare le caratteristiche di tale sequenza, ogni posa viene trasformata in una successione di distanze fra giunti consecutivi, seguendo l'ordine rappresentato in figura 2.4. Ripetendo questo processo per ogni frame e concatenando i risultati ottenuti, si ottiene un'immagine rappresentativa delle caratteristiche spazio-temporali dell'evoluzione della posa nel video. L'intero processo è rappresentato in figura 2.4.

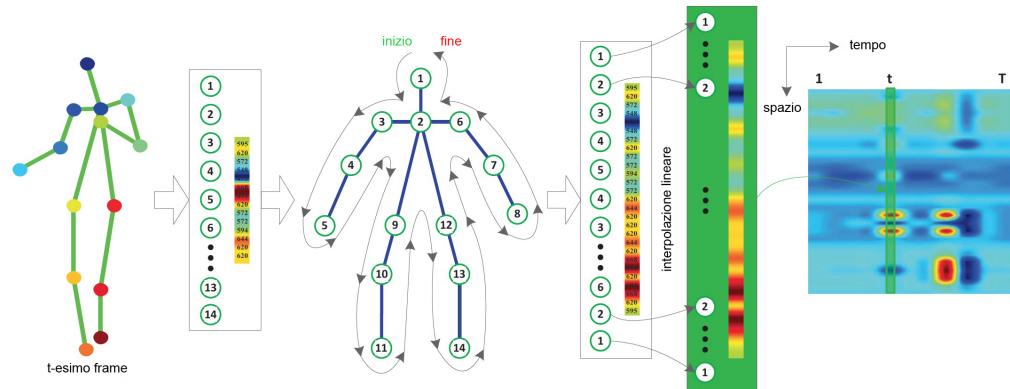


Figura 2.4: Sintetizzazione spazio-temporiale delle heatmaps.

Gli esperimenti effettuati dagli autori sono molteplici e una delle caratteristiche principali sulla quale è necessario porre attenzione è la telecamera usata per la creazione delle pose nei video: in alcuni casi hanno utilizzato la telecamera di profondità ed in altri no.

Uno dei dataset utilizzato nel lavoro è NTU-RGB60 [19], che verrà trattato in dettaglio nel capitolo 5 e che come vedremo propone due metodi di valutazione distinti: *cross-subject* e *cross-view*. I migliori risultati ottenuti in questo lavoro sono riassunti in tabella 2.1.

Telecamera	Accuratezza cross-view	Accuratezza cross-subject
normale	84.21%	78.80
di profondità	95.26%	91.71%

Tabella 2.1: Risultati ottenuti da “Recognizing Human Actions as the Evolution of Pose Estimation Maps - (2018)”

2.2 Actional-Structural Graph Convolutional Networks for Skeleton-based Action Recognition (2019)

Seconda spiegazione.

2.3 Vertex feature encoding and hierarchical temporal modelling in a spatial-temporal graph convolutional network for action recognition (2019)

Terza spiegazione.

Capitolo 3

Stima della posa

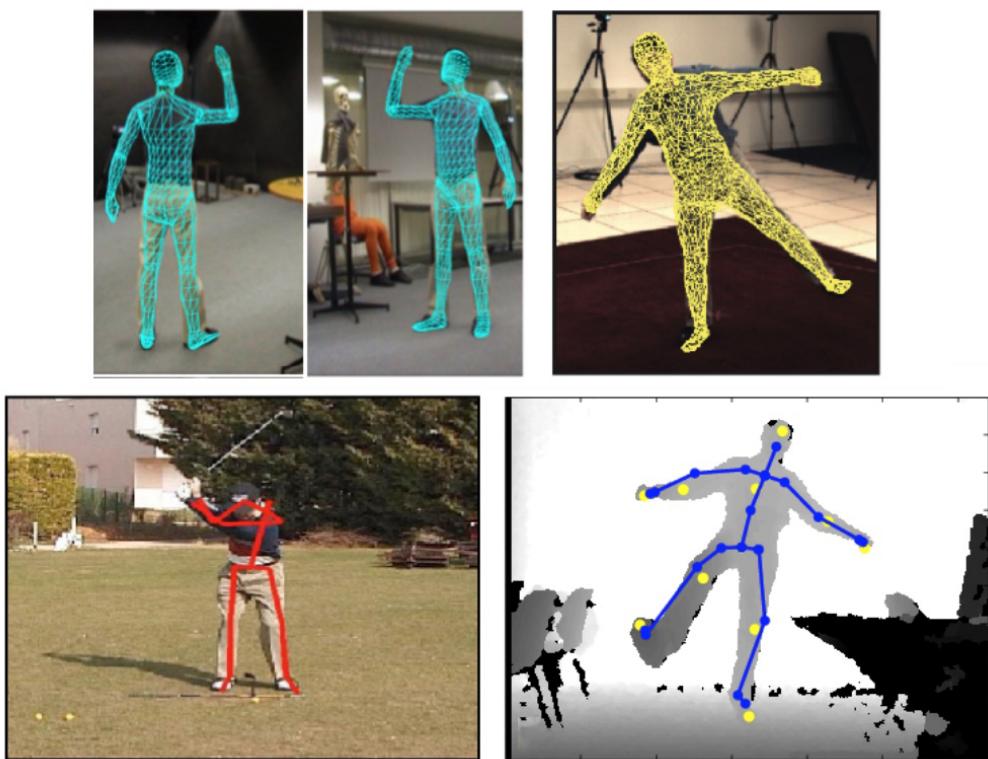


Figura 3.1: Esempi di stima della posa. In alto tre esempi di stima della posa utilizzando modelli di tipo volumetrico. In basso due esempi di stima della posa ottenuti utilizzando modelli di tipo scheletrici.

Cos è la stima della posa?

Quando parliamo di *stima della posa* ci riferiamo ad una tecnica di *computer vision* dedicata al riconoscimento di figure umane all'interno di video ed immagini, così da poter localizzare ad esempio dove, all'interno dell'immagine, si trova la testa, il braccio, la gamba destra, etc.. della persona inquadrata.

Questa tecnica non va assolutamente confusa con tecniche di riconoscimento di persone, infatti la stima della posa è in grado solo di riconoscere dove le parti del corpo di un individuo sono situate all'interno dell'immagine, non *chi* è inquadrato.

I campi di applicazione della stima della posa sono i più svariati: software interattivi che reagiscono al movimento della persona, robotica, realtà aumentata, animazione, fotoritocco intelligente, fitness, riabilitazione, etc. Stiamo parlando di un problema tutt'altro che semplice, infatti la condizione di luce dell'immagine, la variabilità dell'ambiente circostante, l'inclinazione del soggetto inquadrato, rendono il riconoscimento della posa un problema non affatto banale.

Spinti dal crescente interesse, negli ultimi anni sono stati sviluppati diversi algoritmi per la stima della posa, raggiungendo in molti casi risultati davvero sorprendenti con un'accuratezza prossima alla perfezione.



Figura 3.2: Un esempio di utilizzo in campo medico della stima della posa

La maggior parte dei software in circolazione in grado di stimare in maniera

sufficientemente corretta la posa di un individuo non sono liberamente accessibili. Due fra i migliori algoritmi (ad oggi) di *pose detection* sono sicuramente *Posenet* [1] e *Detectron 2* [4], dei quali ci occuperemo in maniera più approfondita nei capitoli seguenti.

3.1 PoseNet

I recenti progressi nel campo della visione artificiale hanno permesso alla comunità scientifica di spostarsi verso problemi ancora più articolati rispetto a quelli classici, con l'obiettivo di riconoscere figure umane in contesti non vincolati e molto variabili.

L'algoritmo *PoseNet* è stato ideato proprio con lo scopo di identificare una o più figure umane in qualsiasi contesto, anche in quelli “affollati”, ed essere in grado di identificare ogni persona stimandone i suoi *punti chiave* (o *key-points*).

Generalmente esistono due approcci principali per affrontare il problema del rilevamento di più persone in un'immagine, la stima della loro posa e la loro *segmentazione* (ovvero l'identificazione dei pixel che rappresentano ogni persona):

- l'approccio *top-down* inizia identificando e localizzando approssimativamente le singole istanze di persona delimitando il riquadro dell'immagine dentro il quale sono contenute, seguito da una fase di stima della posa o di separazione “primo piano-sfondo” nell'area identificata.
- Al contrario, l'approccio *bottom-up* inizia localizzando *entità semantiche individuali*, come ad esempio gambe, braccia, mani, etc, seguito dal loro raggruppamento in istanze di persone complete. PoseNet adotta questo secondo approccio.

Inoltre PoseNet utilizza una rete neurale convoluzionale per la quale il costo computazionale del riconoscimento delle pose è essenzialmente indipendente dal numero di persone raffigurate nella scena ma dipende esclusivamente dalla scelta delle features della rete.

L'approccio adottato in PoseNet è quello di identificare dapprima tutti i punti chiave e successivamente raggrupparli in istanze di persona utilizzando un processo “greedy”, ovvero partendo dal rilevamento “più sicuro” (e non come spesso

accade, da un punto fisso di riferimento, ad esempio il naso). Anche se potrebbe sembrare una tecnica più "disordinata" i risultati empirici hanno dimostrato funzionare estremamente bene.

Oltre a stimare punti chiave sparsi, PoseNet stima anche delle maschere di segmentazione per ogni persona. Per fare ciò, viene allenata una seconda rete neurale con la quale viene associato ad ogni pixel x_i la probabilità di appartenenza di quel pixel ad ogni candidato j identificato. Se la probabilità è sufficientemente alta allora viene associato il pixel x_i al candidato j .

Questo algoritmo è stato allenato utilizzando il famoso dataset COCO [2] (versione 2016) che, fra molte altre cose, contiene anche l'annotazione dei 17 punti chiave (12 del corpo e 5 del volto) di migliaia di persone ed è riuscito a migliorarne l'*AP* (average-precision) da 0,655 a 0,687 diventandone il miglior risultato.

Essendo molto semplice questo metodo è anche quindi molto rapido, poiché non richiede alcuna fase supplementare di raffinamento dei risultati con tecniche di tipo *box-based* o *clustering*, rendendo di fatto PoseNet uno degli algoritmi più facilmente installabili su piccoli dispositivi, come ad esempio i cellulari.

Ma vediamo adesso più nel dettaglio come PoseNet stima e raggruppa i punti chiavi di una o più persone raffigurate in un'immagine.

3.1.1 Stima dei key-points

L'obiettivo di questa fase è quello di rilevare, in modo indipendente dall'istanza, tutti i key-points visibili appartenenti a qualsiasi persona dell'immagine. A tale scopo vengono prodotte delle *heat-maps*, ovvero dei canali della rete neurale dedicati al riconoscimento di particolari caratteristiche dell'immagine (una canale per ogni key-point) e degli *offset* (due canali per ogni key-point per gli spostamenti in orizzontale e verticale). Sia x_i la posizione 2-D nell'immagine, dove $i = 1, \dots, N$ e N è il numero di pixels; $D_R(y) = \{x : \|x - y\| \leq R\}$ un disco di raggio R centrato in y e $y_{j,k}$ la posizione 2-D del k -esimo key-point della j -esima istanza di persona, con $j = 1, \dots, M$, dove M è il numero di istanze nell'immagine.

Per ogni tipo di key-point $k = 1, \dots, K$, viene impostato un task di classificazione binaria come segue. Viene generata una heat-map $p_k(x)$ tale che $p_k(x) = 1$

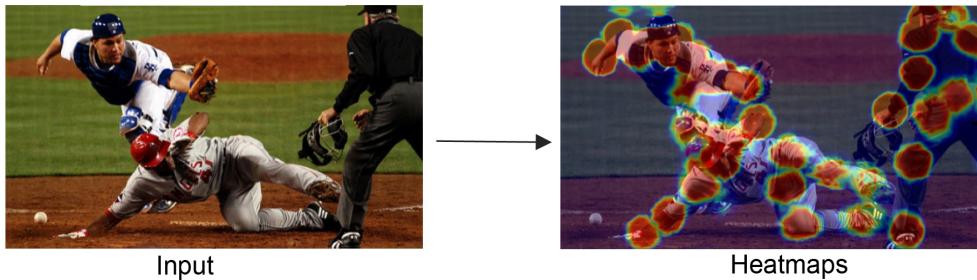


Figura 3.3: Generazione con PoseNet delle heat-maps per ogni tipologia di key-point

se $x \in D_R(y_{j,k})$ per qualsiasi istanza j , altrimenti $p_k(x) = 0$. Abbiamo quindi K tasks di classificazione binaria indipendenti (uno per ogni tipo di key-point) e ciascuno equivale a prevedere un disco di raggio R attorno a un tipo di key-point specifico appartenente a qualsiasi persona nell'immagine.

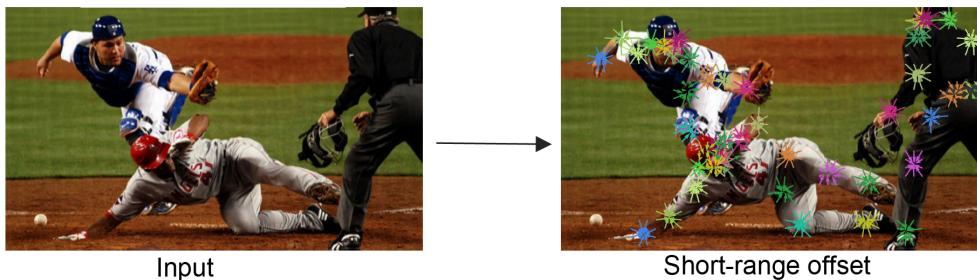


Figura 3.4: Esempio di stima degli offset a corto raggio con PoseNet

Oltre alle heat-maps, vengono anche usati vettori di *offset a corto raggio* $S_k(x)$ il cui scopo è quello di migliorare l'accuratezza della localizzazione dei key-points. Per ogni punto x all'interno dei dischi ricavati al passo precedente, il vettore di offset 2-D a corto raggio $S_k(x) = y_{j,k} - x$ rappresenta la distanza fra il punto x e il k -esimo key-point della j -esima persona. Vengono così generati K vettori per ogni punto x all'interno del disco definito che, combinati insieme in una *trasformata di Hough* $h_k(x)$, miglioreranno l'accuratezza della posizione predetta di ogni key-point. Solo i punti che superano una certa soglia di Hough (0.01, come indicato da [1]) vengono considerati dei key-point, gli altri invece vengono scartati.

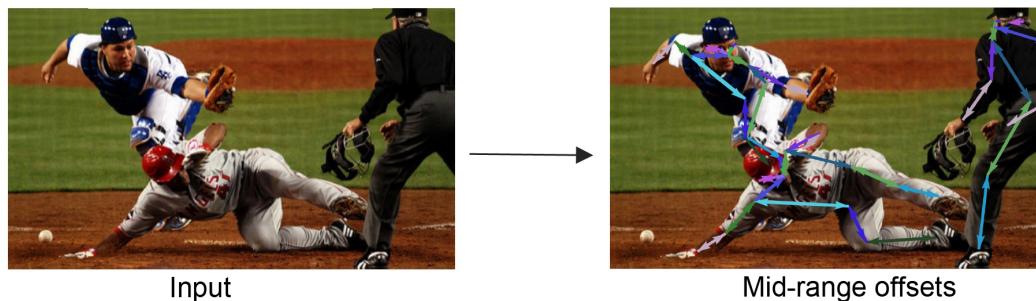


Figura 3.5: Esempio di stima degli offset a medio raggio con PoseNet. L'intento è quello di raggruppare i keypoints appartenenti alla stessa persona.

3.1.2 Raggruppamento dei key-points in istanze di persona

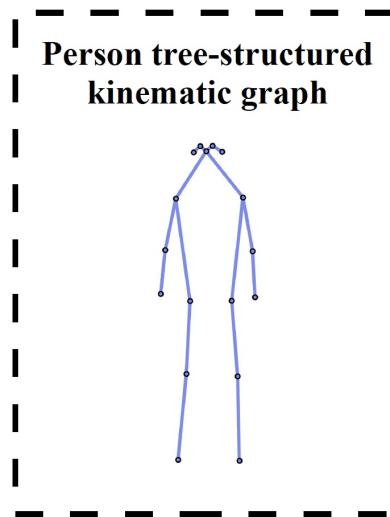


Figura 3.6: Struttura ad albero utilizzata da PoseNet per raggruppare i key-points appartenenti alla stessa persona

A questo punto è però necessario capire come associare ogni key-point alle persone raffigurate nell'immagine (nel caso ce ne sia più di una). Seguendo lo schema delle connessioni fra tipi di key-points (rappresentati in figura 3.6) la rete viene allenata per restituire in output anche i cosiddetti *offset a medio raggio*,

ovvero probabilità di connessioni fra key-points, col lo scopo di raggruppare quelli appartenenti alla stessa persona.

Un esempio di questa stima è raffigurato in figura 3.5 mentre una raffigurazione completa del sistema adottato da PoseNet per il riconoscimento delle pose di persone raffigurate in un'immagine è rappresentato in figura 3.7.

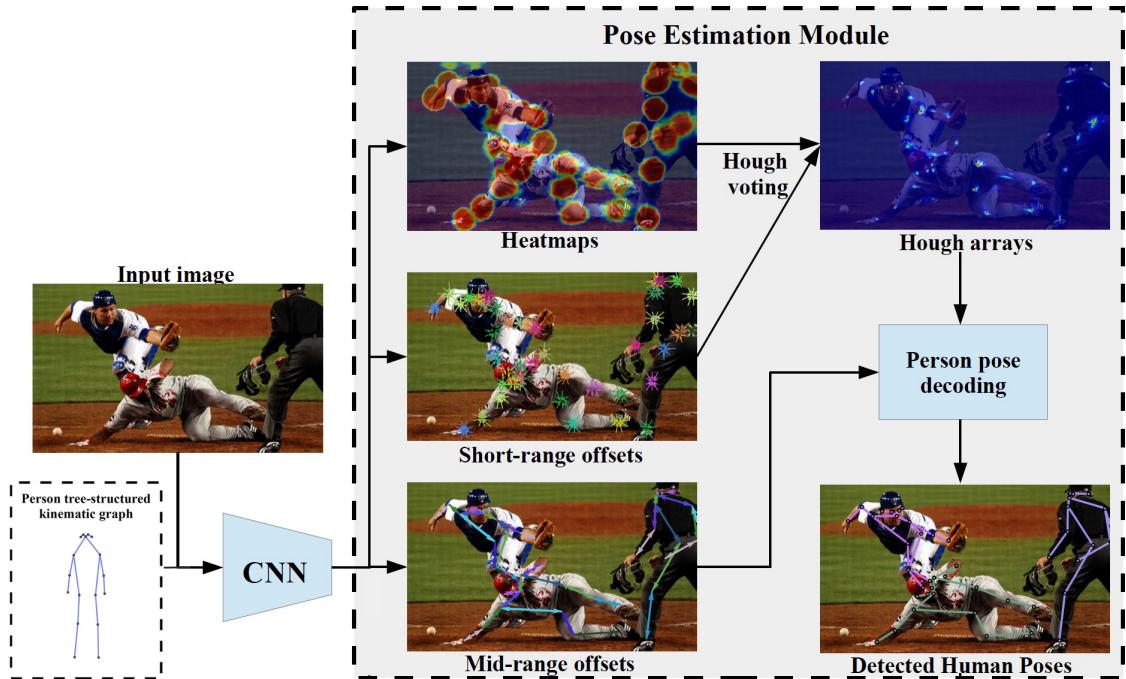


Figura 3.7: Combinazione delle fasi adottate da PoseNet per il riconoscimento della posa in un'immagine.

3.2 Detectron2

Detectron2 è un progetto open-source lanciato dalla *Facebook AI Research (FAIR)* ampiamente usato dalla comunità di ricerca in ambito *computer vision* e rappresenta, ad oggi, una piattaforma per il riconoscimento di oggetti allo stato dell'arte.

Il suo predecessore *Detectron* [5] fu un progetto iniziato nel 2016 con l'obiettivo di creare un sistema rapido e flessibile per il riconoscimento di oggetti in

immagini ed originariamente basato su *Caffe2* [6] (un framework ideato per facilitare la sperimentazione e la divulgazione di nuovi modelli e algoritmi in ambito *deep learning*) e scritto in *Python*.

Negli anni Detectron è stato perfezionato e supportato da una grande quantità di progetti, compreso “*Mask-R-CNN*” [7] e “*Focal Loss for Dense Object Detection*” [8], vincitori rispettivamente del *Premio Marr* e di *Miglior articolo scientifico studentesco* all’*International Conference on Computer Vision* (ICCV) del 2017. L’intuitività e l’efficacia di questi algoritmi hanno permesso un notevole sviluppo nella risoluzione di problemi complessi nell’ambito della computer vision, come ad esempio l’*instance segmentation*, e hanno sicuramente giocato un ruolo rilevante nell’avanzamento tecnologico dei sistemi di riconoscimento visivo.

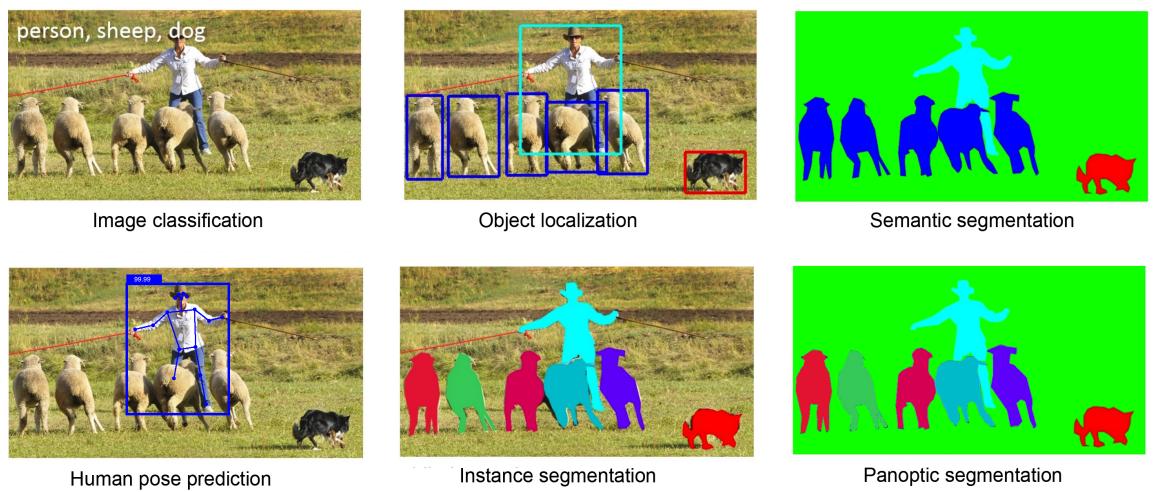


Figura 3.8: Tipologie di analisi visive.

Detectron2 è adesso basato su *Pytorch*, una libreria open-source dedita al machine learning ed ampiamente usata nel campo della computer-vision che ha inglobato in se anche il precedente framework Caffe2. Più nello specifico Detectron2 include ad oggi le implementazioni dei seguenti algoritmi di object-detection:

- Cascade R-CNN [16]
- Panoptic FPN [17]

- TensorMask [18]
- Mask R-CNN [7]
- RetinaNet [8]
- Faster R-CNN [9]
- RPN [9]
- Fast R-CNN [10]
- R-FCN [11]

utilizzando le seguenti reti *backbone* (ovvero reti precedentemente allenate con lo scopo di estrarre in maniera efficiente le *features* di un'immagine):

- ResNeXt{50,101,152} [12]
- ResNet{50,101,152} [13]
- Feature Pyramid Networks (con ResNet/ResNeXt) [14]
- VGG16 [15]

Inoltre, nel caso fosse necessario implementare nuove reti backbone, è possibile farlo facilmente grazie alla struttura modulare di Pytorch, che permette di separare il nuovo modello dai precedenti algoritmi di Detectron2.

Essendo stato interamente riscritto in Pytorch, Detectron2 è più rapido del suo predecessore nei compiti di *object-detection*, *instance segmentation* e *human-pose prediction*, ed in più è in grado di fornire supporto per i nuovi task di *semantic segmentation* e *panoptic segmentation*, ovvero la combinazione fra instance-segmentation e semantic-segmentation (figura 3.8).

Oltre che nella ricerca, questa piattaforma viene usata anche da numerosi team di Facebook (e non solo) per l'addestramento di nuovi modelli in svariati campi della *computer vision*, come ad esempio la *realtà aumentata*, e in materia di sicurezza informatica, come ad esempio la *community integrity*, ovvero la difesa e la protezione di account su piattaforme social da contenuti maligni.

Per quanto riguarda la stima della posa umana in un'immagine, Detectron2 utilizza una Mask R-CNN [7] riadattata all'estrazione dei keypoints.

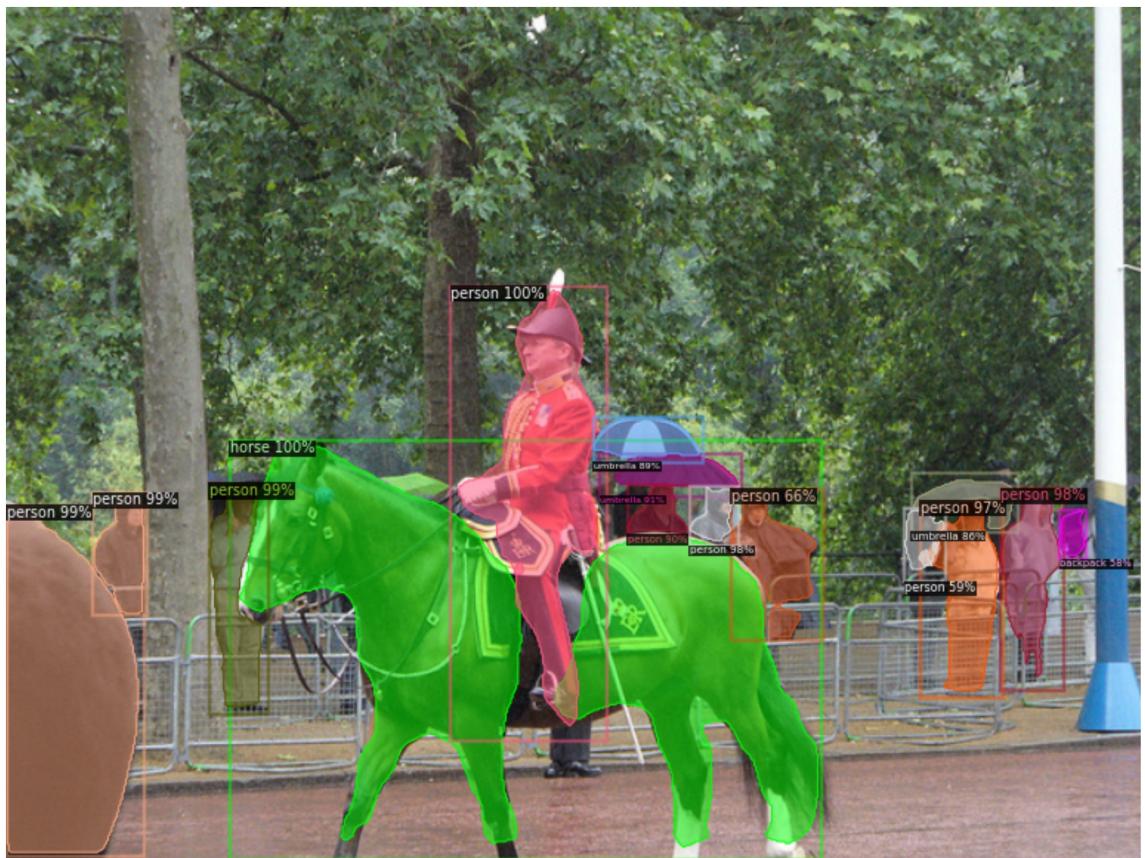


Figura 3.9: Instance segmentation con Detectron2

3.2.1 Mask R-CNN

Mask R-CNN nasce con l'intento di creare un framework semplice e flessibile per affrontare il problema dell'*instance segmentation*. Questo metodo è in grado di identificare gli oggetti in un'immagine e simultaneamente generare una maschera di segmentazione ben definita per ogni istanza. Mask R-CNN è sostanzialmente un'estensione di *Faster R-CNN* [9] in quanto aggiunge parallelamente ai due output già presenti per il *bounding box* e la classificazione anche quello per la predizione della maschera d'istanza. Nonostante sia un metodo nato per l'instance segmentation è però facilmente adattabile ad altri tipi di predizioni, come ad esempio quella della posa umana.

Mask R-CNN è suddiviso in due fasi di procedura. La prima, identica a quella adottata per Faster R-CNN, chiamata “*Region Proposal Network*” (RPN)



Figura 3.10: Alcuni esempi di instance segmentation con Mask R-CNN

ha il compito di proporre porzioni di immagine nelle quali potrebbero essere raffigurate delle istanze. Nella seconda, viene predetto *in parallelo* alla classe e al delineamento preciso dell'istanza (*bounding box*) anche una maschera binaria per ogni *regione d'interesse* (RoI) dell'immagine.

Più precisamente, durante l'allenamento della rete, la *loss* definita per ogni RoI è la seguente: $L = L_{cls} + L_{box} + L_{mask}$. La loss di classificazione e quella del boundig box sono identiche a quelle definite per Fast R-CNN [10].

Per quanto riguarda invece L_{mask} , dato che decodifica K maschere binarie di dimensione $m \times m$ (dove K è il numero di classi), la sua dimensione è Km^2 per ogni RoI.

Questa definizione di L_{mask} permette alla rete, non solo di generare maschere completamente indipendenti l'una dall'altra, ma anche di disaccoppiare la predizione delle maschere dalla classificazione delle istanze. Questa caratteristica, innovativa rispetto alle pratiche comuni in materia di semantic segmentation, si è rivelata essere basilare per il raggiungimento di una buona segmentazione.

3.2.2 Predizione della posa con Mask R-CNN

Grazie alla sua grande flessibilità, questo framework può essere facilmente esteso alla stima della posa umana in un'immagine.



Figura 3.11: Alcuni esempi di pose prediction con Mask R-CNN

Le coordinate dei keypoints vengono trasformate in maschere di tipo *one-hot* e Mask R-CNN viene utilizzato per predire K maschere, una per ogni tipo di keypoint (gomito sinistro, spalla destra, etc..).

Più nello specifico, per ognuno dei K keypoint di un'istanza, il training target è una maschera binaria $m \times m$ di tipo *one-hot* dove cioè solo un pixel viene etichettato come positivo. Durante l'allenamento della rete, per ogni keypoint visibile nell'immagine, viene minimizzata una *cross-entropy* loss con regolarizzatore L2 (per incoraggiare l'identificazione di un singolo punto). Anche per questa procedura, le K maschere dei K tipi di keypoint sono completamente indipendenti l'una dall'altra.

Capitolo 4

Metodo proposto

In questo capitolo oltre ad approfondire le diverse tecniche adottate nel lavoro di tesi, verrà illustrato anche l'iter procedurale che ha portato ai risultati ottenuti. Come vedremo nel capitolo seguente, il dataset utilizzato è NTU-RGB60 [19] e nonostante fornisca anche i dati relativi alla posa dei soggetti inquadrati (quello che viene spesso indicato come *skeleton*) ho preferito non farne assolutamente uso ed utilizzare invece i soli video RGB, dai quali successivamente stimare la posa e servirmi di quest'ultima per il riconoscimento delle azioni svolte. Questa scelta è stata presa con lo scopo di favorire una maggiore generalizzazione e applicazione dell'algoritmo, adoperabile in contesti più diffusi dove la stima della posa non è data, ma abbiamo a disposizione solo un comuniissimo video RGB. Passiamo adesso ad analizzare una ad una le varie fasi affrontate nel lavoro di tesi.

4.1 Estrazione delle pose

Per prima cosa è necessario convertire tutti i video del dataset in sequenze di pose, in modo da sintetizzarne efficacemente il movimento umano senza un'eccessiva perdita d'informazione. Gli algoritmi scelti per questa fase sono stati Detectron2 [4] per la sua rimarcabile accuratezza e PoseNet [1] per la sua velocità d'inferenza e portabilità. I modelli proposti da queste due tecniche sono molteplici. Per Detectron2 abbiamo:

- *R50-FPN-1x*
- *R50-FPN-3x*
- *R101-FPN-3x*
- *X101-FPN-3x*

mentre per PoseNet abbiamo:

- *PoseNet-50*
- *PoseNet-75*
- *PoseNet-100*
- *PoseNet-101*

Ognuno di questi modelli ha valori diversi in accuratezza e velocità di inferenza. Nel caso di Detectron2 questa comparazione viene fornita dagli autori stessi e schematizzata in tabella 4.1, mentre per quanto riguarda PoseNet non viene purtroppo fornito un confronto simile.

Nome	Tempo di inferenza (s/imm)	Box AP	KP AP
R50-FPN-1x	0.072	53.6	64.0
R50-FPN-3x	0.066	55.4	65.5
R101-FPN-3x	0.076	56.4	66.1
X101-FPN-3x	0.121	57.3	66.0

Tabella 4.1: Diversi modelli in Detectron 2 per l'estrazione della posa da un'immagine. Il tempo di inferenza viene misurato in *secondi/immagine*; Box AP = “Bounding box average precision” ; KP AP = “Keypoint average precision”

È stato quindi scelto il modello *X101-FPN* per Detectron2, in quanto quello con il miglior valore di accuratezza globale e PoseNet-101, per una più facile comparazione dei risultati.

4.2 Assegnazione delle pose

Quello che otteniamo dalla fase precedente è, per ogni video, una sequenza di pose umane estrapolate da ogni frame. Sia Detectron2 che PoseNet sono in grado di riconoscere più persone all'interno di un'immagine, producendo una lista di pose P_1, P_2, \dots, P_n ordinata secondo le regole dell'algoritmo, ad esempio Detectron2 le ordina per valore di *score* decrescente, ovvero quel valore che indica quanto l'algoritmo sia sicuro di aver correttamente stimato la posa di quella persona.

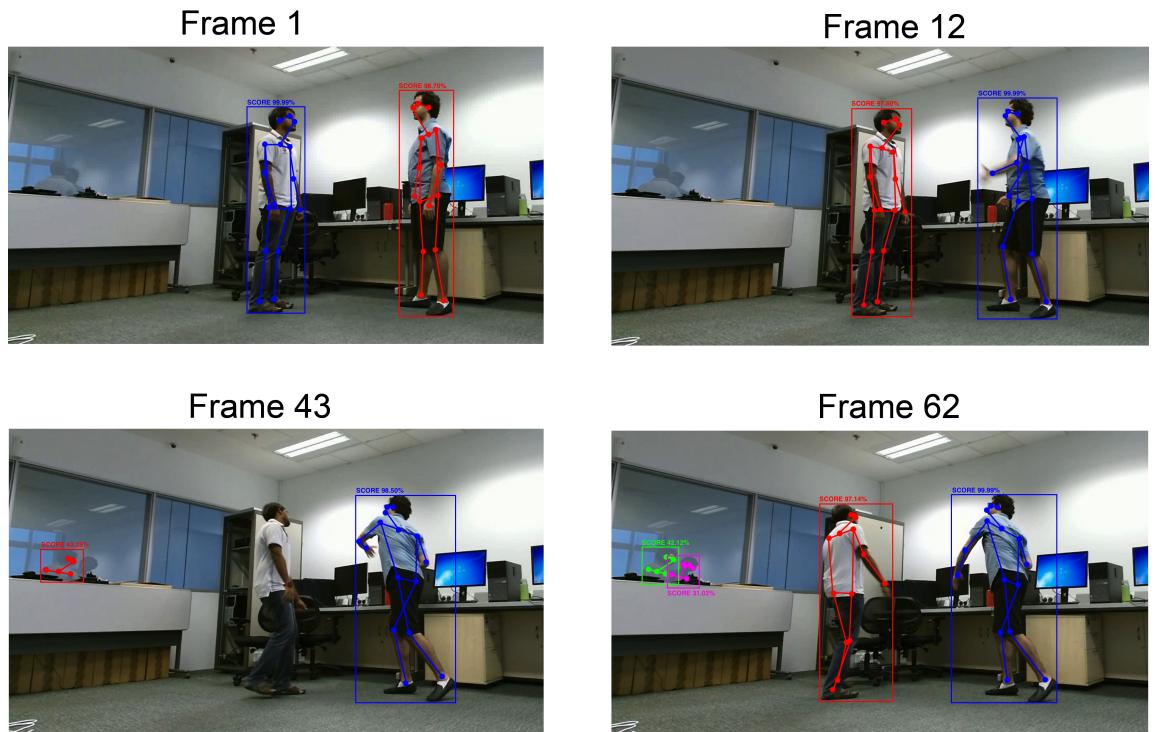


Figura 4.1: Un esempio di assegnazione inconsistente delle pose lungo il video. Nell'esempio, l'ordine delle pose restituite dall'algoritmo è blu, rosso, verde e viola. Quest'ordine è indipendente dai soggetti inquadrati, che possono talvolta essere scambiati fra loro, non essere riconosciuti o addirittura scambiati per dei riflessi.

Quello che però questi algoritmi non hanno è una conoscenza relativa al concetto di video, ovvero alla successione logica dei frame e trattano ognuno di essi

come un'immagine indipendente. Quello che ne segue è una totale indipendenza fra l'ordine delle pose riconosciute e la loro assegnazione consistente ai soggetti rappresentati nel video. Ad esempio, ammettiamo di avere due soggetti A e B rappresentati in un video e che per l' i -esimo frame vengano identificate due pose $\mathcal{P}_{1,i}$ e $\mathcal{P}_{2,i}$ appartenenti ad A e B rispettivamente. Quest'ordine non è però garantito nel frame successivo dove $\mathcal{P}_{1,i+1}$ potrebbe essere la posa di B e $\mathcal{P}_{2,i+1}$ quella di A .

Per ovviare a questo problema la tecnica adottata è stata quella di calcolare iterativamente ad ogni frame la distanza fra l'ultima posa assegnata ad ogni soggetto e tutte quelle riconosciute al frame successivo. La distanza fra due pose non è altro che la somma delle distanze euclidiene fra i punti in comune delle due pose, ovvero se $\mathcal{P}_i = \{p_{i,k}\}_{k=1}^K$ è l'insieme dei punti che definiscono la posa i , dove ogni $p_{i,k} = (x_{i,k}, y_{i,k})$ sono le ascisse e le ordinate del k -esimo giunto e K il numero di tipi di giunto, allora la distanza fra P_i e P_j sarà:

$$\mathcal{D}_{ij} = \sum_{k=1}^K I(p_{i,k}) \cdot I(p_{j,k}) \cdot eucl(p_{i,k}, p_{j,k})$$

dove I è la funzione identità, ovvero $I(p) = 1$ se p è definito, 0 altrimenti e $eucl(p_1, p_2)$ è la funzione di distanza euclidea fra i punti p_1 e p_2 . Se le due pose non hanno punti in comune, ovvero $\sum_{k=1}^K I(p_{i,k}) \cdot I(p_{j,k}) = 0$ allora $\mathcal{D}_{ij} = \infty$.

Una volta ottenuta ed ordinata in maniera decrescente la lista delle distanze fra pose, si può procedere alla loro assegnazione ai soggetti inquadrati e ripetere il procedimento per il frame successivo.

Abbiamo però un ultimo problema da fronteggiare in questa fase, ovvero la variabilità del numero di pose stimate in ogni frame del video. Questo accade ad esempio, quando per uno o più frame uno dei soggetti non viene riconosciuto dall'algoritmo o quando l'improvviso riflesso su una finestra o uno specchio viene interpretato come una nuova persona, come raffigurato in figura 4.1.

Questo problema è stato risolto ordinando le sequenze di pose secondo il loro score medio lungo tutto il video. Si presuppone infatti che un riflesso sia mediamente meno convincente di una persona vera e propria. Una volta ordinate le sequenze, sono state mantenute solo le migliori due e scartate le altre, visto che siamo interessati ad analizzare solo azioni individuali o di coppia.

4.3 Rimozione degli zeri

Come abbiamo appena visto, PoseNet e Detectron2 talvolta in qualche frame possono non riconoscere un'intera persona o non riuscire a stimare la posizione di qualche giunto, se ad esempio coperti o sfocati. Entrambi gli algoritmi trattano questi 2 casi nella stessa maniera, ovvero assegnano il punto $(0, 0)$ ad ogni giunto non riconosciuto.

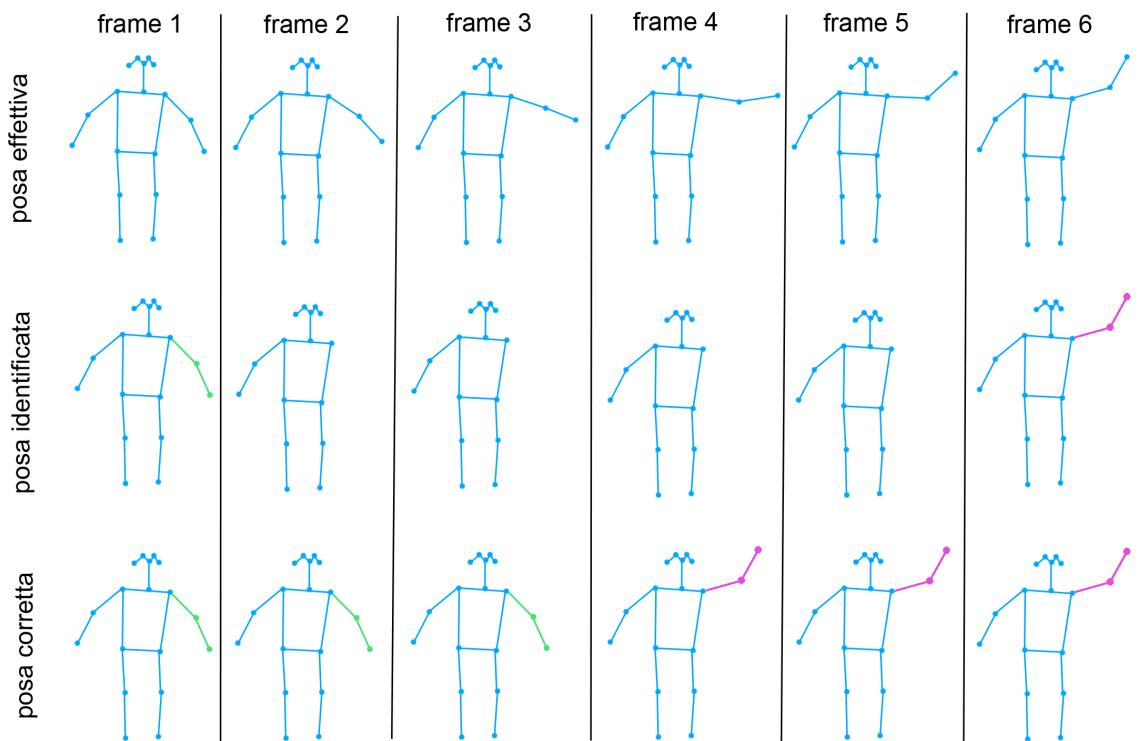


Figura 4.2: Esempio di rimozione degli zeri. Nei frame 2-3-4-5 i giunti del braccio sinistro non sono stati riconosciuti. Nei frame 2-3 verranno quindi rimpiazzati con quelli del frame 1, mentre nei frame 4-5 con quelli del frame 6.

Questo valore però è fortemente fuorviante considerando che le informazioni a nostra disposizione sono solo pochi punti corporei. Con Detectron2 e PoseNet la posa ottenuta è composta di soli 17 punti, quindi per ogni giunto non riconosciuto non solo perdiamo $1/17$ d'informazione ma confondiamo anche il nostro

algoritmo facendogli analizzare un movimento inesistente, per quel giunto, verso il punto (0,0).

È necessario quindi adottare una buona tecnica di pulizia degli zeri che limiti la perdita d'informazione mantenendo una consistenza con i risultati ottenuti.

La cosa più semplice da fare è assegnare ad ogni giunto non riconosciuto la sua ultima posizione identificata, ma così facendo utilizzeremmo solo il passato del video in questione e per lunghe sequenze di valori mancanti potremmo perdere molta informazione utile. È stato quindi scelto di attribuire ad ogni giunto non identificato il primo punto identificato più vicino nel tempo, passato o futuro, per quel giunto.

L'intento di questa tecnica è quello di ridurre la distanza fra ogni giunto non identificato e la loro effettiva posizione nell'immagine. Un caso esempio è rappresentato in figura 4.2

[PARLA DI RIMOZIONE DEGLI ZERI IN MANIERA INTERPOLANTE SE RIESCI AD INCUDERLO NEL LAVORO]

- tecniche:
 - centro video
 - centro frame
 - 5 Baricentri video
 - 3 Baricentri frame
 - 3 Baricentri video
 - 3 Baricentri video ABS
 - 17 baricentri video (uno per ogni keypoint)
 - 17 baricentri video ABS
 - next frame
 - next frame 5
 - next frame 11
 - next frame 17
 - distanze cumulative

- distanze fra punti
- tipi di normalizzazione - Classica o Indipendente
- dropout e dropout ricorrente
- smussamento
- layers
- LSTM e CudNNLSTM?
- regolarizzatori?

Capitolo 5

Il dataset NTU RGB+D

Il recente sviluppo dei sensori di profondità ha permesso un notevole miglioramento degli studi in ambito 3D, come ad esempio il riconoscimento di oggetti o azioni in scenari tridimensionali.

Essendo però un campo scientifico piuttosto “giovane”, la ricerca di un adeguato dataset per il *riconoscimento di azioni umane* che sia solido, voluminoso e vario può non essere un compito facile.

Spesso i dataset pubblicamente disponibili sono composti da uno stretto gruppo di soggetti, il che riduce notevolmente la variabilità intra-classi e un’attività *umana* dipende fortemente dall’età, il sesso, la cultura e le condizioni fisiche di chi la svolge quindi avere una sufficiente variabilità di soggetti nel dataset che stiamo analizzando è di vitale importanza.

Un’altra cosa estremamente importante è il numero di azioni che stiamo analizzando. Se il nostro dataset contiene poche tipologie di attività umane, sarà facile distinguerle e classificarle fra loro, magari identificando un semplice pattern di movimento o addirittura la struttura di un oggetto coinvolto nell’azione da classificare. Se invece il numero di azioni trattate è sufficientemente alto, allora i pattern di movimento e gli oggetti con i quali si interagisce vengono condivisi fra più classi di azioni, rendendo la classificazione estremamente più difficile.

Il terzo punto da considerare nella scelta di un buon dataset è il punto di vista dal quale vengono riprese le azioni. La maggior parte dei dataset contengono video che riprendono l’azione da un unico punto di vista (spesso frontale), in

altri casi invece i punti di vista sono strettamente due (magari frontale e di lato), ottenuti solo perchè sono state utilizzate più telecamere contemporaneamente.

Come ultima cosa e forse la più importante, l'insufficienza di video nei dataset ci impedisce spesso di applicare tecniche di machine-learning al nostro problema, portandoci inevitabilmente ad una situazione di overfitting.

Consapevoli di quanto detto finora Amir Shahroudy, Jun Liu, Tian-Tsong Ng e Gang Wang hanno creato nel 2016 *NTU RGB+D* [19], un vasto e variegato dataset dedicato all'analisi di attività umane in ambito 3D.

NTU RGB+D si compone di 56880 RGB+D video, ottenuti riprendendo 40 differenti attori ed utilizzando Microsoft Kinect v2. Il dataset contiene i video RGB, le sequenze di profondità, lo skeleton dei soggetti (ovvero le posizioni 3D dei principali giunti corporei) e i frames a infrarossi. I soggetti sono stati ripresi da 80 punti di vista differenti e la loro età varia dai 10 ai 35 anni, il che rende più realistica la variazione di qualità delle azioni svolte. Anche se tutte le riprese sono di tipo *indoor*, gli scenari ricreati intorno agli attori sono estremamente variabili.

5.1 NTU RGB+D in dettaglio

Tipologie di dati - I dati raccolti sono stati ottenuti con sensori Microsoft Kinect v2 che consentono di estrapolare dalle riprese 4 tipologie di dato: mappe di profondità, informazioni 3D dei giunti corporei, frame RGB e sequenze a infrarossi.

Le *mappe di profondità* sono sequenze di valori in millimetri che rappresentano la distanza di ogni punto dalla telecamera. Ogni mappa di profondità è stata poi ridotta ad una risoluzione di 512×424 senza perdita di informazione.

Le *informazioni 3D dei giunti corporei* sono coordinate 3D dei 25 giunti corporei più importanti. Per ogni giunto e per ogni frame viene inoltre salvato il corrispondente pixel nel video RGB e la relativa mappa di profondità. I 25 giunti corporei utilizzati sono raffigurati in figura 5.2.

I *video RGB* sono normali video registrati con una risoluzione 1920×1080 .

Le *sequenze a infrarossi* sono, come per le mappe di profondità, salvate frame per frame con una dimensione di 512×424 .



Figura 5.1: Alcuni esempi di frames del dataset NTU RGB+D. Nelle prime 4 righe è possibile notare la varietà degli attori partecipanti al progetto e dei diversi punti di ripresa utilizzati. La quinta riga mostra la varietà intra-classe per una stessa azione. L'ultima riga mostra per uno stesso frame i valori RGB, i valori RGB + giunti, mappa di profondità, mappa di profondità + giunti e i valori a infrarossi.

Le classi delle azioni - Il dataset contiene 60 tipi di azioni differenti divise in 3 gruppi principali: 40 azioni *quotidiane* (bere, mangiare, leggere, etc.), 9 azioni *legale alla salute* (starnutire, barcollare, svenire, etc...) e 11 azioni *di coppia* (dare un cazzotto a qualcuno, abbracciarsi, etc...)

I soggetti - Per la realizzazione di questo dataset sono stati utilizzati 40 attori di età compresa dai 10 ai 35 anni. In figura 5.1 è possibile vedere come l'insieme delle persone scelte sia estremamente variegato in età, altezza e sesso.

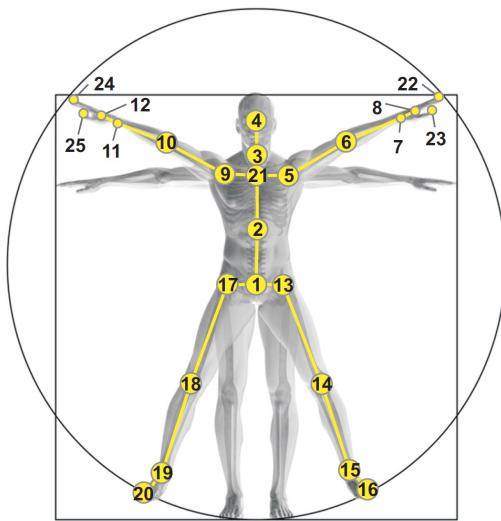


Figura 5.2: La configurazione dei 25 giunti utilizzati in NTU RGB+D. Le etichette utilizzate per i giunti sono: 1 - base della colonna vettrebrale, 2 - punto medio della colonna vertebrale, 3 - collo, 4 - testa, 5 - spalla sinistra, 6 - gomito sinistro, 7 - polso sinistro, 8 - mano sinistra, 9 - spalla destra, 10 - gomito destro, 11 - polso destro, 12 - mano destra, 13 - anca sinistra, 14 - ginocchio sinistro, 15 - caviglia sinistra, 16 - piede sinistro, 17 - anca destra, 18 - ginocchio destro, 19 - caviglia destra, 20 - piede destro, 21 - colonna vertebrale, 22 - punta della mano sinistra, 23 - pollice sinistro, 24 - punta della mano destra, 25 - pollice destro.

Ogni attore è rappresentato con un ID unico in tutto il dataset.

I punti di ripresa - Ogni azione è stata registrata da 3 telecamere contemporaneamente di modo da generare 3 prospettive diverse della stessa azione. Le 3 telecamere sono state posizionate tutte alla stessa altezza e sempre con la stessa angolazione di ripresa, ovvero -45° , 0° e 45° .

Ad ogni attore è stato inoltre richiesto di ripetere l'azione due volte: una rivolto verso la telecamera di sinistra ed una verso quella di destra. Così facendo le 6 riprese ottenute forniscono 5 prospettive diverse della stessa azione, ovvero quella frontale (ripresa 2 volte), quella dal profilo sinistro, quella dal profilo destro, quella ad una angolazione di 45° da sinistra e quella ad una angolazione di 45° da destra. Come per gli attori, l'ID assegnato ad ogni telecamera è fisso per tutto

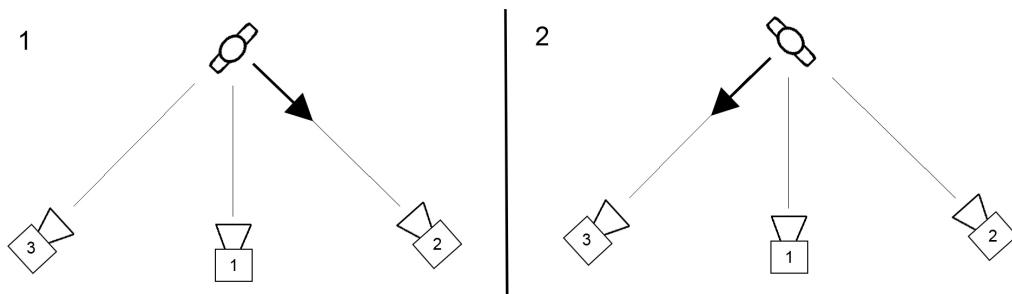


Figura 5.3: Disposizione delle telecamere per ogni setup. La telecamera 1 ha ripreso ogni azione da una posizione centrale mentre le telecamere 2 e 3 hanno ripreso le azioni con un’angolazione di 45° . Ogni attore ha ripetuto l’azione 2 volte: una rivolto verso la telecamera di destra e una verso quella di sinistra.

il dataset: la telecamera 1 è sempre quella che riprende da 0° , la telecamera 2 da 45° e la 3 da -45° . Uno schema della disposizione delle telecamere e della procedura di ripresa è raffigurato in figura 5.3

Per aumentare ulteriormente la variabilità delle riprese, ad ogni *setup* (ovvero ad ogni set di ripresa) le 3 telecamere sono state posizionate ad un’altezza e ad una distanza diversa dall’attore inquadrato, ovviamente mantenendo sempre la stessa altezza fra le 3 telecamere. In tabella 5.1 sono elencate le posizioni delle 3 telecamere per ogni setup.

5.2 Criteri di valutazione

Per avere una valutazione standard dei risultati ottenuti su questo dataset, gli autori definiscono nel loro lavoro [19] due particolari criteri di valutazione, di modo da poter allineare anche i risultati futuri e poterli comparare fra loro.

5.2.1 Valutazione Cross-subject

Per questa valutazione, i 40 attori sono stati divisi in due gruppi: *training* e *test* ed ogni gruppo è costituito da 20 attori. Così facendo è stato ottenuto un insieme di training di 40320 video e un insieme di test di 16560 video.

Setup	Altezza (m)	Distanza (m)	Setup	Altezza (m)	Distanza (m)
1	1.7	3.5	2	1.7	2.5
3	1.4	2.5	4	1.2	3.0
5	1.2	3.0	6	0.8	3.5
7	0.5	4.5	8	1.4	3.5
9	0.8	2.0	10	1.8	3.0
11	1.9	3.0	12	2.0	3.0
13	2.1	3.0	14	2.2	3.0
15	2.3	3.5	16	2.7	3.5
17	2.5	3.0			

Tabella 5.1: Altezza e distanza delle 3 telecamere per ogni setup di ripresa. Le altezze e le distanze sono espresse in metri.

Gli ID degli attori dell'insieme di training sono: 1, 2, 4, 5, 8, 9, 13, 14, 15, 16, 17, 18, 19, 25, 27, 28, 31, 34, 35, 38. I rimanenti fanno parte dell'insieme di test.

5.2.2 Valutazione Cross-view

Per quanto riguarda invece la valutazione *cross-view*, tutti i video ripresi dalle telecamere 2 e 3 costituiscono l'insieme di training, mentre quelli ripresi dalla telecamera 1 l'insieme di test. In altre parole l'insieme di training include tutte le prospettive frontali del soggetto inquadrato e tutte quelle di profilo mentre l'insieme di test include tutte le prospettive a 45°. Dividendo il dataset con questo criterio si ottiene un insieme di training composto da 37920 video e quello di test da 18960.

Capitolo 6

Esperimenti e risultati

Capitolo 7

Conclusioni e sviluppi futuri

Bibliografia

- [1] PersonLab: Person Pose Estimation and Instance Segmentation with a Bottom-Up, Part-Based, Geometric Embedding Model - *George Papandreou, Tyler Zhu, Liang-Chieh Chen, Spyros Gidaris, Jonathan Tompson, Kevin Murphy* - 2018
- [2] Coco 2016 keypoint challenge - Lin, T.Y., Cui, Y., Patterson, G., Ronchi, M.R., Bourdev, L., Girshick, R., Dollr,P. - 2016
- [3] PoseNet with TensorFlow.js - <https://medium.com/tensorflow/real-time-human-pose-estimation-in-the-browser-with-tensorflow-js-7dd0bc881cd5>
- [4] Detectron2 - *Yuxin Wu, Alexander Kirillov, Francisco Massa, Wan-Yen Lo, Ross Girshick* - <https://github.com/facebookresearch/detectron2>, 2019
- [5] Detectron - *Ross Girshick, Ilija Radosavovic, Georgia Gkioxari, Piotr Dollàr, Kaiming He* - <https://github.com/facebookresearch/detectron>, 2018
- [6] Caffe2 - <https://caffe2.ai/docs>
- [7] Mask R-CNN - *Kaiming He, Georgia Gkioxari, Piotr Dollár, Ross Girshick*, 2017
- [8] Focal Loss for Dense Object Detection - *Tsung-Yi Lin, Priya Goyal, Ross Girshick, Kaiming He, Piotr Dollár*, 2017
- [9] Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks - *Shaoqing Ren, Kaiming He, Ross Girshick, Jian Sun*, 2015
- [10] Fast R-CNN - *Ross Girshick*, 2015

- [11] R-FCN: Object Detection via Region-based Fully Convolutional Networks - *Jifeng Dai, Yi Li, Kaiming He, Jian Sun*, 2016
- [12] Aggregated Residual Transformations for Deep Neural Networks - *Saining Xie, Ross Girshick, Piotr Dollár, Zhuowen Tu, Kaiming He*, 2016
- [13] Deep Residual Learning for Image Recognition - *Kaiming He, Xiangyu Zhang, Shaoqing Ren, Jian Sun*, 2015
- [14] Feature Pyramid Networks for Object Detection - *Tsung-Yi Lin, Piotr Dollár, Ross Girshick, Kaiming He, Bharath Hariharan, Serge Belongie*, 2016
- [15] Very Deep Convolutional Networks for Large-Scale Image Recognition - *Karen Simonyan, Andrew Zisserman*, 2014
- [16] Cascade R-CNN: High Quality Object Detection and Instance Segmentation - *Zhaowei Cai, Nuno Vasconcelos* - 2019
- [17] Panoptic Feature Pyramid Networks - *Alexander Kirillov, Ross Girshick, Kaiming He, Piotr Dollár* - 2019
- [18] TensorMask: A Foundation for Dense Object Segmentation - *Xinlei Chen, Ross Girshick, Kaiming He, Piotr Dollár* - 2019
- [19] NTU RGB+D: A Large Scale Dataset for 3D Human Activity Analysis - *Amir Shahroudy, Jun Liu, Tian-Tsong Ng, Gang Wang* - 2016
- [20] Long short-term memory - *Sepp Hochreiter; Jürgen Schmidhuber* - 1997
- [21] Recurrent Nets that Time and Count - *Felix A. Gers; Jürgen Schmidhuber* - 2000
- [22] Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation - *Kyunghyun Cho; Bart van Merriënboer; Caglar Gulcehre; Dzmitry Bahdanau; Fethi Bougares; Holger Schwenk; Yoshua Bengio* - 2014

- [23] Recognizing Human Actions as the Evolution of Pose Estimation Maps - *Mengyuan Liu, Junsong Yuan* - 2018
- [24] Actional-Structural Graph Convolutional Networks for Skeleton-based Action Recognition - *Maosen Li, Siheng Chen, Xu Chen, Ya Zhang, Yanfeng Wang, and Qi Tian* - 2019
- [25] Vertex feature encoding and hierarchical temporal modeling in a spatial-temporal graph convolutional network for action recognition - *Konstantinos Papadopoulos, Enjie Ghorbel, Djamila Aouada, Bj?orn Ottersten* - 2019
- [26] Interpretable 3D human action analysis with temporal convolutional networks - *T. S. Kim, A. Reiter* - 2017