ESAME RETI DI SENSORI

INDICE:

Simone monzoni

VR486241

1 OBIETTIVO

Il progetto, si pone l'obiettivo di sfruttare un sensore thingy-52 (prodotto dall'azienda Nordic Semiconductor https://www.nordicsemi.com), per raccogliere i dati necessari alla creazione di una CNN, in grado di classificare in real-time il movimento di un soggetto, suddiviso in: corsa, camminata e assenza di movimento.

2 INTRODUZIONE AL THINGY-52

Il thingy-52 è un dispositivo multi-sensore che usa la tecnologia BLE per la comunicazione dei dati. Questa infatti, è la tecnologia più consigliata per sensori indossabili e in generale per comunicazioni che necessitano di un basso consumo energetico.

La connessione del sensore con il computer è stata realizzata sfruttando Bleak, libreria molto utile per connettersi a dispostivi BLE.

Il dispositivo essendo multi-sensore può fornire dati riguardanti diverse caratteristiche: audio, pressione atmosferica, temperatura etc… ma in questo progetto sono stati raccolti esclusivamente dati inerziali di accelerazione e rotazione.

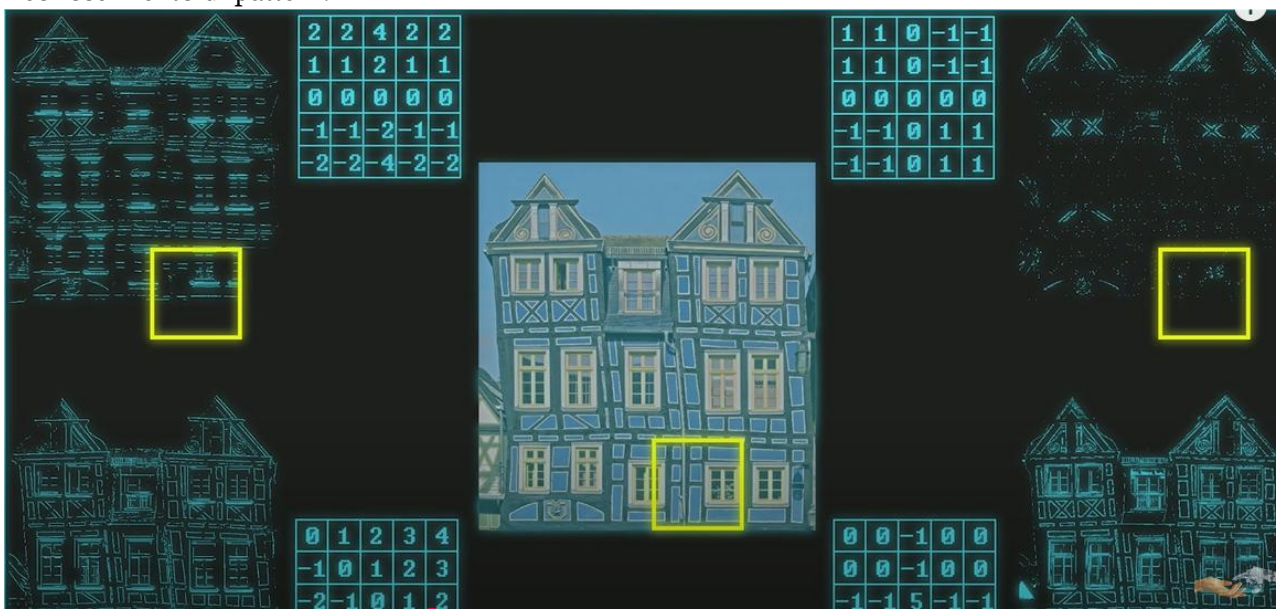
Ciascun tipo di dato è caratterizzato da un numero identificativo (UUID), per permettere all'utente di selezionare esclusivamente i tipi di dato d'interesse. Queste informazioni sono raccolte al seguente link.

Inoltre, ciascun dato viene trasmesso con una codifica diversa, e quindi necessità di una conversione binario→decimale personalizzata.

3 INTRODUZIONE ALLE CNN

Una CNN (convolutional neural network) è una architettura di rete per il deep learning che sfrutta l'operazione matematica della convoluzione. Questo tipo di reti è molto utilizzato per attività di classificazione. Le CNN infatti utilizzano filtri (matrici quadrate chiamate anche kernel) per il

riconoscimento di pattern.



La rete è composta da diversi strati (layer); i primi si occupano di riconoscere pattern e sono i cosiddetti layer convoluzionali, accompagnati da layer di pooling, necessari alla riduzione della complessità del dato. Infine, troviamo layer fully-connecteed che funzionano come una classica rete neurale, che però, riceve in input l'output del layer di pooling invece che il dato grezzo.

In questo progetto abbiamo utilizzato la validazione k-fold, che consiste nella suddivisione dei dati in k gruppi, 3 nel nostro caso. Si procede quindi a svolgere 3 training in cui ogni volta, 2 gruppi vengono usati per il training, e 1 per la validation, alternandoli ogni volta.

Abbiamo quindi 3 training diversi, e gli output di conseguenza saranno 3. A questo punto si può sia selezionare uno dei 3 output sia unirli per ottenere informazioni da tutti e 3.

Nel nostro caso è stato solamente selezionato 1 output dei 3.

4 PIPELINE

- connettere il thingy-52 al pc tramite bleak.
- Raccogliere dati inerziali con il sensore e salvarli in file csv. I dati registrarti sono etichettati in base al tipo di registrazione: corsa, camminata, nessun movimento
- Creare una CNN
- Allenare la rete con i dati precedentemente raccolti
- Importare il modello creato in uno script pyhton per la classificazione in real-time

5 ANALISI DEL CODICE

Il codice relativo alla raccolta dati si divide in 2 sezioni principali: il main e la classe.

Qui a lato troviamo il main, in cui dopo esserci connessi al dispositivo, lo istanziamo come un oggetto della classe Thingy52Client.

Questo gli permette di acquisire attributi e funzioni definite nel file della classe.

Un'esempio è la funzione connect che connette il dispositivo al pc.

Le successive righe chiamano funzioni che permettono di ricevere, e poi salvare, i dati in un file dal nome personalizzabile dall'utente.

main_class_example.py

```python
from classes.Thingy52Client import Thingy52Client
from utils.utility import scan, find
import asyncio


async def main():
    my_thingy_addresses = ["EB:71:55:86:18:D4"]
    discovered_devices = await scan()
    my_devices = find(discovered_devices, my_thingy_addresses)

    thingy52 = Thingy52Client(my_devices[0])
    await thingy52.connect()
    thingy52.save_to(str(input("Enter recording name: ")))
    await thingy52.receive_inertial_data()


if __name__ == '__main__':
    asyncio.run(main())
```

Le funzioni non importate dalla classe sono importate dal file utility.

Nell'immagine a lato troviamo le prime righe del secondo file, in cui viene definita la classe.

Si nota che la classe è "figlia" della classe BleakClient, questo ci permette di ereditare diversi attributi e funzioni.

Creare questa classe ci permette però di personalizzare ulteriormente BleakClient alle nostre necessità.

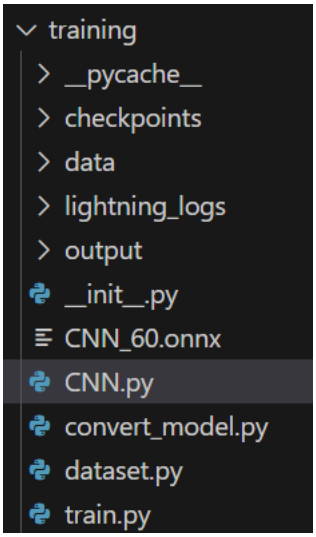
Alcune funzioni, tipo connect possono essere sovrascritte: importiamo la funzione della classe padre ma con la possibilità di fare cambiamenti.

```python
class Thingy52Client(BleakClient):

    def __init__(self, device: BLEDevice):
        super().__init__(device.address)
        self.mac_address = device.address

        self.model = ort.InferenceSession('training/CNN_60.onnx')
        self.classes = [ "run","stand","walk"]

        # Data buffer
        self.buffer_size = 60
        self.data_buffer = []

        # Recording information
        self.recording_name = None
        self.file = None


    async def connect(self, **kwargs) -> bool:
```

Una volta arrivati a questo punto abbiamo come output dei file csv. In questo caso i file sono stati salvati con nomi diversi in base al tipo di attività registrata: run, walk e stand.

Questo permetterà alla CNN di avere delle etichette sui dati di input consentendo di svolgere un training supervisionato.

La struttura del codice per la parte training della CNN è quella in figura.



Nella cartella possiamo trovare un file che crea la rete, CNN.py, un file che lancia l'addestramento della rete, train.py, uno che prepara i dati e li suddivide adeguatamente, e infine un file che converte la rete allenata in un file onnx necessario per la classificazione real-time.

Oltre ai file troviamo diverse sottocartelle:

output contiene le confusion matrix di ogni fold, realizzate dopo il training, checkpoints contiene i pesi migliori di ciascuna fold che poi verranno passati al file convert_model.py, infine data contiene i file csv creati.

6 MODIFICHE APPORTATE AL CODICE

Tralasciando le modifiche necessarie al funzionamento del codice nel mio pc (come sostituire cpu a gpu) ho apportato diverse modifiche per migliore l'efficienza del codice rispetto al tipo di classificazione da me desiderato.

La prima differenza nel mio codice è la durata della finestra, impostata a 2 secondi, con un overlap di 1 secondo. Ho fatto ciò per allenare la rete con informazioni più ampie nel tempo, ritengo infatti che movimenti come la camminata siano abbastanza lenti, e quindi che vengano descritti più correttamente in una finestra di 2 secondi.

L'overlap è stato inserito per non perdere eventuali variazioni importanti nei dati inerziali al confine dei due secondi. Per esempio, il movimento con cui stacco il piede da terra durante la camminata, avrà un'accelerazione importante lungo un'asse, e con l'overlap riduco il rischio di spezzare l'informazione in due segmenti separati.

Una modifica necessaria è stata quella di abbassare la frequenza di campionamento del sensore, da 60hz a 30hz, infatti, mi sono reso conto che a 60hz il mio pc faticava a rimanere al passo con i dati durante la registrazione, e ciò portava alla perdita di informazione. Abbassandola sicuramente perdo precisione, ma guadagno fluidità e coerenza nei dati.

L'ultima modifica è nella seguente riga di codice:

```
df["label"] = file.split("_")[1].split(".")[0].split("-")[0]
```

I dati da me raccolti infatti, erano organizzati in questo modo:

questo mi ha dato il vantaggio di poter raccogliere i dati in più sessioni, mantenendo però il nome dell'etichetta invariato.

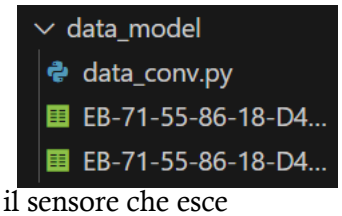
Infatti, mentre per la registrazione fermo in piedi non ho avuto problemi a reggere 10 minuti di registrazione continua, per la corsa sicuramente li avrei avuti.

Inoltre, camminata e corsa sono molto diversi da persona a persona e avevo quindi la necessità di raccogliere dati da soggetti diversi, i miei genitori in questo caso.

Così facendo, la rete non viene allenata a riconoscere la mia corsa rispetto alla mia camminata, bensì è in grado di generalizzare a diverse andature.

Preciso che i dati totali per ciascuna classe sono circa uguali, più o meno 15.000 campioni ciascuno.

È stata inoltre aggiunta una cartella contenente un file: data_conv.py, per la modifica dei dati raccolti.



È successo infatti più volte di avere problemi nella raccolta dei dati: per esempio il sensore che esce dalla posizione fissata, oppure l'interruzione della raccolta dei dati troppo ritardata. In questi casi i file sono stati tagliati degli ultimi 300/400 campioni per essere sicuri di avere dati ad alta qualità per il training.

7 ESECUZIONE DEL PROGETTO

Per la raccolta dati io e i miei genitori abbiamo corso e camminato per circa 3 minuti ciascuno, con il sensore Thingy-52 posizionato sopra la caviglia, sempre con lo stesso orientamento.

Il sensore è stato fissato il più stretto possibile, per evitare errori di misura e garantire la massima coerenza con il movimento del piede.

Dopo aver fatto le prime prove mi sono reso conto che il sensore si scollegava molto facilmente dalla connessione o, meglio, rimaneva connesso, ma la trasmissione dei dati diventava intermittente, o addirittura si bloccava.

È stato necessario quindi seguire il più possibile con il pc il sensore, cercando di rimanere entro un certo raggio, per evitare problemi nella registrazione.

La registrazione è stata avviata dopo l'inizio del movimento per evitare di avere dei campioni etichettati come corsa o camminata, contenenti però misurazioni statiche.

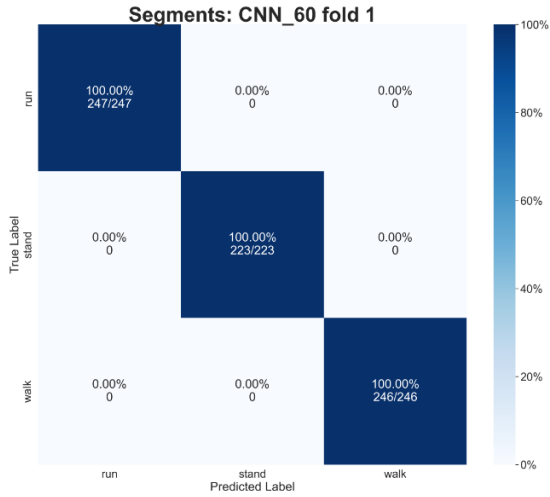
Il sensore inizialmente era stato posizionato all'altezza del tricipite, ma durante le prime registrazioni mi sono reso conto che non tutti muovono le braccia durante la camminata, e soprattutto che la quantità di artefatti era molto superiore.

Infatti, per registrazioni di diversi minuti è capitato diverse volte di fare movimenti non naturali con le braccia: spostare un indumento, asciugarsi sudore, salutare col braccio …

Ho dunque ritenuto più adeguato a questo compito il posizionamento del sensore sopra la caviglia dove, inoltre, è risultato molto più semplice da fissare.

8 ANALISI E DISCUSSIONE DEI RISULTATI

Dopo la raccolta dei dati ho eseguito il training della rete che ha dato ottimi risultati.



La rete ha riconosciuto il 100% delle volte ogni classe di input.

Questo ovviamente è l'obiettivo desiderato, quindi ho creato il file onnx e l'ho importato nel codice per la classificazione real-time.

Di nuovo il risultato è stato molto buono, la classificazione real-time funziona.

La parte più sorprendente è l'abilità nel riconoscere la differenza tra corsa e camminata. Infatti, non avevo dubbi che la rete sarebbe stata in grado di riconoscere l'assenza di movimento dagli altri due gesti, in quanto la differenza nei dati inerziali è evidente; ma la precisione con cui riconosce, per esempio, il momento in cui passo, in maniera fluida, da corsa a camminata mi ha stupito.

Infatti, la differenza tra le due non è così evidente rispetto alla differenza con l'assenza di movimento, quindi mi sarei aspettato un pò di imprecisione.

Preciso che riconosco che, sebbene la differenza tra corsa e camminata non sia tanta, ci sono task più complessi da distinguere.

Prima di questa classificazione, infatti, ho provato a studiare il funzionamento della rete con altri task più facili da registrare, come camminare sul pavimento o sulle scale.

Nonostante le registrazioni fossero più semplici, la classificazione risultava molto più complicata, non superando mai il 94% di precisione in entrambe le classi.

Le motivazioni sono probabilmente due:

1) la differenza nei dati inerziali è minore rispetto al caso corsa/camminata.
2) essendo una prova, i dati sono stati raccolti con più superficialità.

La differenza nei due risultati mi ha fatto capire l'importanza nel dare a queste reti dati di alta qualità.

Aggiungo che, avendo usato diverse andature nel training, la rete è stata in grado di generalizzare, arrivando a riconoscere le differenze tra corsa e camminata in andature mai viste prima.

9 CONCLUSIONI

Sono dunque riuscito a creare una CNN per la classificazione di corsa, camminata, e assenza di movimento, partendo da zero, raccogliendo dati personalmente attraverso un sensore e dei codici per la trasmissione e il salvataggio dei dati.

Ritengo dunque che il progetto si possa definire riuscito.