RELAZIONE D'ESAME IN RETI DI SENSORI E DISPOSITIVI INDOSSABILI

OBIETTIVO

Programmazione di un sensore indossabile, capace di riconoscere che attività viene svolta grazie a misure provenienti da un sistema di misurazione inerziale (IMU)

Per rendere la spiegazione del codice più chiara possibile, penso sia fondamentale, prima di tutto, suddividere il problema in tre macrocategorie, le stesse che abbiamo sviluppato durante il corso, passo dopo passo.

1. ***Gestione Bluetooth del sensore indossabile***
   La prima parte del progetto ha come scopo quello di riuscire a connettere al Bluetooth il dispositivo che ci è stato consegnato, il Thingy52, un sensore dotato di caratteristiche per la raccolta dati.

2. ***Raccolta dei dati***
   Viene effettuata grazie al sistema di misurazione inerziale presente nel sensore, capace di raccogliere dati come accelerazione e velocità angolare, sui tre assi di riferimento.

3. ***Training dei dati***
   Quest'ultimo passaggio ha lo scopo di allenare la macchina su un set di dati, tramite algoritmi di Machine Learning, per renderla capace di fare previsioni su che attività viene svolta nel momento in cui indosso il sensore.

Chiarito il quadro generale del problema, è ora possibile affrontare il vero e proprio codice, ed entrare più nei dettagli su come è stato raggiunto l'obiettivo

GESTIONE BLUETOOTH DEL SENSORE

***LIBRERIE*: Bleak**
Bleak è una libreria progettata per poter interagire con dei dispositivi Bluetooth, ed è alla base di tutta la parte che riguarda la connessione del sensore

Altre keywords fondamentali sono:

- ASYNC: si utilizza quando voglio definire una funzione asincrona. Quando viene eseguita una funziona di questo tipo, viene restituita una coroutine, un'operazione che richiede del tempo, e che non blocca il programma. Quest'ultimo è quindi capace di procedere con altre operazioni, senza attendere che prima termini la coroutine

- AWAIT: usata quando richiamo una funzione asincrona. Mi permette di attendere che venga restituito un risultato, senza però interrompere il flusso del programma

***FUNZIONI*: (utils/utility.py)**

async def scan()
utilizza il metodo BleakScanner.discover per trovare dispositivi BLE in un certo raggio di ricezione del segnale, restituisce i dispositivi trovati.

def find
prende in input i dispositivi trovati e l'indirizzo/i di quello/i che voglio connettere. Un ciclo for trova tra i dispositivi, quelli con lo stesso indirizzo di quello/i dati in input, e li aggiunge a una nuova lista.

Metodi della classe Thingy: (classes/Thingy52Client.py)

per rendere "unico" ciascun dispositivo è stato utile creare una classe Thingy52Client. Ogni istanza di questa classe mi rappresenta un oggetto di tipo Thingy, con tutti i metodi e gli attributi propri di un sensore.
E' una classe che eredita tutte le funzioni di Bleak, la classe "padre", e prende come input un BLE device.

Attributi importanti per la parte di connessione sono

- **super().__init__(get_uuid(device)) inizializza correttamente la classe padre passando il MAC address del device.**
- **self.mac_address = device.address, attributo che mi permette di ottenere l'identificatore del sensore.**

def connect()
richiama il metodo connect della classe padre. C'è quindi un tentativo di connessione. Se va a buon fine la luce del led del sensore diventa verde grazie alla chiamata della funzione change_status (connessione riuscita), se l'operazione non va a buon fine mi viene restituito False e un messaggio di mancata connessione.

def change status
la prima che utilizza la Firm Architecture del Thingy52.
La firm architecture contiene le caratteristiche programmabili del Thingy, ciascuna identificata da uno specifico UUID, e ciascuna può essere utilizzata in lettura, scrittura o sola notifica.

Scrittura -> posso inviare dati al sensore

Notifica -> ricevo dati dal sensore, senza nessuna richiesta

Lettura -> richiedo dati al sensore

La change status prepara un pacchetto di 4 byte (3 per il colore, 1 per il tipo di luce), pacchetto contenente l'informazione "luce verde" nel caso di

un dispositivo connesso, “luce rossa” nel caso di un dispositivo che stia registrando dati.
Questo pacchetto viene poi scritto sulla caratteristica LED Characteristic, tramite il metodo **client.write_gatt_char**.

Questa prima parte di connessione si traduce nel main con 5 passaggi:

1) definisco l’indirizzo del sensore (scritto sul retro del Thingy)
2) richiamo della scan
3) richiamo della find (mi trova solo un dispositivo avendo solo un indirizzo)
4) l’output della find lo utilizzo per istanziare un oggetto di tipo Thingy
5) richiamo il metodo connect dell’oggetto

Alla fine di questa parte, avrò il mio sensore collegato al Bluetooth, pronto a trasmettere o ricevere dati

RICEZIONE DATI

Il Thingy52 possiede moltissime caratteristiche programmabili (ricezione dati sulla pressione, umidità, temperatura, luce del led, microfono…)
Quello che interessa a noi però è ricevere dati di accelerazione e velocità angolare grazie all’accelerometro e giroscopio presenti nell’IMU del sensore. Saranno i dati principali che permetterano all’algoritmo di Machine Learning di capire che attività sta svolgendo il soggetto.

Dalla Firm Architecture:

Name	UUID	Type	Data	Description
Base UUID	EF68xxxx-9B35-4933-9B10-52FFA9740042			
Raw data characteristic	0406	Notify	18 bytes	Motion sensor raw data: • Accelerometer ◦ int16_t - x [G] (6Q10 fixed point) ◦ int16_t - y [G] (6Q10 fixed point) ◦ int16_t - z [G] (6Q10 fixed point) • Gyroscope ◦ int16_t - x [deg/s] (11Q5 fixed point) ◦ int16_t - y [deg/s] (11Q5 fixed point) ◦ int16_t - z [deg/s] (11Q5 fixed point)

- L'UUID identifica univocamente la caratteristica Raw Data (accelerazione e velocità angolare)

- La caratteristica è di tipo notify: il Thingy trasmette dati all'utente, senza che venga fatta alcun tipo di richiesta (al contrario la tipologia read necessita della richiesta di trasmissione da parte dell'utente)

- 18 bytes è il numero massimo di bit richiesti. Nel nostro caso basteranno 12 bytes (i dati trasmessi sono in bit, andranno poi trasformati per poterli rendere leggibili)

Ora, fatta questa premessa, la prima cosa da impostare è il file su cui andranno i dati che ricevo dal sensore.

`def save_to()` (metodo Thingy)
metodo della classe Thingy, associa a self.recording_name il nome su cui voglio scrivere i dati, nel mio caso sarà del tipo
F1-58-6C-E2-D8-44_filename.csv

`def motion_characteristics()` (utility.py)
prepara il sensore a inviare dati, configurando i parametri di movimento. Lo fa restituendo un pacchetto di 9 byte (4H B), contenente i valori di default dei parametri (il più importante è la motion_processing_unit, stabilisce quanti dati al secondo vengono processati)

def raw_data_callback() (metodo Thingy)
Prende in input il trasmettitore e il dato. Viene utilizzata la struct.unpack per decodificare coppie di byte, ciascuna rappresentante una grandezza fisica diversa (accelerazione su x, su y…)
Il dato decodificato viene scritto sul path per arrivare al file F1-58-6C-E2-D8-44_filename.csv, e viene anche scritto sul terminale in tempo reale

receive_inertial_data (metodo Thingy)
Con questa funzione posso infine avviare il processo di notifica dei dati

1) imposto i parametri di movimento scrivendo sulla caratteristica TMS_CONF_UUID tramite la .write_gatt_char
2) apro il path al file csv, se non esiste ancora lo creo
3) avvio le notifiche, la TMS_RAW_DATA_UUID trasmette il dato, ogni volta che un dato viene trasmesso è invocata la raw_data_callback
4) invoco la change_status, impostando il sensore su luce rossa (recording)
5) appena si presenta un errore del tipo asyncio.CancelledError, le notifiche del dato vengono bloccate, e il Thingy è disconnesso dal bluetooth. Il file viene chiuso.

Tutta questa parte si traduce nel main in due passaggi: dare all'utente la possibilità di scrivere in input da tastiera il nome del file, e richiamare la receive_inertial_data

TRAINING DEI DATI

Quello che vorrei che l'algoritmo di Machine Learning imparasse a riconoscere sono:

- palleggio nel basket (il sensore ho pensato di posizionarlo sul braccio destro, sull'avambraccio. Il palleggio presusspone l'utilizzo della mano destra)

- tiro nel basket (qualsiasi tipo di tiro che presuppone l'utilizzo del braccio destro)

PREPARAZIONE DEL DATASET (dataset.py)

Avviene grazie alla classe CSVDataModule, che prende in input alcuni parametri importanti

- **BATCH SIZE**: utilizzato quando si parla di addestramento di modelli di Machine Learning. Mi dice quante finestre temporali vengono elaborate dal modello in un'unica iterazione

- **K-FOLDS**: il modello usato nel codice è il K-fold Cross Validation. Il dataset viene suddiviso in 3 fold (nel nostro esempio), e vengono fatte k iterazioni. A ogni iterazione utilizzo un fold diverso per il training e la validazione, in questo modo il modello lavora sempre con dati diversi. Il modello vede i dati di training e si allena a fare previsioni, aggiustando poi i parametri che gli consentono di fare meno errori. I dati di validazione servono per capire le performance del modello durante il training, sono i dati principali su cui posso fare considerazioni sull'efficacia del modello. Il concetto importante è

che per queste fasi vengono usati sempre dati diversi, questo per evitare che il modello impari sulla base di bias e pattern.

- **WINDOW-SIZE**: lunghezza delle finestre temporali con cui seleziono i dati, scelta sulla base delle azioni che voglio far riconoscere al modello

- **OVERLAP**: sovrapposizione delle finestre temporali, parametro utile per mantenere continuità temporale e quindi non perdere dipendenze tra i dati

- **SAMPLE-RATE**: mi dice quanto velocemente il modello riceve i dati

FUNZIONI: (dataset.py)

def windowing()
ad ogni iterazione viene creata una finestra temporale sovrapposta a quella precedente. La funzione mi restituirà un array numpy per i valori delle finestre e un array per le etichette (si pressuppone che le righe di una finestra abbiano tutte la stessa label)

def create-dataset()
grazie alla libreria pandas viene letto il file csv. Viene aggiunta una colonna al file con l'etichetta dell'attività svolta, e viene poi richiamata la funzione windowing, per suddividere in finestre temporali i dati all'interno del file csv. La funzione labels_econding è chiamata, per associare a ciascuna classe (balling, shooting) un'etichetta numerica. Mi vengono restituiti l'array dei valori nel tempo delle finestre, e l'array delle etichette numeriche

def setup()
imposto il metodo di k-fold cross validation, passando il numero di fold che voglio utlizzare, e randomizzando anche i dati, questo lo faccio per evitare possibili predizioni del modello basate solo sulla sequenza di dati. Successivamente splitto i dati, così da ottenere gli indici associati alle righe di training, e gli indici delle righe di validation.

TRAINING E VALIDATION: (CNN.py)

In questo file quello che viene fatto è impostare le caratteristiche delll'algoritmo di training e validation.
Uso una rete neurale, i dati passano attraverso più layer e alla fine mi viene detta la probabilità con cui il modello ha predetto le classi in modo corretto.

Questi sono i parametri di valutazione del modello:

Train_loss_epoch e train_loss_step → errore tra le predizioni del modello e le etichette reali, calcolate durante il training, più basso è più, bravo è il modello a fare previsioni sui dati di addestramento.
Il parametro posso vederlo per un singolo batch (visione a bassa scala) o per epoca (tutti i batch, per visione più generale e a larga scala)

Forward → applico un'operazione di propagazione in avanti attraverso la rete neurale per arrivare all'output finale, è la parte centrale del modello, che sfrutta la rete a più layers.

Val_loss_epoch e val_loss_step → fa la stessa cosa del parametro train_loss ma sui dati di validazione, ottengo informazioni sull'efficacia

del modello durante la validazione, su epoce intere o su singoli batch rispettivamente.

On-validation → calcola il **precision-score** e **l'f1-score**

Per questi due parametri va introdotto il concetto di CONFUSION MATRIX. È una tabella che aiuta a capire meglio l'efficacia che ha avuto il modello di classificazione, viene fatto grazie a un confronto tra le etichette previste e quelle reali.
Nel mio caso (classificazione binaria, matrice 2x2), avrei potuto ottenere 4 possibili risultati dal modello:

- True Positive: modello ha previsto correttamente che stessi palleggiando
- False Positive: il modello ha sbagliato e ha predetto "palleggio" quando in realtà stavo tirando
- True Negative: il modello ha previsto correttamente che stessi tirando
- False Negative: il modello ha sbagliato e ha predetto "tiro" quando in realtà stavo palleggiando

Precision-score → la percentuale di predizioni corrette tra tutte quelle fatte dal modello come positive.

F1-score → compromesso tra precision-score e recall
Recall → capacità del modello di catturare tutti i positivi reali

Test loss → quanto bene fa il modello su dati che non ha mai visto prima, ovvero sui dati di test

Questa era la descrizione generale di come è stato preparato il dataset, e di quali parametri sono stati usati per valutare la bravura del modello. Ora è possibile vedere che risultati sono stati ottenuti nel mio caso specifico

RISULTATI

Analisi dei due movimenti da riconoscere:

per capire come impostare alcuni parametri con cui avrebbe lavorato il modello e aumentare quindi al massimo le prestazioni bisogna fare alcune considerazioni sulle due attività su cui avrei registrato i dati.

Il palleggio è un movimento molto ripetitivo, un ciclo intero dura all'incirca un secondo, ed è possibile variarlo molto. Ho cercato di registrare dati:

- Cambiando mano del palleggio
- Cambiando la velocità del palleggio
- Muovendomi durante il palleggio

Tutto questo per raccogliere dati i più generali possibili.

Per quanto riguarda il tiro invece, è sempre molto ripetitivo, un ciclo intero può però durare anche 3 secondi, e qui la variabilità è molto minore:

- Solitamente il tiro si esegue rimanendo nella stessa posizione con i piedi
- La tecnica del tiro bene o male è sempre la stessa, così come la velocità di esecuzione

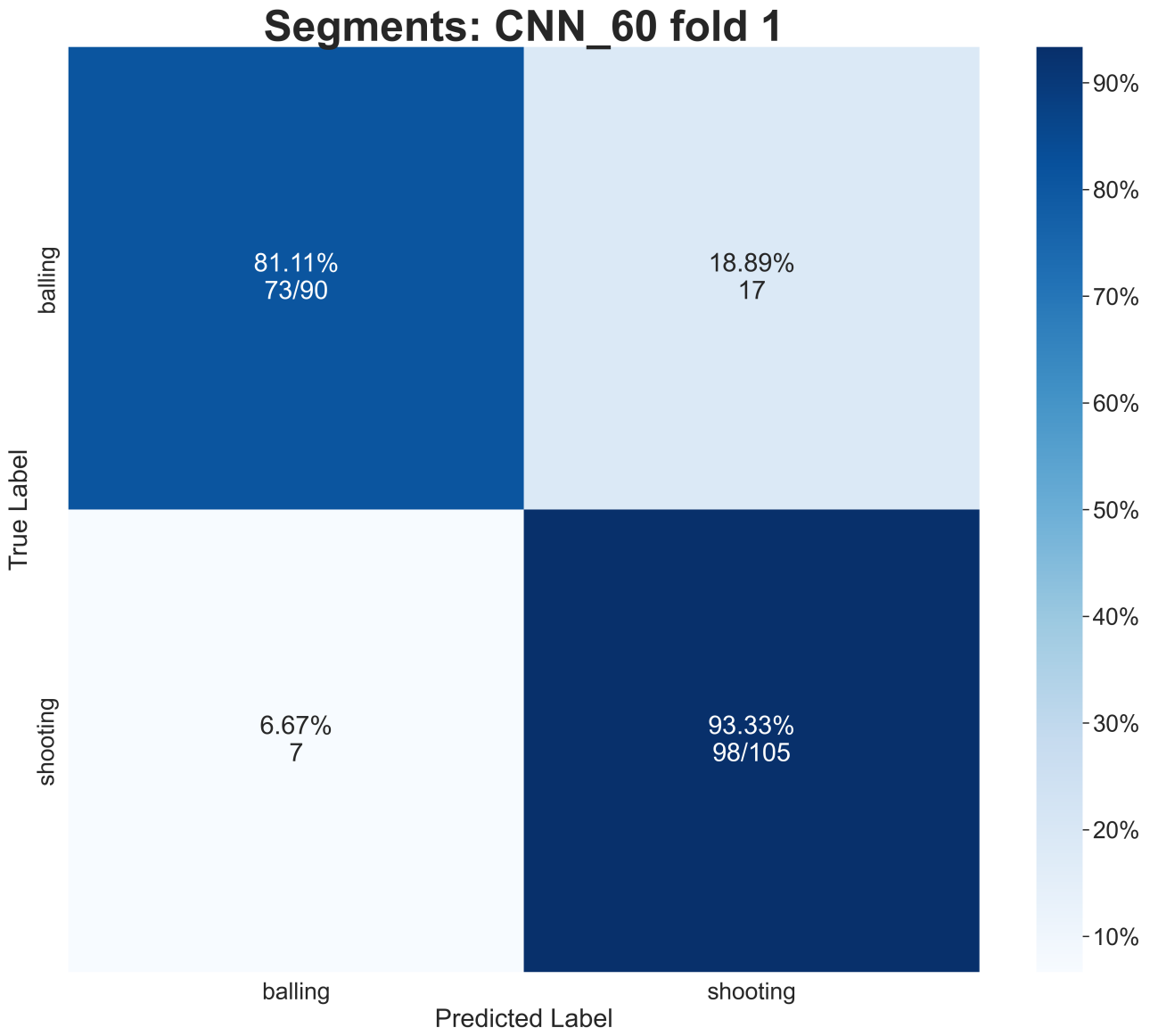
Non ho potuto quindi variare troppo il movimento, mi aspettavo dal modello che riuscisse a prevedere in modo molto efficace questa classe.

Per la decisione delle finestre temporali, volevo trovare un intervallo che potesse comprendere entrambi i cicli completi delle due attività. Ho optato

per finestre di 2 e 3 secondi. Il caso delle finestre di un secondo ho comunque voluto provarlo, per farmi comunque un'idea delle prestazioni. Per la decisione dell'overlap delle finestre invece, ho optato per sovrapposizioni del 25% e del 50%. Volevo delle sovrapposizioni che potessero darmi una continuità temporale intermedia, con l'overlap a zero sapevo che il modello avrebbe fatto molta fatica.

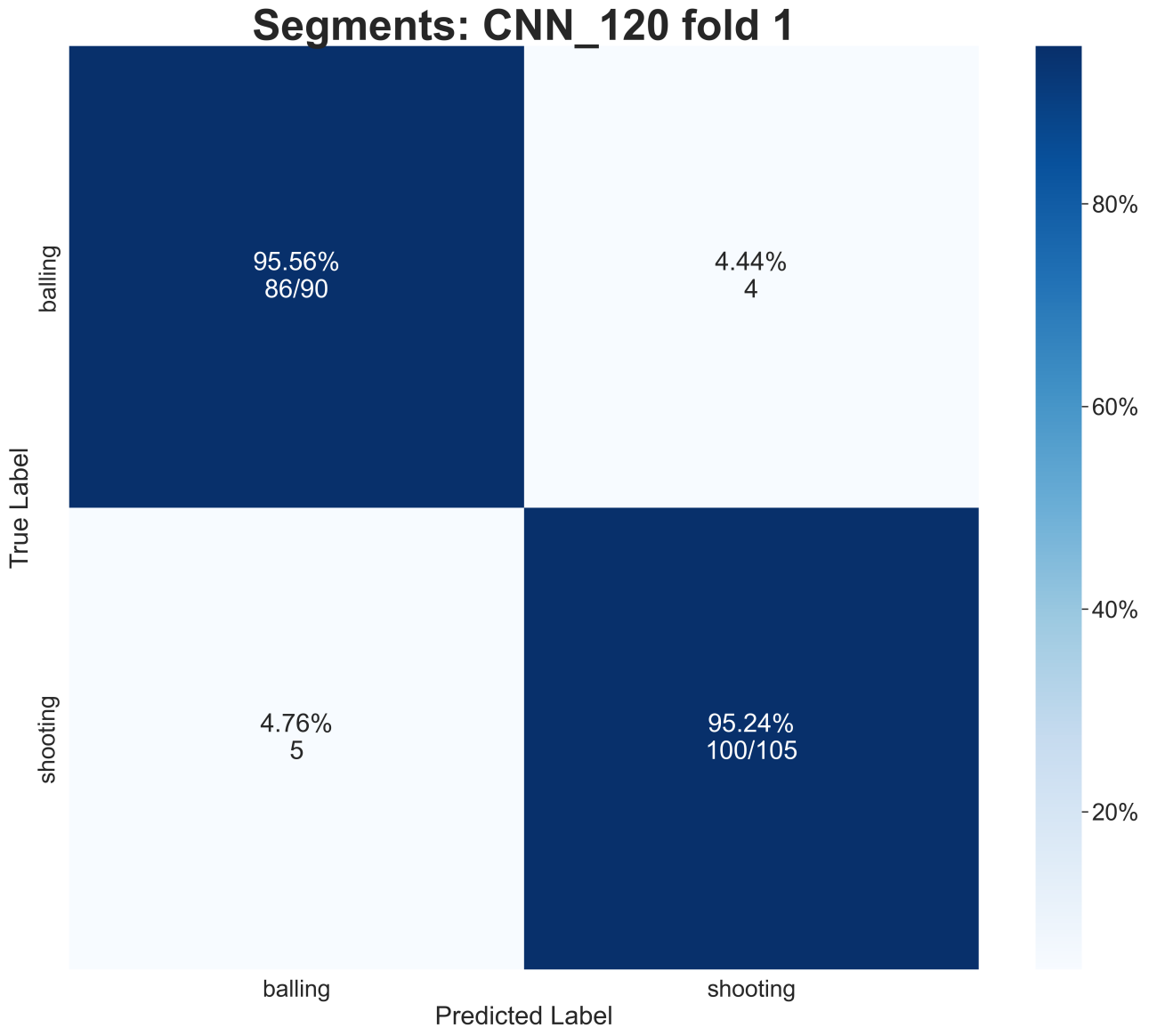
I batch sono stati tenuti a 32 in tutte le mie prove, così come la sampling-frequency e il numero di fold.

CASO 1 secondo di finestre e 25% di overlap:



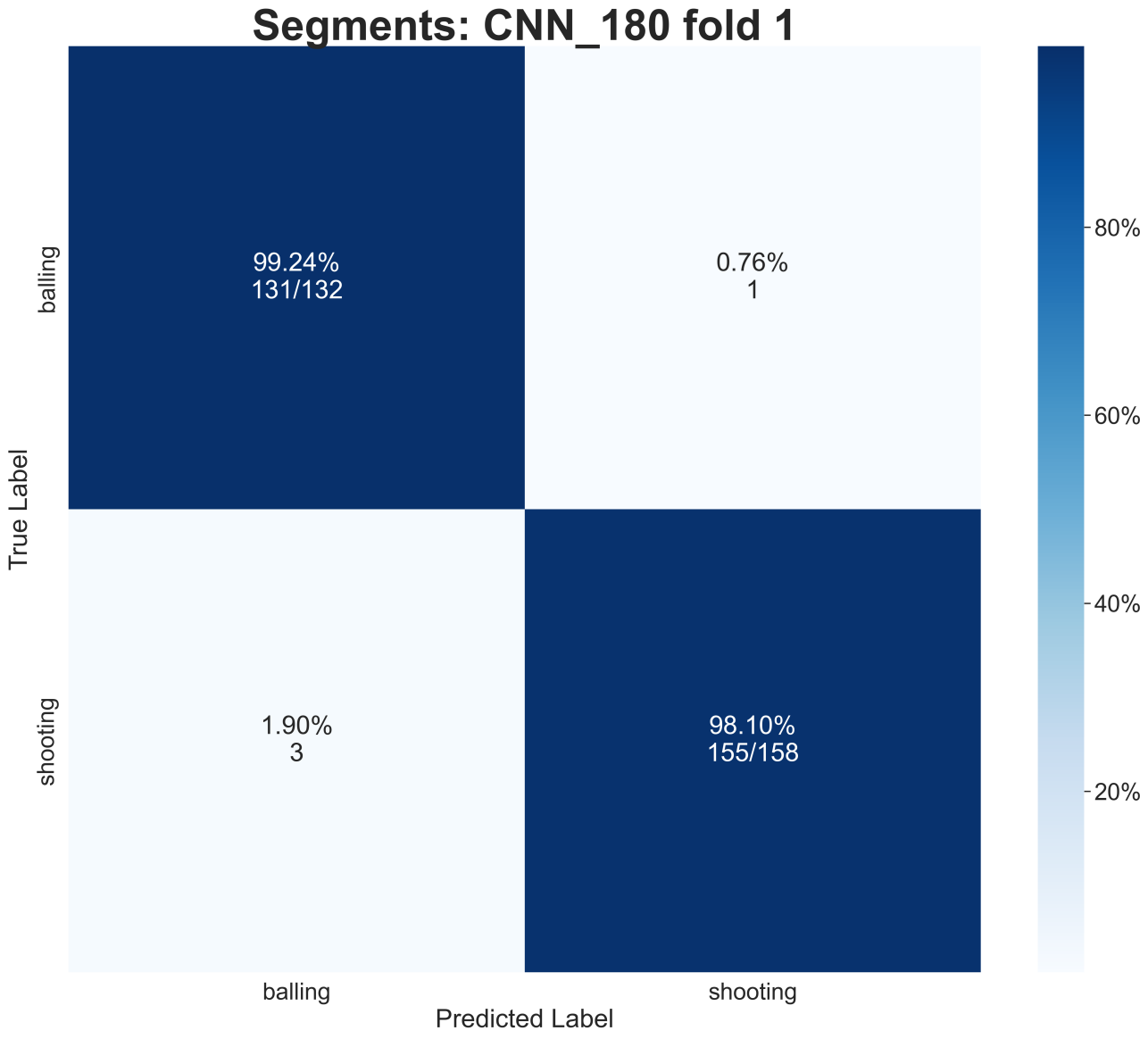
Questa è la matrice che ho ottenuto, il modello è stato efficace nel predire lo shooting, molto poco efficace nel predire balling. Durante le epoche ho spesso ottenuto valori di f1-score sotto lo 0.900, molto basso rispetto ad altri casi su cui ho lavorato. I parametri di train_loss_epoch e val_loss_epoch non mi soddisfacevano e non presentavano molti miglioamenti.

CASO 2 secondi di finestre e 25% di overlap:



In questo caso ero molto più soddisfatto. Il modello ha avuto prestazioni migliori. L'f1 score ha raggiunto valori elevati di 0.979, il train_loss_epoch si era stabilizzato a 0.317 con il passare delle epoche, l'unico parametro che volevo ancora diminuisse era il val_loss_epoch, molto stabilizzato a 0.347, valore maggiore rispetto ad altre variabili in gioco.

CASO 3 secondi di finestre e 50% di overlap:



Qui ho ottenuto quello che speravo. Il modello è stato molto efficace a fare previsioni, sono riuscito a far abbassare e stabilizzare il val_loss_epoch a 0.316.

CONCLUSIONI

In sinstesi, costruire questo modello per il riconoscimento delle attività ha dimostrato quanto sia importante scegliere e adattare correttamente i parametri in base alle caratteristiche del caso studiato. Il fatto che le attività prese come esempio avessero caratteristiche molto diverse tra loro ha facilitato l'addestramento del modello migliorandone l'efficacia nel fare previsioni.

Se avessimo lavorato con un caso più complesso, nell'ambito medico per esempio, tutto ciò non sarebbe bastato, ma avremmo dovuto raccogliere una quantità molto maggiore di dati, per consentire di allenare il modello su più scenari, per generalizzare meglio, e per arrivare ad applicare questo tipo di tecnologie in contesti reali in modo sicuro per le persone.