Applicazione di Reti Neurali compatte su dati reali in ambito HAR

Marzio Della Bosca[1]*, Sara Montagna[2], Stefano Ferretti[3]

Sommario
Questo lavoro si inserisce nell'ambito del riconoscimento di attività umane (Human Activity Recognition - HAR) tramite dati provenienti da sensori inerziali (Inertial Measurement Unit - IMU) montati su dispositivi mobili, come in questo caso i telefoni cellulari. L'obiettivo principale è sviluppare modelli di classificazione leggeri ma robusti, capaci di generalizzare su più dataset. Sono stati aggregati e analizzati dati provenienti da quattro dataset pubblici (HHAR, MotionSense, WISDM e MobiAct), accomunati dalla disponibilità di segnali da accelerometro e giroscopio (unici sensori utilizzati per i dati, direzioni: x, y e z). Per affrontare le differenze tra dataset, in particolare le variazioni nelle frequenze di campionamento (come nel caso di HHAR e WISDM), è stato adottato un approccio basato su tecniche di estrazione di feature avanzate, come catch22 e TSFEL, permettendo di ricavare rappresentazioni descrittive da finestre temporali uniformi di 2 secondi, standardizzando così la forma dei dati in ingresso indipendentemente dal numero di campioni effettivi contenuti in ciascuna finestra. L'utilizzo di questi estrattori, anche se il loro utilizzo non rientra nelle pratiche tipiche del deep learning, si rivela strategico in questo contesto perchè consente di anticipare parte del carico computazionale alla fase di preprocessing, alleggerendo il modello e rendendolo più adatto all'esecuzione in tempo reale (delle inferenze) su dispositivi aventi risorse computazionali limitate.

Keywords
TSFEL — Catch22 — MLP — CNN — HAR — MobiAct — HHAR — MotionSense — WISDM

[1] *Laurea Magistrale in Informatica e Innovazione Digitale, Università degli Studi di Urbino Carlo Bo, Urbino, Italia*
[2] *Docente di Applicazioni dell'Intelligenza Artificiale, Università degli Studi di Urbino Carlo Bo, Urbino, Italia*
[3] *Docente di Applicazioni dell'Intelligenza Artificiale, Università degli Studi di Urbino Carlo Bo, Urbino, Italia*
***Author**: m.dellabosca@campus.uniurb.it

Introduzione

Negli ultimi anni, il riconoscimento automatico delle attività umane (Human Activity Recognition - HAR) ha suscitato sempre più interesse in diversi ambiti. Per diversi motivi, come ad esempio grazie alla diffusione capillare di dispositivi mobili e indossabili dotati di sensori inerziali (IMU). Questi dispositivi che in genere includono accelerometri, giroscopi e magnetometri permettono la raccolta continua di dati relativi al movimento. Nonostante i progressi ottenuti negli utlimi anni permangono ancora delle criticità in ambito HAR: molti modelli sono addestrati e valutati sui singoli dataset, spesso acquisiti in condizioni controllate, e mostrano prestazioni limitate in scenari reali più variabili (mancanza di generalizzazione) inoltre i modelli più accurati tendono ad essere computazionalmente pesanti rendendone difficile l'esecuzione in tempo reale su dispositivi con risorse limitate, come telefoni o oggetti ancora più piccoli come ad esempio le ESP32.

Questo lavoro prova ad esplorare queste problematiche attraverso l'aggregazione di diversi dataset pubblici raccolti tramite sensori IMU su smartphone (HHAR, WISDM, MobiAct, MotionSense) con l'obiettivo di sviluppare modelli capaci di generalizzare su dati eterogenei ma mantenendoli con un numero di parametri e strati limitato. Per superare le disomogeneità tra i dataset, in termini di frequenza di campionamento e formato, viene adottato un approccio di estrazione di feature da finestre temporali fisse tramite strumenti come catch22 e TSFEL. Questa scelta, anche se non convenzionale rispetto alle pratiche del deep learning, consente di anticipare parte della complessità computazionale e standardizzare la rappresentazione dei dati.

In questo modo si cerca di ottenere modelli eseguibili su dispositivi mobili, cercando un compromesso tra accuratezza ed efficienza. I risultati mostrano la validità dell'approccio e suggeriscono la possibilità di un HAR robusto e real-time anche in ambienti non controllati con risorse computazionali limitate. Tutti gli esperimenti condotti sono stati eseguiti su una macchina personale avente 32 GByte di memoria RAM, CPU AMD da 12 core ($\tilde{4}$.2 GHz cad.) e scheda GPU NVIDIA da 8 GByte (architettura cuda fruttata attraverso librerie di PyTorch).

1. Metodi

Alcuni dataset presentano frequenze di campionamento diverse e formati diversi, dunque si sono rese necessarie diverse strategie implementative per ognuno dei dataset. E' stato scelto un finestramento con lo scopo di ottenere un aggregato di caratteristiche ottenute mediante estrazione Catch22 e TSFEL su finestre di 2 secondi (finestra molto comune in ambito HAR). Infine si sono ottenuti 2 dataset, uno composto dalle feature estratte mediante Catch22 e un altro dalle feature estratte mediante TSFEL da ognuno dei 4 dataset. Al termine delle estrazioni, operate in maniera separata rispetto alle altre, è stata fatta l'aggregazione dei dataset e la normalizzazione. Successivamente è stato condotto il bilanciamento e un ulteriore controllo di eliminazione di valori non conformi (nan, null) dai dati. La differenza principale tra questi due estrattori è il numero delle feature estratte dai canali. I canali di dati scelti da cui estrarre le feature per allenare i modelli sono stati identificati nelle direzioni x, y e z di accellerometro e giroscopio. La normalizzazione dei dati è stata eseguita con la funzione StandardScaler di skalearn che trasforma ogni elemento x_i secondo la seguente equazione (il che motiva la scelta della normalizzazione post-aggregazione):

$$z_i = \frac{x_i - \mu}{\sigma} \tag{1}$$

dove:

- x_i è il valore originale della variabile;
- μ è la media dei valori della variabile;
- σ è la deviazione standard dei valori della variabile;
- z_i è il valore standardizzato.

Successivamente sono stati sperimentati dei Multilayer Perceptron (MLP) e delle Convolutional Neural Network (CNN) sui dati aggregati con l'idea di trovare delle architetture leggere in grado di eseguire inferenze su dispositivi con risorse computazionali limitate mantenendo delle buone performance in fase di valutazione. Il procedimento utilizzato nel corso dell'esplorazione è stato quello di identificare i modelli e il dataset (catch22 o tsfel) che dava le migliori performance e poi ne è stata fatta una grid search in profondità.

1.1 Dataset e preprocessing

In questa sezione si entra nello specifico in tutte le fasi intercorse nel preprocessing sui diversi dataset, in cui sono state fatte le seguenti operazioni:

1. Acquisizione dati "grezzi";
2. Finestramento;
3. Estrazione;
4. Aggregazione;
5. Normalizzazione dati;
6. Bilanciamento e pulizia dati.

Ogni sottosezione, per comodità e chiarezza, farà riferimento ad uno specifico dataset e al lavoro svolto su di esso.

1.1.1 Motion Sense

MotionSense[1] include dati provenienti da sensori IMU con una frequenza di campionamento di 20 Hz. I test sono stati condotti su 24 partecipanti che hanno eseguito una serie di attività come camminare, correre, salire e scendere le scale, stare seduti e in piedi.

Acquisiti i dati sono state tolte tutte le colonne tranne quelle delle accellerazioni, rotazioni da giroscopio ed etichette. Successivamente, dato che il dataset è stato acquisito campionando uniformemente a 20 Hz, si è eseguito un finestramento a 40 campioni in modo da ottenere finestre rappresentative di 2 secondi di attività. Infine dal dataset delle finestre sono stati ottenuti il dataset delle feature estratte con TSFEL e il dataset delle feature estratte con Catch22.

Data la grandezza di Motion sarebbe stato possibile eseguire un estrazione TSFEL completa, in quanto TSFEL tramite parametrizzazione permette di estrarre feature da diversi domini, ad esempio: statistico, temporale e spettrale, però per altri dataset non è stato possibile per via di problemi di campionamento e in alcuni casi anche per la grandezza del dataset a cui applicare l'estrazione. Non era possibile in quanto l'operazione saturava la ram a disposizione quindi si è optato per utilizzare TSFEL mediante estrazione dei domini statistici e temporali delle finestre.

1.1.2 MobiAct

MobiAct[2], acquisito tramite smartphone, è un dataset in cui i dati dei sensori sono raccolti in file separati (accelerometro, giroscopio), la cui frequenza di campionamento è 20 Hz, anche se sono state usate tecniche di interpolazione e downsampling per uniformare i dati in quella forma. I test sono stati condotti con partecipanti che svolgevano attività come camminare, correre, salire/scendere scale, stare in piedi/seduti e diverse cadute, seguendo protocolli controllati. MobiAct è utilizzato in HAR e in particolare come benchmark nel rilevamento delle cadute.

Il procedimento di finestramento ed estrazione delle feature è stato simile a quello utilizzato con Motion in quanto anche i dati forniti da MobiAct sono dati con campionamento omogeneo a 20 Hz.

1.1.3 Heterogeneity Activity Recognition

Heterogeneity Activity Recognition (HHAR)[3] è un dataset progettato per valutare algoritmi di riconoscimento delle attività in scenari del mondo reale con dispositivi quali smartphone e smartwatch di diversi modelli e sistemi operativi. È costituito da oltre 40 milioni di campioni raccolti da accelerometro e giroscopio registrati alla massima frequenza consentita da ogni dispositivo. Le attività incluse sono ciclismo, seduto, in piedi, camminata, salire e scendere le scale, i dati erano suddivisi in file csv:

- Dati accellerometro da smartphone;

- Dati giroscopio da smartphone;

Una problematica a cui si è andato incontro nella fase di preprocessing, oltre alle diverse frequenze di campionamento, sono stati i dati mancanti da parte del giroscopio. Dalle esplorazioni fatte è emerso che si sono persi dati da parte del giroscopio in maniera più o meno uniforme durante tutte le registrazioni ma per fortuna non in maniera importante. Mediante un operazione di raggruppamento sui dati in base alle caratteristiche: "utente, dispositivo e attività" dei campioni, è stato possibile suddividere i dati per registrazioni in quanto gli unici campioni (in ordine secondo time stamp) con stesso utente, dispositivo e attività possono essere riconducibili alla stessa registrazione. E' stato comunque stato fatto un controllo sui time stamp minimo e massimo di ogni registrazione ottenuta per verificare che non fossero due registrazioni fatte in giorni o ore diverse dallo stesso user con lo stesso dispositivo che svolgeva la stessa attività. Infine per ciascun gruppo definito dalla tripla (u,d,a), con u utente, d dispositivo e a attività, sono stati eseguiti i seguenti passaggi per il calcolo delle finestre:

- Calcolo del numero di campioni: $N_{u,d,a}$
- Calcolo della durata della sessione:

  $$T_{u,d,a} = t_{\max} - t_{\min}$$

  dove $t_{\max}$ e $t_{\min}$ sono rispettivamente il massimo e il minimo timestamp registrato per quel gruppo, in secondi.
- Calcolo della frequenza di campionamento (in Hz):

  $$f_{u,d,a} = \frac{N_{u,d,a}}{T_{u,d,a}}$$

- Calcolo della dimensione della finestra in numero di campioni per una finestra di 2 secondi:

  $$w_{u,d,a} = f_{u,d,a} \cdot 2$$

Questa procedura consente di adattare dinamicamente il finestramento alla frequenza effettiva di ciascun dispositivo. Dopo il finestramento è stata applicata l'estrazione TSFEL e Catch22 come negli altri casi.

1.1.4 WISDM

WISDM[4] è un dataset che raccoglie dati da accelerometro e giroscopio sia da diversi smartphone che da smartwatch, registrati a 20Hz. Vengono eseguite 18 attività quotidiane della durata di tre minuti ciascuna, da 51 soggetti diversi. Il dataset viene principalmente impiegato per il riconoscimento delle attività in HAR ed è utilizzato in studi con reti neurali come dataset di benchmark.

I dati sono stati presentati in circa 3200 file in formato txt, per i sensori di accellerometro e giroscopio degli smartphone. Ogni file conteneva i campioni di una registrazione di 3 minuti di una certa attività per uno dei due sensori imu, quindi circa 1600 file per accellerometro da smartphone e 1600 file per giroscopio da smartphone. Anche se la frequenza di campionamento teorica era uniforme a 20 Hz la grande quantità di mancanza di campioni dei giroscopi ha reso necessario un approccio di finestramento dinamico come quello usato per HHar, anche se non si è dovuto raggruppare in quanto le registrazioni erano già divise in file, inoltre è specificato che ogni file si riferisce a registrazioni di 3 minuti. Dopo il finestramento sono stati ottenuti i dataset Catch22 e TSFEL con un metodo analogo a quelli utilizzati con gli altri.

Dopo l'estrazione delle feature, e aggregazione, dei dataset ottenendo 2 dataset finali, uno con le estrazioni ottenute da Catch22 e l'altro da TSFEL, è stata fatta la normalizzazione. La normalizzazione è stata fatta sui dataset totali e non prima delle estrazioni per non far estrarre informazioni da serie temporali normalizzate. I dataset di "merge" su cui vengono fatte le estrazioni sono derivanti da un filtraggio sulla base delle classi a cui le finestre campione sono associate. Questi dataset sono stati scelti non solo in base ai contesti simili di acquisizione dei dati (smartphone, accellerometro, giroscopio, frequenz. di campionamento simili) ma anche sulla base del loro set di attività. Molte attività erano comuni tra i dataset ed altre simili (jogging - running) ma altre no e sono state scartate. Il set finale delle attività di inferenza è:

Tabella 1. Etichette delle attività

Attività	Etichetta
Walking	0
Jogging	1
Downstairs	2
Upstairs	3
Sitting	4
Standing	5

Le dimensioni finali dei dataset sono:

- TSFEL e Catch22 non bilanciati - versione 2d: (375358, 270) - versione 3d: (375358, 6, 45)
- TSFEL e Catch22 bilanciati - versione 2d: (218802, 270) - versione 3d: (218802, 6, 45)

[1]

1.2 Catch22 e TSFEL

Catch22 (Canonical Time-series Characteristics)[5] è un insieme di funzioni che estrae 22 feature a partire da serie temporali, selezionate da una libreria iniziale di 4791 feature per eliminare ridondanze e mantenere interpretabilità ed efficienza. L'obiettivo è offrire rappresentazioni compatte e interpretabili per clustering, classificazione e analisi di serie temporali. Le feature finali estratte da Catch22 sono:

[1] Walking 0 - Jogging 1- Downstairs 2 - Upstairs 3 - Sitting 4 - Standing 5

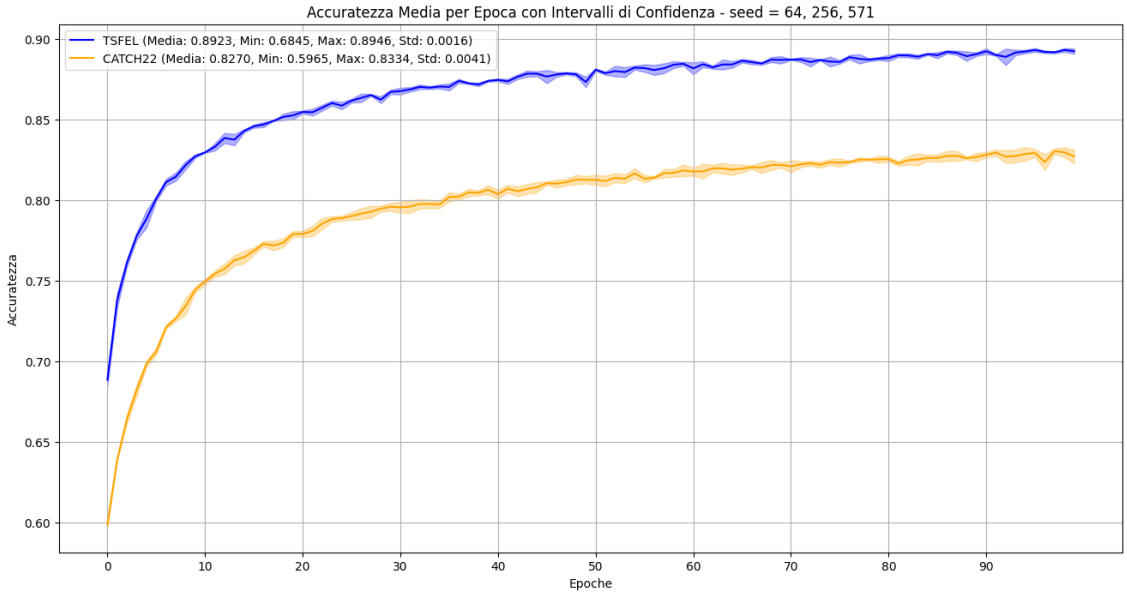

Figura 1. Confronto TSFEL - CATCH22 (n_epoche=100, lr=1e-4, batch=1024)

1. **DN_HistogramMode_5** — Moda della distribuzione normalizzata (z-score) su un istogramma a 5 bin.
2. **DN_HistogramMode_10** — Moda della distribuzione normalizzata (z-score) su un istogramma a 10 bin.
3. **SB_BinaryStats_mean_longstretch1** — Periodo più lungo di valori consecutivi sopra la media.
4. **DN_OutlierInclude_p_001_mdrmd** — Intervallo medio tra eventi estremi successivi sopra la media.
5. **DN_OutlierInclude_n_001_mdrmd** — Intervallo medio tra eventi estremi successivi sotto la media.
6. **CO_f1ecac** — Primo attraversamento della funzione di autocorrelazione a $1/e$.
7. **CO_FirstMin_ac** — Primo minimo della funzione di autocorrelazione.
8. **SP_Summaries_welch_rect_area_5_1** — Potenza totale contenuta nel primo quinto dello spettro di potenza di Fourier.
9. **SP_Summaries_welch_rect_centroid** — Centroide dello spettro di potenza di Fourier.
10. **FC_LocalSimple_mean3_stderr** — Errore medio nella previsione basata su una media mobile di 3 campioni.
11. **CO_trev_1_num** — Statistica di non reversibilità temporale basata su $(x_{t+1} - x_t)^3$.
12. **CO_HistogramAMI_even_2_5** — Informazione mutua automatica con $m = 2$, $\tau = 5$.
13. **IN_AutoMutualInfoStats_40_gaussian_fmmi** — Primo minimo della funzione di automutua informazione stimata con kernel gaussiano.
14. **SB_BinaryStats_diff_longstretch0** — Periodo più lungo di decrementi incrementali successivi.
15. **SB_MotifThree_quantile_hh** — Entropia di Shannon di due simboli successivi in una simbolizzazione a 3 lettere equiprobabili.
16. **FC_LinearTrend_residual_std** — Deviazione standard del residuo di una regressione lineare.
17. **FC_LinearTrend_coefficients** — Coefficienti (intercetta e pendenza) di un fit lineare alla serie temporale.
18. **IN_AutoMutualInfoStats_40_gaussian_fmmi** — Primo minimo della funzione di informazione mutua con 40 ritardi (kernel gaussiano).
19. **DN_OutlierInclude_p_001_mdrmd** — (ripetuta) Intervallo medio tra eventi estremi sopra la media.
20. **DN_OutlierInclude_n_001_mdrmd** — (ripetuta) Intervallo medio tra eventi estremi sotto la media.
21. **FC_LocalSimple_mean1_tauresrat** — Variazione nella lunghezza di correlazione dopo differenziazione iterativa.
22. **CO_Embed2_Dist_tau_d_expfit_meandiff** — Fit esponenziale alle distanze successive in uno spazio incorporato a 2 dimensioni.

TSFEL (Time Series Feature Extraction Library)[6] è una libreria che estrae automaticamente oltre 60 feature da serie

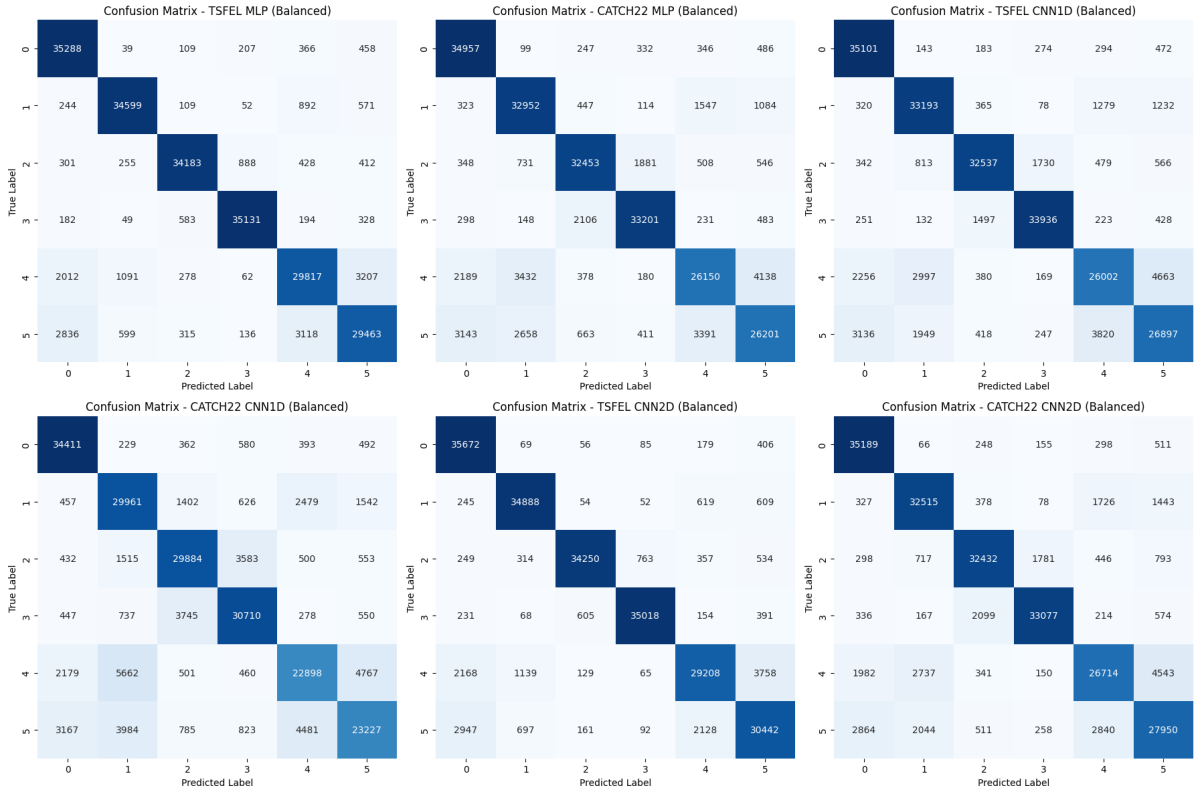

Figura 2. Matrici di confusione (n_epoche=100, lr=1e-4, batch=1024)

temporali nel dominio statistico, temporale, spettrale e frattale. Dato il carico computazionale necessario all'estrazione completa delle feature in tutti i domini è stato optato per utilizzare solo quelle derivanti dal dominio temporale e statistico. Un altro motivo per la scelta di questi domini è stato per via dei problemi derivanti dalla mancanza di dati e diverse frequenze di campionamento nello stesso dataset, l'estrazione delle feature nel dominio spettrale e frattale richiedevano finestre di stesso numero campioni per estrazione.

1.3 Architetture Reti neurali

Nel presente lavoro sono state sviluppate tre architetture neurali, costruite dinamicamente in funzione della forma dell'input. Le reti implementate includono una rete multi-layer perceptron (MLP), una convoluzionale 1D (CNN 1D) e una convoluzionale 2D (CNN 2D), ciascuna progettata per adattarsi a differenti rappresentazioni del dato: feature flattenizzate, sequenze temporali o strutture bidimensionali. La forma dell'input non solo determina le dimensioni dei layer, ma influisce anche sul comportamento semantico della rete, permettendo di ottenere modelli differenti a partire da una stessa architettura.

La rete **MLP** (SimpleMLP) riceve in input vettori completamente flattenizzati, dove ogni finestra temporale è rappresentata da un singolo vettore di dimensione pari a $d = n_{feature} \times n_{canali}$. Il primo layer completamente connesso ha quindi $d \times 256$ pesi e 256 bias, per un totale di $256(d+1)$ parametri. Segue un secondo layer da 256 a 64 unità, con 16448 parametri, e infine uno strato di output con $64 \times C + C$ parametri, dove C è il numero di classi.

La rete **CNN 1D** (SimpleCNN) opera tipicamente su input tridimensionali della forma (B, C_{in}, L), dove L rappresenta la lunghezza della sequenza temporale, e C_{in} il numero di canali. Tuttavia, è stata utilizzata anche per processare input flattenizzati di forma $(B, F \times C)$, considerandoli come sequenze 1D di lunghezza $F \times C$ con un singolo canale implicito. In entrambi i casi, la stessa architettura viene mantenuta, ma il comportamento appreso cambia radicalmente. La convoluzione 1D applica 64 filtri di kernel 3, con $64(3C_{in}+1)$ parametri. Dopo il max pooling, la sequenza si dimezza, e l'output è appiattito in un vettore di dimensione $64 \cdot (L/2)$. Il layer fully connected finale ha quindi $64 \cdot (L/2) \times C + C$ parametri. In questo modo, la CNN 1D viene utilizzata sia per rappresentazioni temporali, sia per rappresentazioni vettoriali interpretate come segnali sequenziali fittizi.

La rete **CNN 2D** (SimpleCNN2D) è stata progettata per trattare l'input come una mappa bidimensionale, della forma $(B, 1, F, C)$, dove F è il numero di feature per finestra e C il numero di canali. Tuttavia, questa stessa architettura viene impiegata anche per input originariamente in forma $(B, F \times C)$, che vengono reshaped a $(B, 1, F, C)$ per poter sfruttare le operazioni spaziali 2D. In questo modo si ottengono modelli diversi a partire dalla medesima architettura, sfruttando il fatto che la convoluzione 2D può cogliere strutture e dipendenze locali tra feature e canali. Il primo layer convoluzionale ha

$32 \times 1 \times 3 \times 3 + 32 = 320$ parametri, mentre il secondo layer ha $64 \times 32 \times 3 \times 3 + 64 = 18496$ parametri. Dopo il pooling, l'output ha dimensione $64 \cdot (F/2) \cdot (C/2)$, che viene appiattito e passato a un layer fully connected di 128 unità con $128 \cdot 64 \cdot (F/2) \cdot (C/2) + 128$ parametri, seguito da uno strato di output con $128 \cdot C + C$ parametri.

In sintesi, mentre la MLP è vincolata a rappresentazioni completamente flattenizzate, le architetture convoluzionali 1D e 2D sono state progettate per essere riutilizzate in scenari differenti, adattandosi automaticamente alla forma dell'input. La CNN 1D può essere utilizzata sia su dati temporali $(batch, canali, lunghezza)$ che su vettori flattenizzati trattati come sequenze. Analogamente, la CNN 2D consente di sfruttare le stesse operazioni convoluzionali sia su dati strutturati come matrici $(feature, canali)$, sia su input vettoriali opportunamente reshaped. Questa flessibilità permette di confrontare approcci diversi mantenendo la coerenza architetturale e valutare l'impatto della struttura dell'input sulle performance dei modelli.

1.4 Set di sperimentazione

Nella fase preliminare di sperimentazione, sono stati effettuati test sui modelli *SimpleMLP*, *SimpleCNN1D* e *SimpleCNN2D* al fine di valutarne il comportamento iniziale e ottenere indicazioni utili per l'impostazione della successiva fase di ottimizzazione tramite *grid search*. Tutti i criteri di addestramento sono stati lasciati alle impostazioni di default, ad eccezione di alcuni parametri scelti per garantire un training controllato e comparabile. In particolare il numero di epoche è stato fissato a 100, il *batch size* è stato impostato a 1024, la funzione di perdita adottata è la *CrossEntropyLoss* e il tasso di apprendimento è stato mantenuto costante a 1×10^{-4}. Gli esperimenti sono stati ripetuti utilizzando tre diversi seed iniziali (64, 256 e 571), al fine di verificare la robustezza delle prestazioni rispetto alla variabilità introdotta dalla randomizzazione nei processi di inizializzazione dei pesi e campionamento dei dati(1).

Per il training dei modelli è stata implementata una funzione personalizzata, "train_model" che esegue l'addestramento e valuta le prestazioni a ogni iterazione su un set di validazione separato. Durante ogni epoca, viene calcolata la perdita di classificazione (loss) mediante la funzione *CrossEntropyLoss*, che guida l'aggiornamento dei pesi tramite l'ottimizzatore *Adam* mentre viene calcolata anche l'accuratezza del modello. Entrambe le metriche sono monitorate e salvate in vettori (`loss_vec` e `acc_vec`) per permettere l'analisi sull'andamento dell'addestramento.

La selezione del modello migliore durante l'addestramento avviene identificando l'epoca in cui si è ottenuto il valore minimo della loss sul validation set. Il corrispondente stato del modello viene salvato e restituito al termine del training. E' stato scelto un approccio di addestramento orientato alla minimizzazione dell'errore, considerando che la loss è una misura continua e quindi più sensibile rispetto all'accuratezza.

Modello	Accuracy	Execution Time
MLP + TSFEL	0.91	2.17 s
MLP + Catch22	0.85	1.72 s
CNN1D + TSFEL	0.86	2.75 s
CNN1D + Catch22	0.78	2.07 s
CNN2D + TSFEL	0.91	3.50 s
CNN2D + Catch22	0.86	2.56 s

Tabella 2. n_epoche=100, lr=1e-4, batch=1024

Per ciascun esperimento vengono tracciati i valori minimi e massimi sia della loss che dell'accuratezza insieme alle epoche corrispondenti. Questo permette una valutazione comparativa tra architetture e impostazioni iperparametriche fornendo un quadro completo sull'efficacia di ciascun modello nella fase esplorativa.

2. Risultati

I modelli che si sono dimostrati più performanti sono stati quelli addestrati su feature estratte da TSFEL, abbastanza prevedibile dato che TSFEL, configurato con i domini statistico e temporale, estrae 45 feature da ogni finestra contro le 22 di Catch22. Per quanto riguarda i modelli, il modello MLP si è rivelato molto performante sia in termini di accuratezza che di dimensioni nel senso di "peso" computazionale, prevedibile in quanto la convoluzione è un'operazione molto dispendiosa dal punto di vista computazionale. La CNN2D è l'altro modello che ha fornito i risultati migliori(2) con un'accuratezza più o meno simile a quella dell'MLP anche se risulta più pesante rispetto a quest'ultimo(3), rendendo MLP l'architettura candidata più adatta al deployment su dispositivo.

Caratteristica	MLP	CNN2D
Accuracy	0.91	0.91
Execution Time	2.17 s	3.50 s
Numero di parametri	86,214	560,390
Dimensione del modello	0.3289 MB	2.1377 MB

Tabella 3. Confronto tra MLP + TSFEL e CNN2D + TSFEL.

A supporto di questa valutazione è stata stimata la memoria occupata dai modelli, sia in termini di dimensione dei pesi che di memoria necessaria per l'inferenza. La stima è stata effettuata tenendo conto del numero totale di parametri addestrabili, moltiplicato per il numero di byte utilizzati per rappresentare ciascun parametro (4 byte in caso di modelli standard, è stata comunque fornita la possibilità di calcolo in caso di quantizzazione ovvero 1 byte). Per una stima più completa è stata considerata la memoria necessaria a gestire l'input e le attivazioni intermedie durante l'inferenza introducendo un fattore di scaling ($1.5\times$) per rappresentare gli overhead delle operazioni attive e di mantenimento dei dati in memoria.

Successivamente è stata condotta un'analisi approfondita sui due modelli migliori: MLP e CNN2D, entrambi addestrati utilizzando le feature estratte con TSFEL. Per entrambi i

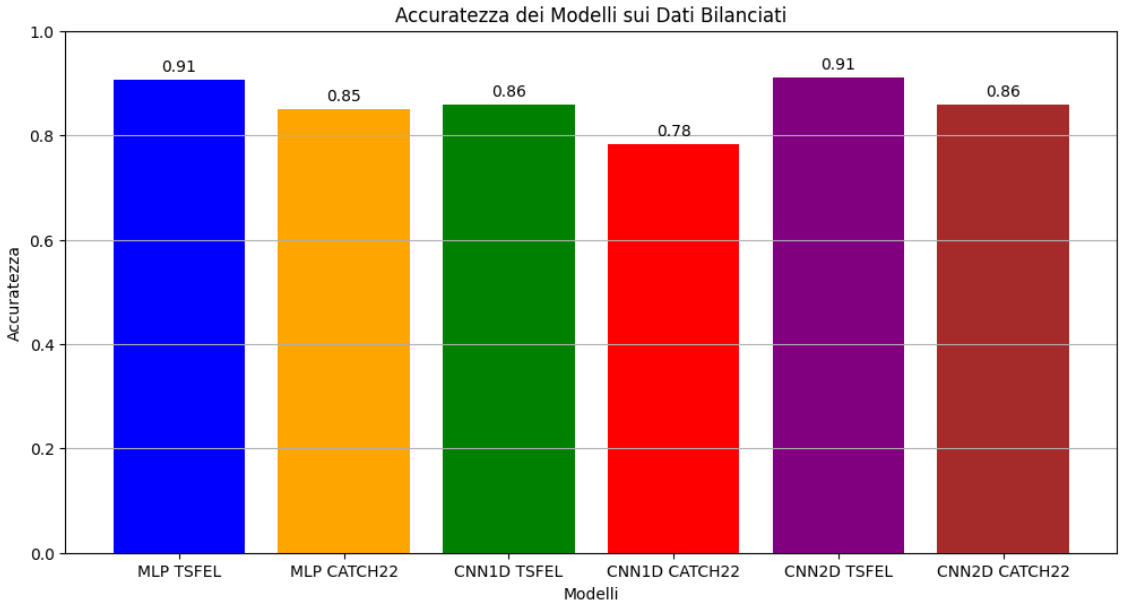

Figura 3. Performance modelli (n_epoche=100, lr=1e-4, batch=1024)

Tabella 4. Risultati Grid search

Metrica	**MLP**	**CNN2D**
Tempo medio di addestramento (s)	715	1430
Tempo medio di valutazione (s)	0.47	0.74
Accuratezza finale media	0.8878	0.8935
Accuratezza massima	0.9006	0.8998
Accuratezza minima	0.3464	0.6308
Loss finale media	0.4560	1.0500
Loss minima	0.2978	0.3001
Loss massima	1.7425	1.8474

modelli selezionati, **MLP** e **CNN2D**, è stata eseguita una *grid search* al fine di ottimizzarne gli iperparametri. I valori esplorati sono stati il learning rate: [1e-3, 1e-4, 1e-5] e il batch size: [512, 1024, 2048].

I risultati dimostrano come entrambi i modelli, seppur diversi nella struttura, raggiungano prestazioni molto simili nella configurazione ottimale, con il modello MLP che mostra una maggiore stabilità tra le diverse esecuzioni.

A supporto di questa analisi è stata applicata una visualizzazione alternativa delle matrici di confusione mediante l'utilizzo della libreria *NetworkX*. In questa rappresentazione le classi vengono disposte come nodi su un grafo orientato, con ogni errore di classificazione (mismatch) rappresentato come un arco che collega la classe reale (da $->$) a quella predetta (in $<-$).

Lo spessore degli archi è proporzionale, in modo esponenziale, al numero di errori tra due classi (per stesso verso di mismatch), rendendo molto visibili le confusioni più frequenti. Questa modalità di visualizzazione consente di evidenziare in modo chiaro le principali aree di debolezza del modello facilitando l'interpretazione degli errori e supportando l'eventuale revisione delle classi o delle feature impiegate.

3. Conclusioni

L'esplorazione condotta ha permesso di sfruttare gli estrattori *tsfel* e *catch22* non solo per il loro scopo originario, ovvero l'estrazione automatica di caratteristiche rilevanti da segnali temporali, ma anche come strumenti di unificazione tra dataset eterogenei. Nonostante le naturali differenze statistiche tra i dataset impiegati, dovute a variabili come utenza, frequenza di campionamento, qualità e struttura dei dati, nonché l'incompatibilità tra gli insiemi di classi di inferenza, è stato possibile identificare un sottoinsieme coerente di classi comuni che ha permesso l'integrazione di informazioni provenienti da fonti diverse, portando alla costruzione di due dataset bilanciati di circa 220.000 istanze (di cui ogni istanza rappresenta una finestra di 2 secondi) ciascuno.

Anche se i modelli *shallow* (come Support Vector Machine o Random Forest) sono generalmente più leggeri e rapidi nell'addestramento presentano una capacità di generalizzazione molto più limitata rispetto alle reti neurali, soprattutto nel riconoscimento di attività complesse.

In ambito HAR queste limitazione possono rivelarsi particolarmente penalizzanti per via di diversi fattori come la variabilità tra utenti o la frequenza di campionamento, tipologie di sensori, configurazioni dei modelli e condizioni ambientali, ecc.. I risultati ottenuti indicano la possibilità di miniaturizzare in modo efficiente reti neurali anche per applicazioni in tempo reale, dove l'inferenza deve avvenire in ambienti a risorse limitate come nel caso di dispositivi edge o embedded.

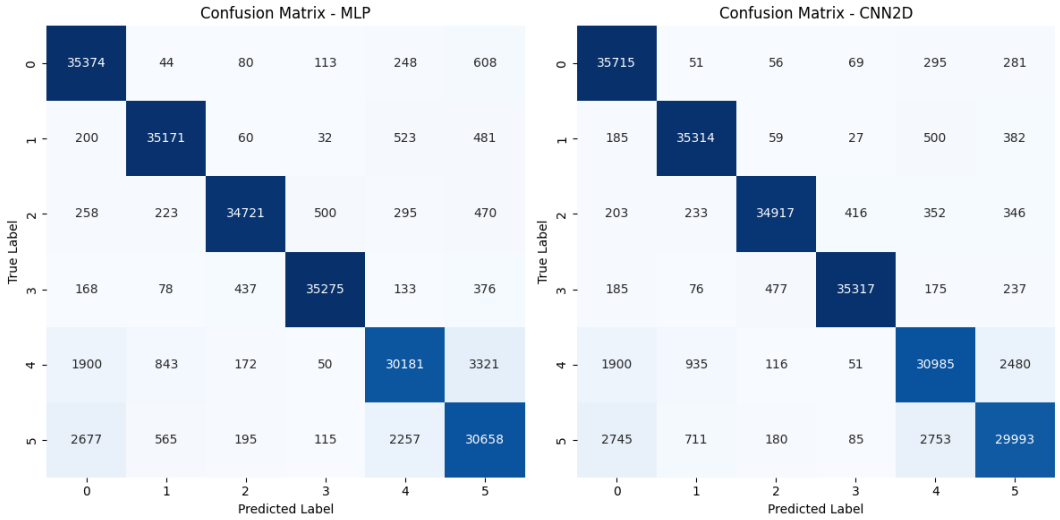

Figura 4. Matrici di confusione del best model selezionato tramite grid search

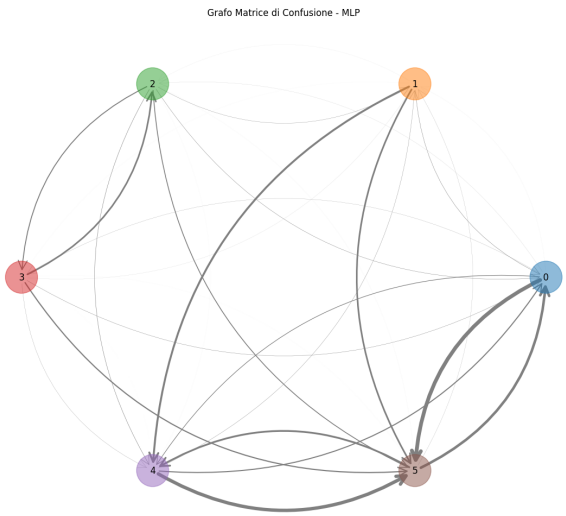

Figura 5. Grafo Mismatch MLP

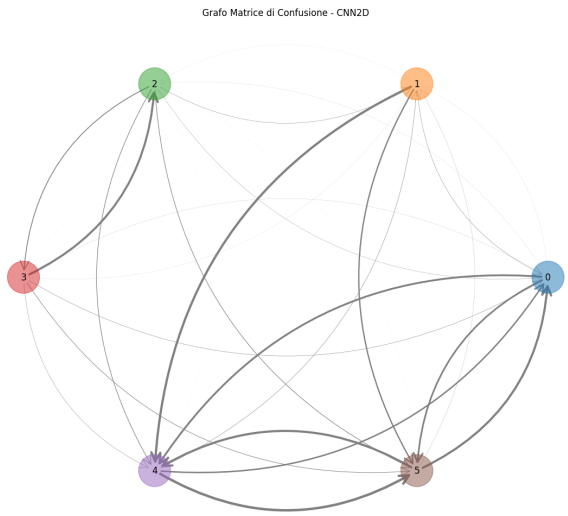

Figura 6. Grafo Mismatch CNN2D
Walking 0 - Jogging 1- Downstairs 2 - Upstairs 3 - Sitting 4 - Standing 5

Infine si nota una differenza importante tra le soluzioni MLP e CNN2D (56) di come il modello MLP tenda a scambiare molto di più, rispetto alla CNN2D, l'attività "camminare" con "fermo in piedi" e "stare seduto" con "fermo in piedi" mentre la CNN2D tende a scambiare di pù rispetto la cnn soprattutto il "camminare" con il "stare seduto".

Riferimenti bibliografici

[1] Mohammad Malekzadeh, Richard G. Clegg, Andrea Cavallaro, and Hamed Haddadi. Mobile sensor data anonymization. In *Proceedings of the International Conference on Internet of Things Design and Implementation*, IoTDI '19, pages 49–58, New York, NY, USA, 2019. ACM.

[2] George Vavoulas, Christos Chatzaki, Theodoros Malliotakis, Marios Pediaditis, and Manolis Tsiknakis. The mobiact dataset: Fall detection and classification with a smartphone. *International Journal of Monitoring and Surveillance Technologies Research (IJMSTR)*, 4(1):44–56, 2016.

[3] Henrik Blunck, Sourav Bhattacharya, Thomas Prentow, Mikkel Kjærgaard, and Anind Dey. Heterogeneity activity recognition. UCI Machine Learning Repository, 2015.

[4] Gary Weiss. Wisdm smartphone and smartwatch activity and biometrics dataset. UCI Machine Learning Repository, 2019.

[5] Carl H. Lubba, Sarab S. Sethi, Philip Knaute, Simon R. Schultz, Ben D. Fulcher, and Nick S. Jones. catch22:

Canonical time-series characteristics. *Data Mining and Knowledge Discovery*, 33(6):1821–1852, 2019.

[6] Marília Barandas, Duarte Folgado, Letícia Fernandes, Sara Santos, Mariana Abreu, Patrícia Bota, Hui Liu, Tanja Schultz, and Hugo Gamboa. Tsfel: Time series feature extraction library. *SoftwareX*, 11:100456, 2020.