

FACOLTA’ DI INGEGNERIA

DIPARTIMENTO DI INGEGNERIA INDUSTRIALE E DELL’INFORMAZIONE

APPRENDIMENTO AUTOMATICO

TESI DI LAUREA

**ANALISI DEL DATASET MAMMOGRAPHIC MASSES**

Candidato: Cosimo Strusi

A.A. 2022/2023

**Indice**

[1 Introduzione 3](#_Toc114651792)

[2 Descrizione dataset 5](#_Toc114651793)

# **1 Introduzione**

Gli esseri umani hanno la capacità intrinseca di riconoscere correttamente le attività umane tramite il proprio sistema di percezione. Tuttavia, le macchine non possiedono questa capacità: non può interpretare facilmente un video e riconoscere le attività che sta compiendo il protagonista. I movimenti da riconoscere sono spesso attività tipiche eseguite al chiuso, come camminare, parlare, stare in piedi e sedersi. Per riconoscerli è indispensabile raccogliere dati da sensori che possono:

* Essere registrati in remoto.
* Essere registrati direttamente tramite dispositivi in loco.

In questo elaborato implementeremo un algoritmo di Human Activity Recognition (HAR). Per HAR si intende un settore del machine learning che studia tramite le acquisizioni multiple di dati, tecniche che permettono a una macchina di apprendere e poi classificare delle attività umane. La raccolta dei dati differenzia due diversi approcci dell’activity recognition:

1. Attività di riconoscimento basata su sensori.
2. Attività di riconoscimento basata su sistemi di visione.

La prima fa uso di una rete di sensori per monitorare il comportamento dell’uomo, la maggior parte di questi sensori sono dispositivi indossabili come giroscopi o accelerometri che i soggetti indossano continuamente. Tramite canali wireless, questi dispositivi trasmettono i dati a un sistema centralizzato. Uno degli svantaggi di questo approccio è dovuta all’instabilità dei dispositivi usati, possono scaricarsi e possono essere dimenticati da chi deve indossarli. La seconda utilizza video come input per gli algoritmi. Queste specie di telecamere inquadrano tutto l’ambiente, quindi lo scopo è individuare il soggetto e tracciarlo all’interno dello spazio. Naturalmente sono procedimenti più complicati e che richiedono un pre-processing più importante, ma ha il vantaggio di non costringere il soggetto ad indossare sensori.

Nel passato la raccolta dei dati dei sensori per il riconoscimento delle attività era impegnativa e costosa, perché richiedeva hardware particolare. Ora gli smartphone che sono utilizzati per il monitoraggio della salute sono dispositivi economici e utili a questo scopo. Uno dei problemi più affrontati è il riconoscimento dell’attività dati dei dati dei sensori, per classificare delle serie temporali uni-variate o multivariate.

Ciò non è semplice perché i nostri movimenti per quanto possano essere individuati concettualmente come uno solo, hanno molte variazioni. Ogni essere umano svolge un’attività in modo diverso da un altro. Lo scopo è quello di registrare i dati dei sensori e le attività corrispondenti (da utilizzare come classe) per soggetti specifici, da quest’ultimi costruire un modello e adattarlo in modo che possa generalizzare al meglio il problema e comportarsi allo stesso modo con dati che non ha mai visto.

Il riconoscimento dell’attività è alla base per lo sviluppo di molte potenziali applicazioni, la maggior parte inerenti alla salute e al fitness. Ogni app oggi può trattare questi dati dopo un’approvazione, e sono anche facilitati da grandi aziende che hanno implementato metodi per elaborarli a basso costo economico e computazionale come Google. Infatti Google ha reso accessibili delle Api di Activity Recognition in modo da rendere le app più intelligenti e aumentare l’interazione con l’utente con nuove modalità. L’API Activity Reconognition si basa sui sensori disponibili in un dispositivo come lo smartphone o lo smart-watch, raccoglie una miriade di dati e li elabora attraverso algoritmi solidi. Un esempio di queste API è il riconoscimento del fatto che l’utente è entrato in auto e quindi l’app può interagire con lui in diversi modi: Spotify fornisce una visione più limitata e intuitiva per l’utente che è entrato in auto.

L’esecuzione del riconoscimento dell’attività umana ha ormai raggiunto ottimi risultati, molti metodi di deep-learning hanno ottenuto buone prestazioni in diversi ambiti. In questo elaborato affronteremo questo problema sfruttando dei dati disponibili e la rete LSTM Long short-term memory, un’architettura di rete neurale ricorrente artificiale per l’attività di riconoscimento delle azioni umane. La LSTM imparerà caratteristiche complesse del segnale grezzo dei sensori per essere in grado di distinguere tra comuni attività dell’essere umano.

# **1.1 Movimento umano**

Classificare il movimento significa determinare quale tipo di azione viene rappresentata da un dato movimento umano nello spazio. Essere in grado di comprendere il mobimento significa determinare le relazioni temporali tra i cambiamenti che si verificano nei dati raccolti.

# **2 Descrizione dataset**

Smartphone e smartwatch contengono accelerometri triassiali che misurano l'accelerazione in tutte e tre le dimensioni spaziali. Questi accelerometri sono in grado di rilevare l'orientamento del dispositivo, che può fornire informazioni utili per il riconoscimento dell'attività. Il dataset MotionSense raccoglie questi dati da 24 utenti diversi mentre eseguivano alcune attività umane comuni come: camminare, fare jogging, salire le scale, scendere le scale, sedersi e stare in piedi per periodi di tempo specifici. In tutti i casi, i dati vengono raccolti ogni 50 millisecondi, ovvero 20 campioni al secondo.

Questo set di dati include dati di serie temporali generati da sensori di accelerometro e giroscopio. Vengono raccolti grazie ad un Iphone 6s tenuto nella tasca del partecipante utilizzando SensingKit che raccoglie informazioni dal framework Core Motion sui dispositivi IOS. Tutti i dati sono stati raccolti con una frequenza di campionamento di 50 Hz. I soggetti che hanno partecipato alla raccolta dei dati sono stati 24 dell’età compresa fra i 18 e 46 anni, che eseguivano una delle sei attività standard in 15 prove nello stesso ambiente e condizioni per garantire una situazione omogenea fra i vari partecipanti. Le sei attività principali sono indicate di seguito con la loro abbreviazione usata nel dataset:

1. Passeggiare (wlk)
2. Camminare al piano di sopra (dws)
3. Camminare al piano di sotto (ups)
4. Stare seduto (sit)
5. Stare in piedi (std)
6. Fare jogging (jog)

La raccolta dei dati è avvenuta tramite il SensigKit che è un framework di rilevamento mobile multipiattaforma, quindi funziona sia su IOS che Android. E’ multisensoriale, supporta più sensori come movimento, orientamento, posizione e altri. Il Core Motion invece è l’oggetto che incapsula inizialmente le informazioni che ci servono ed è tipico del sistema IOS. Per ogni partecipante, lo studio è stato avviato raccogliendo le informazioni demografiche (età e sesso) e fisiche (peso e altezza). E’ stato fornito loro uno smartphone dedicato da lasciare nella tasca anteriore dei pantaloni durante l’esperimento. A tutti i partecipanti è stato chiesto di indossare scarpe basse. Le attività da svolgere sono state completate nel campus MIle End della Queen Mary University di Londra.



Figura 1 - Mappa delle attività

Una volta consegnato il cellulare è stata avviata l’app presente sull’App Store, CrowdSense che è un sistema di rilevamento continuo e gratuito, basato sulla libreria SensingKit di cui abbiamo parlato prima. E’ lo strumento più facile da utilizzare per i ricercatori che vogliono raccogliere dei dati utilizzando dei dispositivi IOS.

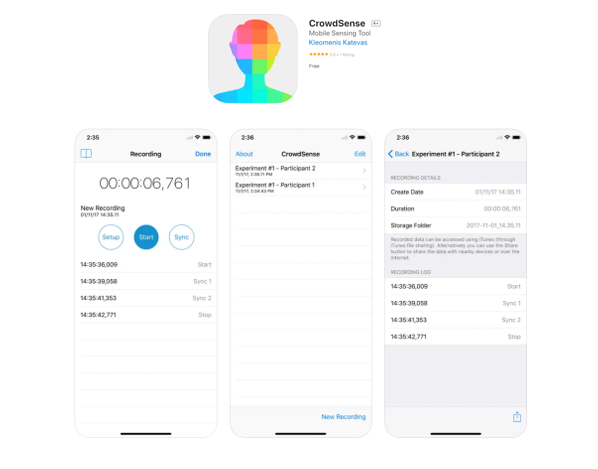


Figura 2 - App CrowdSense

Alla fine di ogni attività il partecipante preme sull’app il pulsante di arresto per definire con alta precisione l’inizio e la fine di ogni attività.

Come possiamo vedere dalla Figura n.1 ci sono 15 prove effettive:

* Prove lunghe: quelle con numero da 1 a 9 con durata da 2 a 3 minuti circa.
* Prove corte: quelle con i numeri da 11 a 16 che durano da 30 secondi a 1 minuto circa.

I 24 partecipanti sono riassunti con le loro informazioni nella seguente tabella.

Tabella 1 - Riassunto partecipanti

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Codice | Peso (kg) | Altezza (cm) | Età (anni) | Sesso (0:F , 1:M) |
| 1 | 102 | 188 | 46 | 1 |
| 2 | 72 | 180 | 28 | 1 |
| 3 | 48 | 161 | 28 | 0 |
| 4 | 90 | 176 | 31 | 1 |
| 5 | 48 | 164 | 23 | 0 |
| 6 | 76 | 180 | 28 | 1 |
| 7 | 62 | 175 | 30 | 0 |
| 8 | 52 | 161 | 24 | 0 |
| 9 | 93 | 190 | 32 | 1 |
| 10 | 72 | 164 | 31 | 0 |
| 11 | 70 | 178 | 24 | 1 |
| 12 | 60 | 167 | 33 | 1 |
| 13 | 60 | 178 | 33 | 1 |
| 14 | 70 | 180 | 35 | 1 |
| 15 | 70 | 185 | 33 | 1 |
| 16 | 96 | 172 | 29 | 0 |
| 17 | 76 | 180 | 26 | 1 |
| 18 | 54 | 164 | 26 | 0 |
| 19 | 78 | 164 | 28 | 0 |
| 20 | 88 | 180 | 25 | 1 |
| 21 | 52 | 165 | 24 | 1 |
| 22 | 100 | 186 | 31 | 1 |
| 23 | 68 | 170 | 25 | 0 |
| 24 | 74 | 173 | 18 | 0 |

All’interno della cartella fornita ci sono 3 file:

1. “DeviceMotion\_data”: contiene le serie temporali raccolte sia dall’accelerometro che dal giroscopio per tutte e 15 le prove e per ogni partecipante. Per ogni prova abbiamo una serie temporale multivariata come quella mostrata di seguito.

**ESEMPIO**

Le serie temporali sono formate da 12 features: Attitude.roll, attitude.pitch, attitude.yaw, gravity.x, gravity.y, gravity.z, rotationRate.x, rotationRate.y, rotationRate.x, userAcceleration.x, userAcceleration.y, userAcceleration.z.

L’accelerometro misura la somma di due vettori di accelerazione: gravità e accelerazione utente che è quella che l’utente impartisce al dispositivo. Core Motion è in grado di tracciare l’assetto di un dispositivo utilizzando sia il giroscopio che l’accelerometro, può distinguere tra accelerazione di gravità e accelerazione dell’utente.

1. “Accelerometer\_data”: contiene parte dei dati della cartella “DeviceMotion\_data”, in particolare i dati dell’accelerometro. Come features all’interno dei vari file troviamo 3 campi che descrivono le componenti dell’accelerazione sull’asse x, y e z.
2. “Gyroscope\_data”: contiene parte dei dati della cartella “DeviceMotion\_data”, in particolare i dati del giroscopio. Come features all’interno dei vari file troviamo 3 campi che descrivono le componenti sull’asse x, y e z.

Per quanto riguarda il problema di questo elaborato utilizzeremo solo il file DeviceMotion che contiene tutte le features di cui abbiamo bisogno, essendo gli altri file solo rielaborazioni di quest’ultimo.

# **3 Metodi**

Per affrontare questo problema si è deciso di utilizzare uno Python che è un linguaggio open source di alto livello, orientato ad oggetti perché dispone di molte librerie per il machine learning

## **3.1 Software**

### **3.1.2 Tensorflow**



E’ una libreria di software open source per l’apprendimento automatico che fornisce moduli sperimentati e ottimizzati, utili nella realizzazione di algoritmi per diversi tipi di compiti.

### **3.1.2 Keras**

E’ una libreria open source per l’apprendimento automatico e le reti neurali, scritta in Python. E’ progettata come un’interfaccia a un livello di astrazione superiore rispetto alla maggior parte delle librerie simili, e supporta come back-end le librerie come quella di TensorFlow.

## **3.2 LSTM**

La rete di memoria a lungo termine (LSTM) è un tipo di rete neurale ricorrente (RNN), che è un tipo speciale di rete neurale progettata per problemi di sequenza come, ad esempio, testi, voce e animazioni. Gli RNN tradizionali contengono cicli che alimentano l’attivazione della rete da un passaggio temporale precedente come input per influenzare le previsioni in quello attuale. Nonostante il fatto che le RNN possano apprendere le relazioni temporali, il loro principale limite riguarda un problema ben noto come “gradiente evanescente”. Questo problema si verifica quando, durante l’allenamento di un processo ricorrente, i pesi cambiano diventando così piccoli da non avere alcun effetto nell’apprendimento dei dati. Gli LSTM risolvono questi problemi grazie alla sua progettazione, tutte le informazioni propagate attraverso la rete dovrebbero passare attraverso 3 gates, in modo da consentire solo alle informazioni rilevanti di continuare a essere propagate durante l’addestramento:

1. Forget Gate: decide quali informazioni scartare da ogni strato.
2. Input Gate: decide a quali valori dell’input aggiornare la memoria.
3. Output Gate: decide cosa produrre in output sulla base dell’input e della memoria.

In letteratura si possono trovare molte variazioni di questa rete, che prevedono un numero diverso di gate per adattarsi a contesti ed esigenze specifiche.

# **4 Approccio**

Si è deciso di approcciare a questo problema con una LSTM.

## **4 Pulizia dataset e preelaborazione**

In questo paragrafo dopo una lettura iniziale dei vari csv forniti era necessario accodarli mantenendo l’ordine perché in questo caso trattandosi di serie temporali l’ordine è molto importante. Dopo aver fatto un’analisi dei dati e controllato eventuali dati mancanti si è deciso di costruire l’effettiva matrice da cui partiamo per implementare tutta la pipeline. Costruendola si è deciso di aggiungere una colonna “time” calcolata in modo relativo basandoci sull’informazione della frequenza della raccolta dei dati.

L’accelerometro misura la somam di due vettori di accelerazione: la gravità e l’accelerazione dell’utente (l’accelerazione dell’utente è l’accelerazione che l’utente impartisce al dispositivo). Poiché CoreMotion è in grado di tracciare l’asseto di un dispositivo utilizzando sia il giroscopio che l’accelerometro, può distinguere tra gravità e accelerazione dell’utente. Nel nostro caso specifico può essere interessante invece sommare le varie componenti per ottenere la componente finale dell’accelerazione. Dunque sono stati trasformati i dati, calcolando la somma di “userAcceleration” e “gravity” otteniamo il nuovo valore “accel”:

E inoltre è stata calcolata la norma dell’accelerazione:

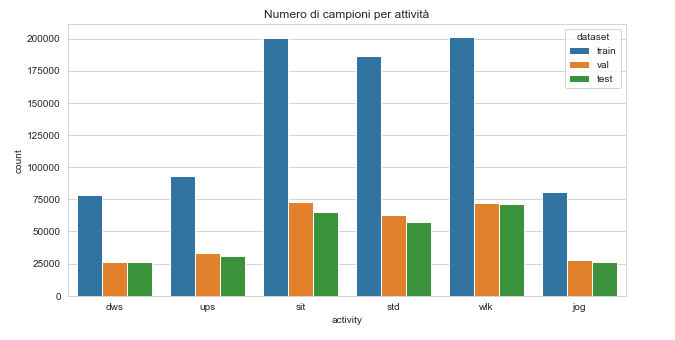
Sono stati scartati anche dal dataset iniziale gli attributi sull’attitude che non sono rappresentativi del nostro problema indicando soltanto orientamento del dispositivo nello spazio.

Le features finali diventano, scartando gli “userAcceleration” (che sono stati trasformati nella nuova variabile “accel”):

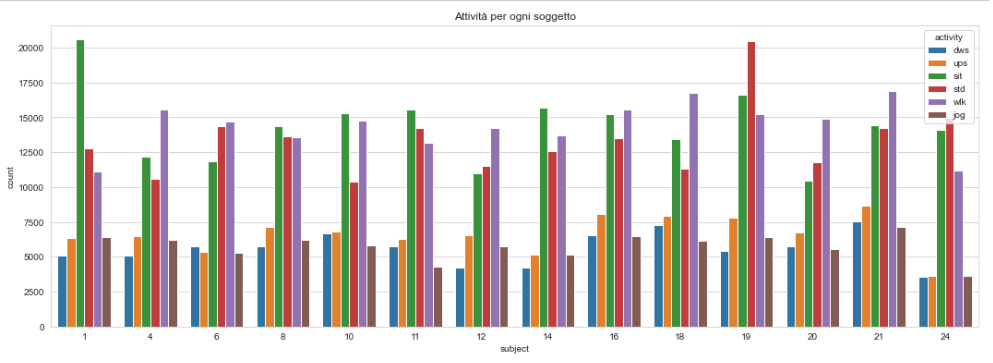
* accel\_x
* accel\_y
* accel\_z
* accel\_norm
* gravity\_x
* gravity\_y
* gravity\_z

## **4 Split dataset**

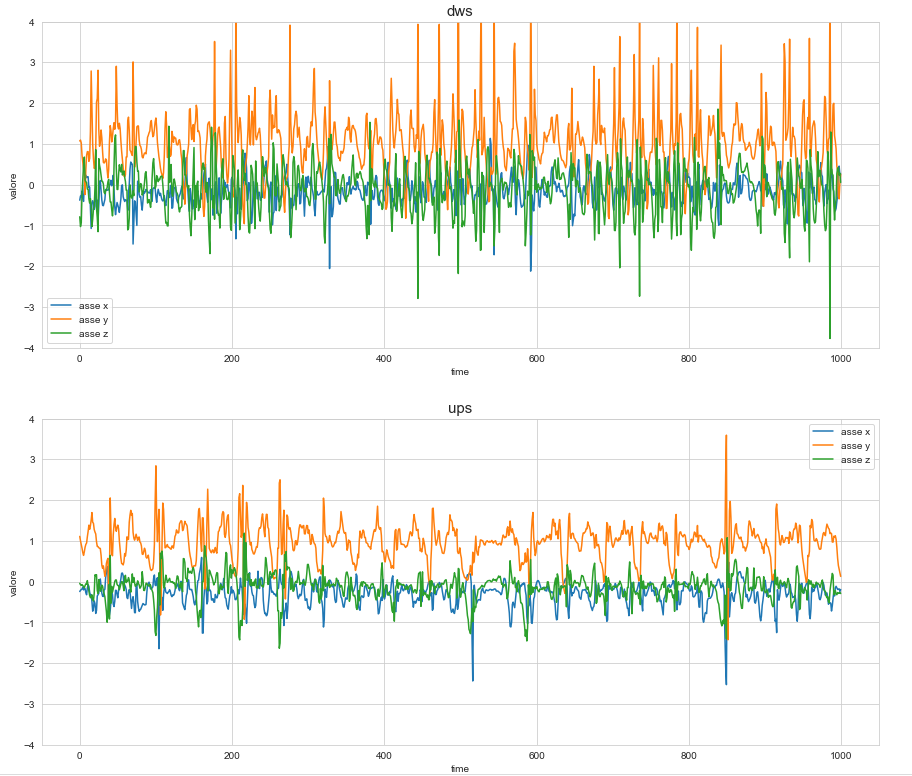
Il set di dati è stato diviso con l’approccio hold-out in un set di tran, validation e test, utilizzando come parametro di discriminazione l’id del soggetto. Sono stati assegnati quindi tutte le series di 14 soggetti al train, 5 soggetti al validation e 5 al test. Questa scelta di suddivisione ha permesso di garantire la proporzione delle classi nei differenti dataset come possiamo vedere dal seguente grafico:



Come si può vedere il numero di attività per le varie classi non è lo stesso, ciò potrebbe influenzare il lavoro di training sul modello avendo classi che sono in minoranza.



Per dare un’idea della natura dei dati che abbiamo a disposizione possiamo graficare i dati sull’accelerazione dei diversi assi per ogni attività.



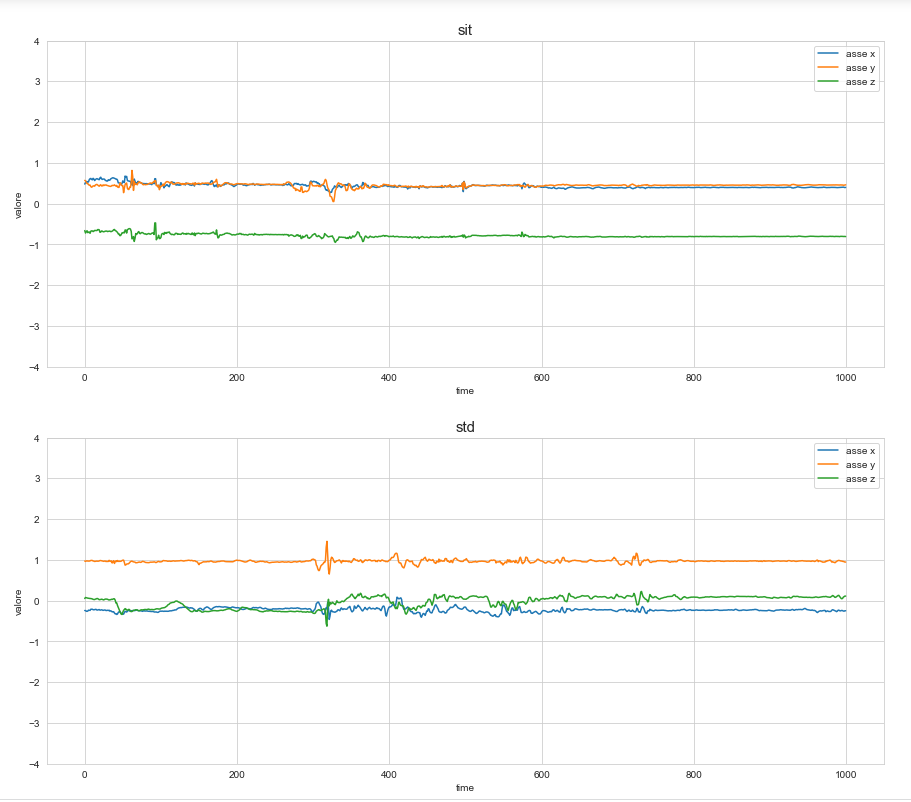


Immagine che contiene strumento scrittorio, stazionario, matita

Descrizione generata automaticamente

Come possiamo notare, solo utilizzando come metro di paragone la feature “accel” possiamo notare delle differenze a colpo d’occhio che possono differenziare alcune attività. Per questo siamo molto fiduciosi sul risultato di un modello.

## **4 Normalizzare i dati**

Per ogni algoritmo è quasi sempre indispensabile scalare le features in modo che eventuali misure su scale diverse possano influenzare il modello allo stesso modo. Per evitare eventuali problemi si è deciso di standardizzare le varie colonne del dataset di training utilizzando una libreria di python chiamata sklearn come possiamo vedere di seguito:



Naturalmente tutti i calcoli di preprocessing devono essere fatti sul training e poi riutilizzati sul test e validation. In questo caso infatti si è standardizzato il training e sono stati utilizzati gli stessi parametri del training (media e deviazione standard) per “standardizzare” i dati di test e validation.

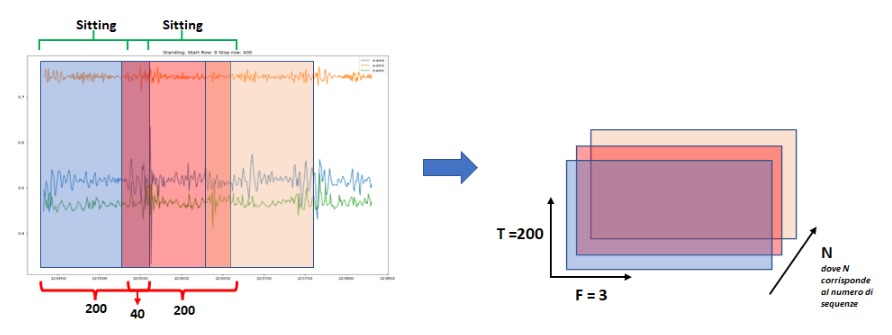
## **4 Reshape e One-Hot-Encode**

Per trattare le serie temporali con dei modelli non possiamo utilizzare i dati grezzi direttamente. Invece, dobbiamo prima trasformare i dati grezzi delle serie temporali usando la tecnica del “windowing”. In questa tecnica dividiamo i dati in finestre di un certo tempo, generando nuovi set di dati a cui viene assegnata la classe corrispondente.

Inoltre è possibile finestre sovrapposte, ciò implica che ogni set di dati successivo contenga anche alcune informazioni della finestra precedente.

Tutto ciò è stato fatto perché la rete LSTM deve avere come input un tensore di dimensioni (N, T, F):

* N è il numero di sequenze che compongono il dataset
* T è il numero di time\_steps che compongono la sequenza
* F è il numero di feature che descrive il dataset



Nel nostro caso specifico per il training è stato scelto un TIME\_STEPS di 200 con uno STEP di 40 garantendo quindi una leggera sovrapposizione delle varie finestre. Per il test e validation lo STEP è uguale a 0 perché non vogliamo che ci sia sovrapposizione in quanto dobbiamo verificare solo ciò che ha imparato il modello dal set di training. A causa di questa modifica in input, l’uscita sarà un vettore di 6 elementi e quindi è necessaria una rielaborazione del vettore della classe per trasformarlo in una matrice di 0 e 1 dove sulle righe abbiamo i diversi casi e sulle colonne abbiamo le varie classi che possono rappresentare un esempio. In ogni riga quindi troveremo un solo 1 e cinque volte lo 0, la posizione dell’1 indica la classe che identifica quel caso. Questa trasformazione è stata effettuata per train, test e validation.

## **4 Modello costruito in Keras**

Sono stati costruiti due reti con complessità diversa attraverso la libreria Keras, utilizzando una base di un modello sequenziale sulla quale sono stati aggiunti i vari strati che ci interessavano. Tutti i modelli sono stai compilati in modo che per la funziona di perdita si cerchi di ridurre al minimo la “categorical crossentropy” cioè l’entropia delle classi, è stato utilizzato come ottimizzatore Adam e come metrica di stampa l’accuratezza.

## **4.1 Primo modello**

Il primo modello è stato costruito con uno strato LSTM di 100 unità seguito da uno strato di dropout con parametro di 0.5 e da due strati densi. Il penultimo strato denso ha come funzione di attivazione “relu” mentre l’ultimo strato denso necessario per la classificazione, ha una funziona di attivazione “softmax”.

Immagine che contiene tavolo

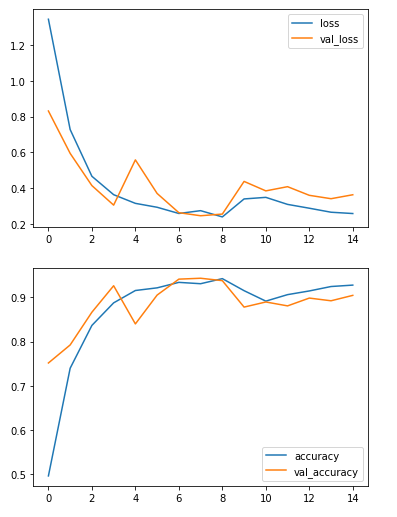
Descrizione generata automaticamente

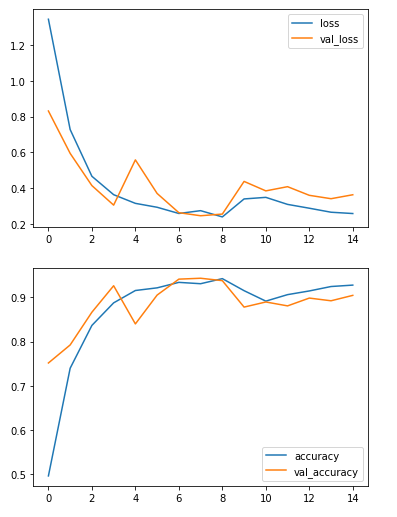
La struttura del modello determina un totale di 53,906 parametri da determinare. L’allenamento è stato lanciato per 15 epoche senza alcun vincolo di stop della rete, di seguito una parte dell’output dell’esecuzione dell’allenamento:

Immagine che contiene testo

Descrizione generata automaticamente

Dopo il primo allenamento è possibile graficare la funzione di perdita e la funzione di accuratezza per mostrare l’andamento sulle epoche.





I grafici mostrano che il validation e il train seguono lo stesso andamento, ciò vuol dire che i parametri che sono stati ritoccati sulla base dei risultati sul validation hanno permesso al modello di generalizzare il problema e di definire una struttura del modello adeguata per questo problema. Per valutare il modello sono state prese in considerazione come misure accuratezza, precisione e recall per le varie classi.

A questo punto validato il modello si è riallenato mettendo insieme sia i dati train che quelli di validation per poi testarli per la prima volta su dati mai visti prima, cioè quelli di test.

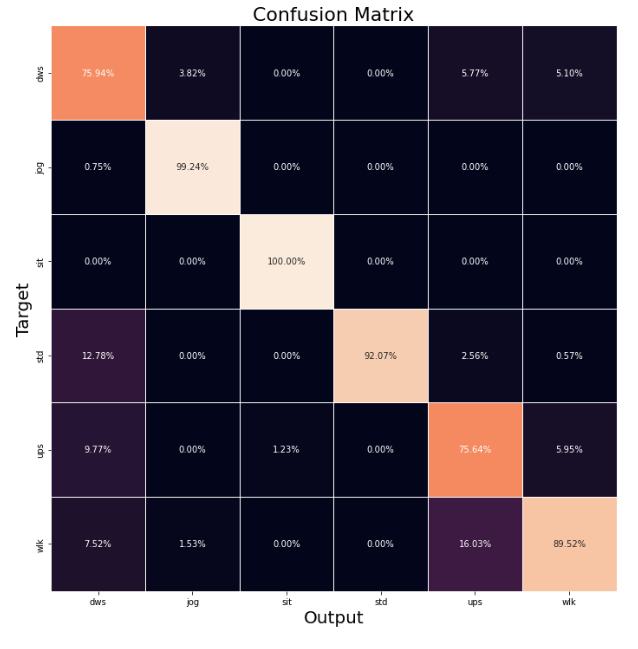
Complessivamente il modello sul set di train ha raggiunto un’accuratezza totale di 0.95 e per ogni classe sono state calcolate le seguenti statistiche:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | dws | jog | sit | std | ups | wlk |
| Accuratezza | 0.97 | 0.99 | 0.99 | 0.99 | 0.97 | 0.96 |
| Precisione | 0.86 | 0.94 | 0.98 | 0.99 | 0.95 | 0.91 |
| Recall | 0.85 | 0.96 | 0.99 | 0.98 | 0.83 | 0.96 |

Sul set di test è stata ottenuta un’accuratezza uguale a 0.9055 e le seguenti statistiche su ogni classe:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | dws | jog | sit | std | ups | wlk |
| Accuratezza | 0.94 | 0.99 | 0.99 | 0.98 | 0.94 | 0.94 |
| Precisione | 0.71 | 0.94 | 0.98 | 1 | 0.75 | 0.88 |
| Recall | 0.75 | 0.99 | 1 | 0.92 | 0.75 | 0.89 |

Inoltre per il test è stata calcolata la matrice di confusione.



# **5 Limiti**

Una delle prime limitazioni è il dataset che abbiamo a disposizione. Un numero più elevato di dati avrebbe permesso di generalizzare ancora meglio il problema. Si potrebbe pensare di ripetere la stessa campagna di raccolta dati cercando di garantire che ogni soggetto faccia lo stesso numero di prove in modo da avere per ogni attività lo stesso numero, e quindi classi completamente bilanciate. Un’altra limitazione è il mancato utilizzo della GPU o di un computer con elevate prestazioni computazionali.

# **6 Sviluppi futuri**

Uno dei possibili sviluppi futuri è quello di costruire modelli capaci di riconoscere altre caratteristiche dei dati catturati dal movimento come le posizione effettive del corpo.