1 Prefazione

1.1 scopo del progetto

L'obbiettivo di questo progetto è la realizzazione di una modello di machine learning in grado di apprendere le relazioni tra i dati fruiti da una scheda embeded dotata di accelerometro, giroscopio e magnetometro, e la posizione spaziale di un braccio umano. Il problema che ci si è posti di risolvere poteva essere risolto in modo deterministico ma per lo scopo del progetto si voleva risolvere il problema tramite un algoritmo di ML, per poter verificare l'efficacia di questo metodo per la risoluzione di un problema con soluzione nota.

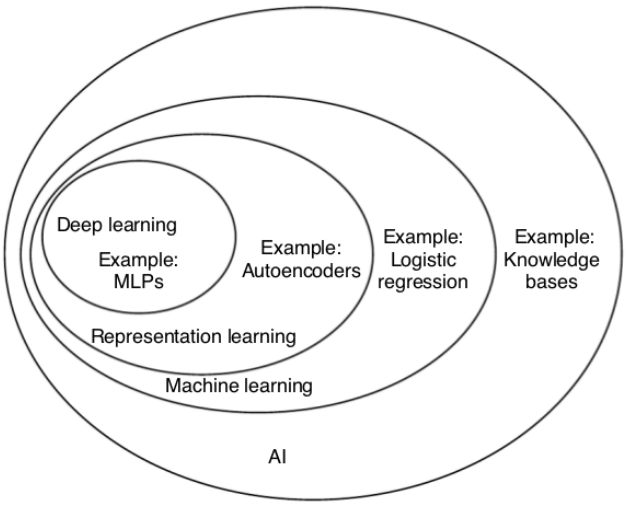
1.2 introduzione generale alle tecniche utilizzate

Per la realizzazione del progetto è stata realizzata una scheda con microcontrollore arduino, provvista di bluetooth, accelerometro, magnetometro, alimentata a batteria. É stata realizzata una libreria in C++/CUDA per la creazione, l'addestramento e l'utilizzo di una rete neuronale con supporto per CPU e GPU. É stato realizzato un secondo programma C++, per l'acquisizione dei dati della scheda e per l'acquisizione dei punti dello scheletro forniti dal kinect, lo stesso programma si occupa della sincronizzazione di tali dati e della creazione di un dataset necessario per il processo di addestramento della rete neuronale. In fine è stato realizzato un programma C# che sfrutta il game engine Unity per la realizzazione dell'ambiente grafico, che rende possibile la visualizzazione dell'output della rete, muovendo un manichino secondo gli input della scheda. Quest'ultima operazione è stata effettuata mettendo in comunicazione il programma C++ che si occuppa dell'acquisizione dei dati della scheda, e che ne esegue l'input nel modello addestrato, per poi passare i dati elaborati tramite comunicazione socket interna al motore grafico.

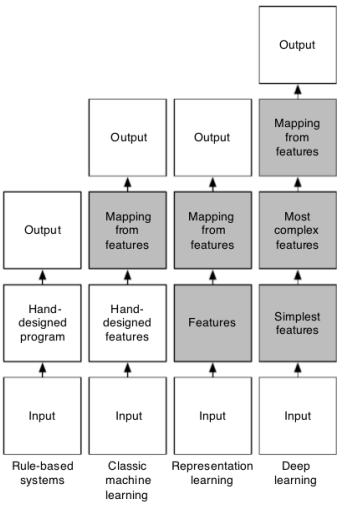
2 Richiami di teoria

2.1 Dal machine learning al deep learning

Oggi l'intelligenza artificiale è un fiorente campo di ricerca, con l'obiettivo di risolvere una grande varietà di problemi, che per essere risolti attraverso la programmazione classica necessitano di una grande quantità di conoscenze pregresse, spesso non disponibili.
I Programmi basati sul paradigma dell'intelligenza artificiale si propongono di superare questi ostacoli acquisendo direttamente queste conoscenze dai dati grezzi, tale capacità è nota come machine learning. Sotto questa famiglia di algoritmi si trovano altri sottogruppi quali il representation learning e all'interno di quest'ultimo il deep learning.
Il deep learning rispetto ai metodi più classici, tipicamente in grado di riconoscere relazioni lineari(come ad esempio il noto SVM o support vector machine), si propone come alternativa per l'apprendimento di funzioni a molte variabili non lineari anche molto complesse. Le applicazioni più comuni comprendono il riconoscimento di oggetti nelle immagini, delle parole in tracce audio o anche per le traduzioni multilingue.
Il termine "deep learning" deriva proprio dalla capacità di questi modelli di riuscire a cogliere delle relazioni molto "profonde" tra i dati di ingresso e di uscita, approssimando con un certo errore ,dipendente dal caso specifico, la funzione che dati gli input restituisce l'output desiderato, purtroppo però presentano il grande problema di non rendere disponibile in maniera chiara le relazioni apprese.



famiglie di algoritmi



differenze generali tra
le diverse tipologie di algoritmi

2.1.1 Le reti feed-forward (MLP)

Uno dei modelli della famiglia del deep learning più comuni e semplici da utilizzare sono le reti neurali feed-forward o multi layer perceptron, le quali sono assimilabili a modelli di regressione statistica.
Le reti neurali sono un modello matematico molto semplificato del cervello, e per parlarne è necessario qualche riferimento al modello biologico.

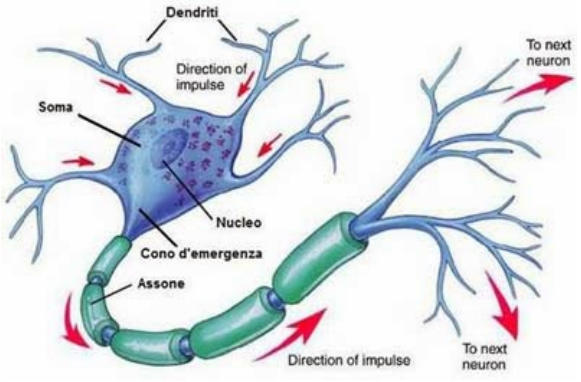


Figura 2: Il neurone biologico

Il neurone biologico

L'unità base del cervello è rappresentata dai neuroni (Figuara 2), i quali a loro volta sono composti dai dendriti, dal soma, dall'assone e dalle sinapsi.
I dendriti sono l'apparato di input del neurone, attraverso il quale riceve i segnali dall'ambiente o da altri neuroni, tali segnali caricano il neurone, che accumula un potenziale elettrico all'interno del soma. In determinate condizioni o al raggiungimento di una certa soglia di potenziale il neurone emette un impulso lungo l'assone giungendo alle sinapsi, poste alla fine dell'assone, le quali sono connesse con altri neuroni o con altri apparati del corpo, come i muscoli.
Questa particolare cellula oltre a scambiare impulsi è in grado di accumulare informazioni attraverso l'ispessimento o l'assottigliamento della guaina mielinica, la quale è presente lungo l'assone. Le variazioni dello spessore della guaina comportano un aumento o una diminuzione della resistenza elettrica dell'assone determinando una variazione dell'impulso percepito dai neuroni connessi alle sinapsi a parità di potenziale rilasciato.

Il percettrone

Il neurone artificiale è composto da una serie di connessioni in ingresso, come i dendriti nel caso del neurone biologico, questa volta però il compito di immagazzinare informazione viene svolto da queste connessioni in input e non più dalle connessioni in output. Ad ogni input del neurone è associato un coefficiente che moltiplica il potenziale in ingresso, a questo punto gli ingressi pesati giungono al nodo sommatore, il corrispettivo del soma, che ne accumula la sommatoria. Lo step successivo è il blocco contenente la funzione di trasferimento, una funzione che prende in input il potenziale del neurone e rende disponibile il risultato come output finale o come input per i neuroni successivi. La funzione di trasferimento del neurone può essere di vari tipi, può variare infatti anche tra i neuroni della medesima rete, alcune delle più comuni sono:
(a) la funzione gradino.
(b) la funzione lineare con saturazione
(c) la funzione sigmoide tra 0 e 1
(d) la funzione sigmoide tra -1 e 1

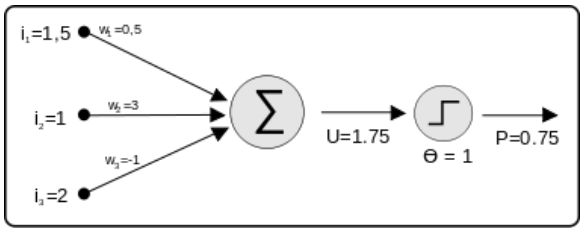


Figura 3: Il percettrone

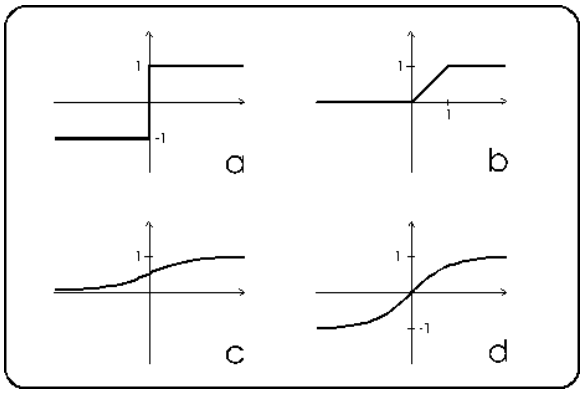


Figura 4: Le funzioni di trasferimento

Volendo esprimere il percettrone in termini matematici si ottiene la seguente equazione:

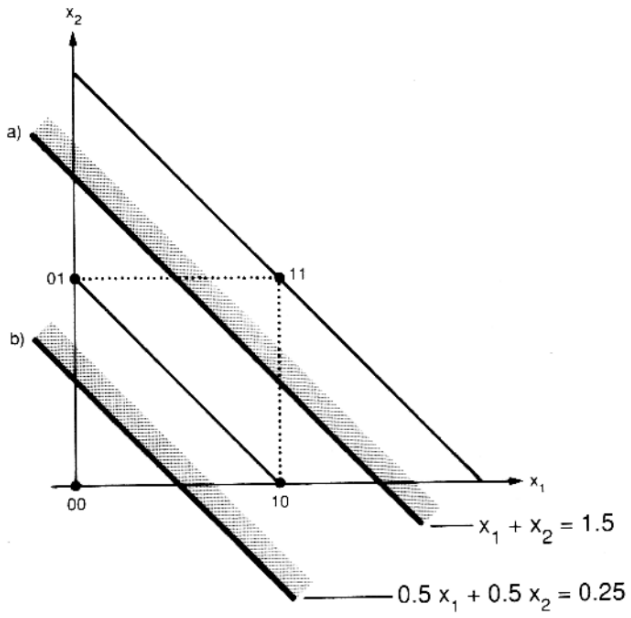
$$y = f(P) = f(\sum_i W_i \cdot X_i)$$

y - output del percettrone
f - funzione di trasferimento del percettrone
P - potenziale del neurone
W_i - peso o coefficiente dell'i-esimo ingresso
X_i - potenziale d'ingresso dell'i-esimo neurone

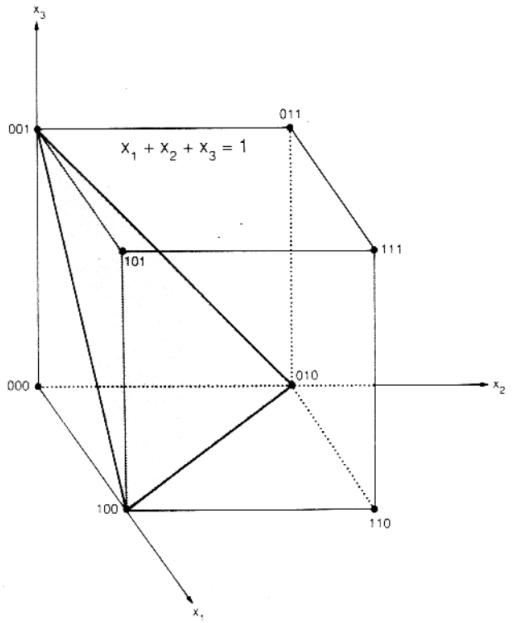
Una delle caratteristiche più importanti del percettrone è la sua capacità di compiere separazioni lineari. Tali separazioni avvengono in uno spazio a n dimensioni, dove n corrisponde al numero di variabili in ingresso al neurone, perciò si avrà ad esempio per un percettrone a due input la capacita di separare i valori su $\Re^2$ attraverso un retta, nel caso $\Re^3$ si potranno separare i valori dello spazio tridimensionale attraverso un piano, e così via.

Le reti multi-strato e la separazione non lineare

Le reti multi-strato sono composte da strati di percettroni, interconnessi tra loro in modo che gli output degli strati precedenti siano connessi con gli input dei percettroni successivi. Tale struttura "profonda" permette di rappresentare funzioni non linerai molto complesse, tanto più sono gli strati della rete.



esempio di linee di separazione per neurone a 2 ingressi binari
(a - funzione OR)
(b - funzione AND)

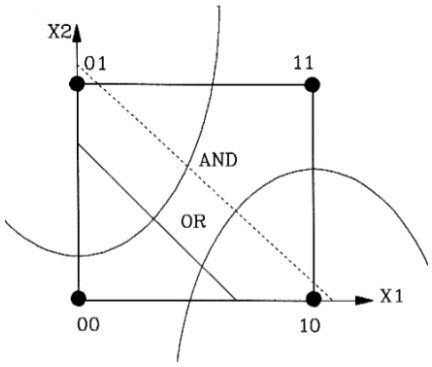


esempio di piano diseparazione lineare
a 3 ingressi per un neurone binario

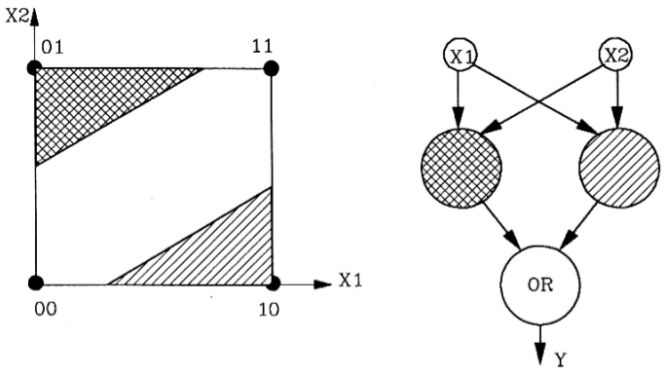
Uno degli esempi che meglio chiarisce questa problematica è la funzione xor nel seguente esempio, la quale per semplicità è rappresentata con neuroni binari a soglia.

Il problema dello XOR

Lo xor è una delle funzioni non lineari più semplice, nella quale diventa evidente l'impossibilità di approssimare tale funzione con un solo strato di neuroni. Il grafico illustra come un neurone singolo possa effettuare la separazione lineare che riproduce la funzione and e or, ma come si nota lo xor necessità di due parabole per la separazione e non più di rette, tale separazione si effettua utilizzando una rete ad almeno due strati.



Esempio di linee di separazione per and or e xor



MLP in grado di rappresentare la funzione xor

2.1.2 La funzione di addestramento: Il back-propagation

Come già detto la struttura della rete neurale deve apprendere una data funzione da dei dati.
Tale operazione si effettua presentando degli input alla rete ai quali questa risponderà con degli output, i quali dovranno essere confrontati con degli output corretti per il set di input presentato, per calcolare l'errore nell'output della rete.
L'errore che la rete commette ad ogni tentativo viene utilizzato per calcolare attraverso una prescelta funzione, nel nostro caso il BP, dei coefficienti di correzione che vanno applicati ai pesi della rete per avvicinare il suo output a quello desiderato.
Come dice il nome "retropropagazione" in italiano, l'errore calcolato dall'output viene applicato ai neuroni precedenti e poi a quelli ancora precedenti fino ad arrivare al primo strato, tale procedura puo essere effettuata in diversi modi, calcolando gli errori su un determinato numero di esempi del dataset, accumulando le correzioni ed applicandole successivamente, tale tecnica è attualmente la più utilizzata ed è detta "mini-batch", ma è possibile anche effettuare la correzione esempio per esempio, e questo è detto addestramento in-line, in fine tra i metodi più comuni c'è anche la possibilità di eseguire tutti gli esempi del dataset e solo alla fine di un'intero ciclo di addestramento applicare le correzioni. questi tre metodi ovviamente risultano validi ma a seconda del problema da risolvere possono

avere prestazioni migliori o peggiori in modo difficilmente prevedibile.
Dopo questa introduzione generale all'addestramento delle reti passiamo a dare una vista più accurata dell'algoritmo utilizzato nel progetto che è principalmente pensato per reti mlp con neuroni ad ingressi continui e con funzione di trasferimento sigmoidea o ad arcotangente.
Il BP cerca di minimizzare l'errore quadratico medio, relativo a un training set di esempi dati, scendendo lungo la superficie d'errore e seguendo la direzione di massima pendenza, in cerca di una "valle" sufficientemente profonda.
Consideriamo una rete con n input X_i (i = 1, 2,.. n), uno strato nascosto di q neuroni Z_k (k = 1, 2..., q) e uno strato output di m neuroni Y_j; (j = 1, 2... m). Il training set sia costituito da p esempi o casi C_r, (r = 1, 2,..,p). Tutti i neuroni abbiano la stessa funzione di trasferimento.
L'errore quadratico medio dello strato output è:

$$E = \frac{1}{2} \sum_j \sum_r (Y_{rj} - D_{rj})^2 \tag{1}$$

dove Y, è l'output del neurone Y_j alla presentazione dell'esempio C_r e D_{rj} è il suo valore desiderato. Per semplificare la notazione, eliminiamo l'indice r e minimizziamo l'errore quadratico medio ad ogni presentazione di esempio (modalità on-line), anziché dopo ogni ciclo completo di presentazione del training set o epoca (modalità batch). Avremo allora:

$$E = \frac{1}{2} \sum_j (Y_j - D_j)^2 \tag{2}$$

Si applica a questo punto la regola del gradiente (fig. Propagazione a):

$$\Delta W_{jk} = -\eta \frac{\partial E}{\partial W_{jk}} = -\eta \frac{\partial E}{\partial Y_j} \frac{\partial Y_j}{\partial P_j} \frac{\partial P_j}{\partial W_{jk}} = -\eta (Y_j - D_j) f'(P_j) Z_k \tag{3}$$

$$\frac{\partial E}{\partial Y_j} = (Y_j - D_j) \tag{4}$$

$$\frac{\partial Y_j}{\partial P_j} = f'(P_j) \tag{5}$$

$$\frac{\partial P_j}{\partial W_{jk}} = \frac{\partial (\sum W_{jk} Z_k)}{\partial W_{jk}} = Z_k \tag{6}$$

Ponendo a questo punto:

$$\delta_j = (Y_j - D_j) f'(P_j) \tag{7}$$

$$\Delta W_{jk} = -\eta \, \delta_j \, Z_k \tag{8}$$

La formula (3) viene impiegata per aggiornare i pesi sinattici delle connessioni tra strato nascosto e strato output, analogamente, per quanto riguarda le connessioni tra strato input e strato nascosto si utilizza la formula (fig. Propagazione b):

$$\Delta W_{ki} = -\eta \frac{\partial E}{\partial W_{ki}} = -\eta \frac{\partial E}{\partial Z_k} \frac{\partial Z_k}{\partial P_k} \frac{\partial P_k}{\partial W_{ki}} = -\eta \frac{\partial E}{\partial Z_k} f'(P_k) X_i \tag{9}$$

Confrontando questa formula con la precedente formula (3), al posto degli errori noti $(Y_j - D_j)$ troviamo le derivate $\frac{\partial E}{\partial Z_k}$, che dobbiamo ora calcolare retropropagando l'errore dallo strato output a ogni neurone nascosto:

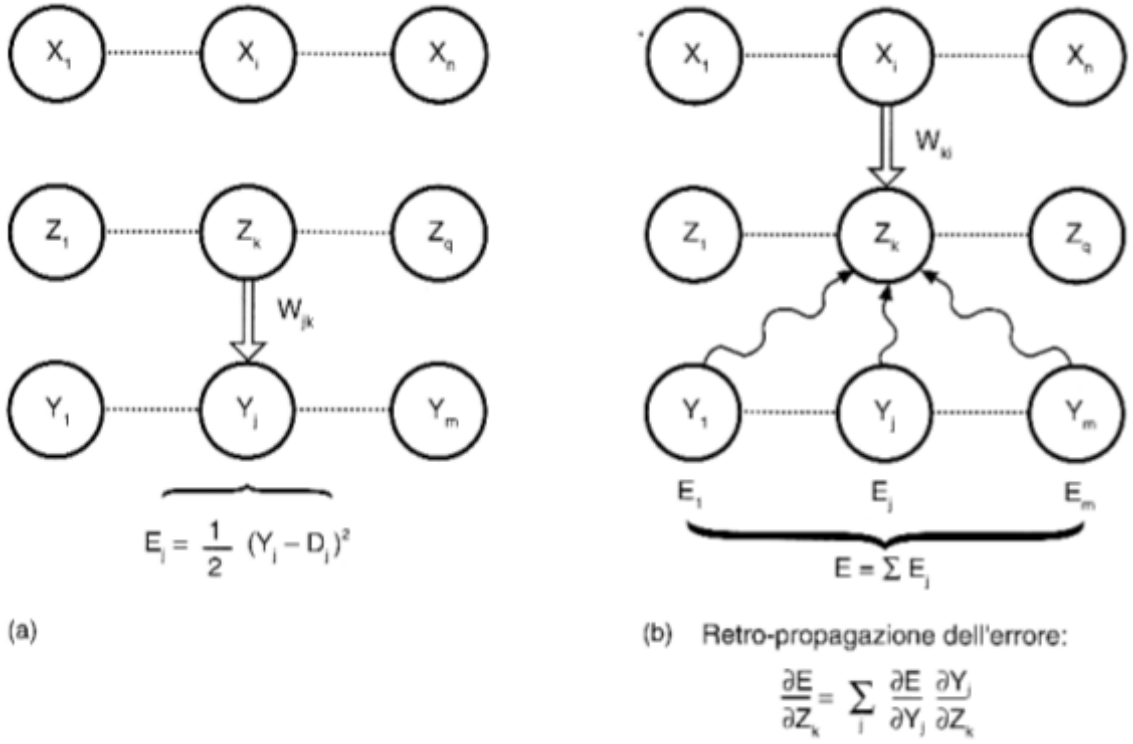
$$\frac{\partial E}{\partial Z_k} = \sum_j \left(\frac{\partial E}{\partial Y_j} \frac{\partial Y_j}{\partial P_j} \frac{\partial P_j}{\partial Z_k} \right) = \sum_j (Y_j - D_j) f'(P_j) W_{jk} \tag{10}$$

o anche, applicando la formula (7) :

$$\frac{\partial E}{\partial Z_k} = \sum_j \delta_j \, W_{jk} \tag{11}$$

In fine otteniamo le equazioni:

$$\Delta W_{ki} = -\eta f'(P_k) (\sum_j \delta_j W_{jk}) X_i \tag{12}$$



Propagazione dell'errore nella rete

che ponendo:

$$\delta_k = f'(P_k)(\sum_j \delta_j W_{jk}) \tag{13}$$

la quale assume la stessa forma della (8) :

$$\Delta W_{ki} = -\eta\, \delta_k\, X_i \tag{14}$$

Le formule (12) (13) (14) sono valide non solo per aggiornare i pesi sinattici tra strato nascosto (l'unico nella rete che abbiamo ipotizzato) e strato input ma anche, in reti più generali, per le connessioni tra due strati nascosti consecutivi. Se si adotta la funzione di trasferimento sigmoide si ha:

$$Y = f(P) = \frac{1}{1 + e^{kP}} \tag{15}$$

$$f'(P) = kf(P)(1 - f(P)) = kY(1 - Y) \tag{16}$$

Dove k è proporzionale alla pendenza della sigmoide nel suo punto di flessione, normalmente si pone k = 1. A questo punto otteniamo la rielaborazione delle formule (8) (12):

$$\Delta W_{jk} = -\eta k(Y_J - D_j)Y_j(1 - Y_j)Z_k \tag{17}$$

$$\Delta W_{ki} = -\eta k Z_k(1 - Z_k)(\sum_j \delta_j W_{jk})X_i \tag{18}$$

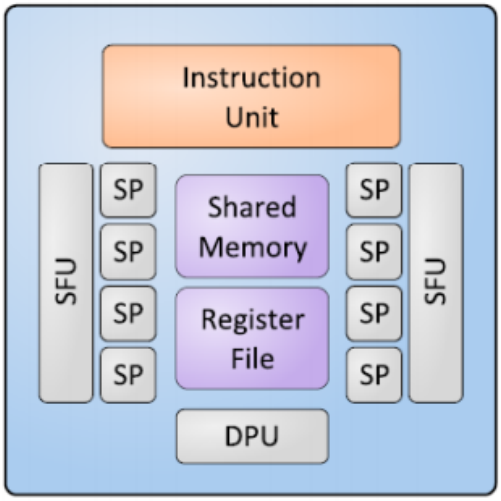
infine i pesi sinattici vengono aggiornati, ad ogni tempo t dell'intervallo 1, 2, ..., (t-1), t, (t+1), ...ecc con:

$$W_{jk}(t + 1) = W_{jk}(t) + \Delta W_{jk} \tag{19}$$

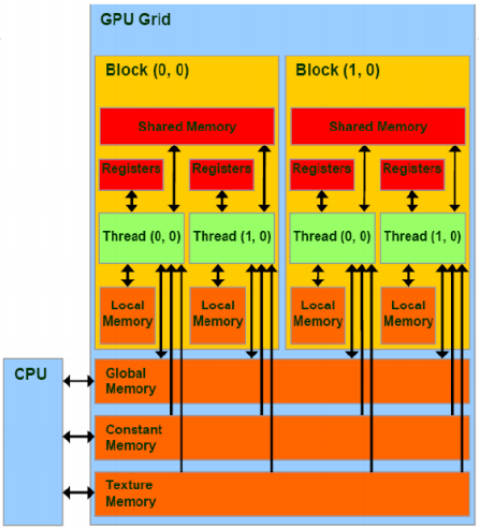
$$W_{ki}(t + 1) = W_{ki}(t) + \Delta W_{ki} \tag{20}$$

2.2 Il computo parallelo

L'incremento di prestazioni dei calcolatori fa sempre più affidamento all'ausilio delle GPU, le quali vengono utilizzate sempre di più per operazioni non prettamente grafiche, le quali presentano un' architettura che punta alla massimizzazione del numero di unità di elaborazione e non più all'incremento della frequenza di lavoro.
Tale approccio non è sempre il migliore, infatti la loro efficacia è strettamente legata al tipo di applicazioni, in particolare alla parallelizzabilità del codice, ovvero alla dipendenza di determinate istruzioni di avere a disposizione i dati elaborati da istruzioni precedenti.
Questa architettura trova la massima efficacia quando un grande numero di istruzioni è completamente indipendente dalle altre.



Struttura di uno stream-multiprocessor



Struttura logica della GPU

2.2.1 L'architettura della GPU

La struttura della GPU è definita "SIMT" (single instruction, multiple thread), tale aggettivo è dato in quanto la GPU è realizzata per eseguire un medesimo programma su un gran numero di processori contemporaneamente, i quali però ricevono dati diversi da computare. Tipicamente i processori presenti su una GPU sono più piccoli e semplici di quelli di una CPU e anche molto meno performanti ma il loro grande numero e la strutturazione delle memorie presenti al suo interno permette per il codice parallelizzabile di ottenere tempi di esecuzione estremamente minori rispetto a quelli in una CPU.
Uno dei componenti più importanti in una GPU è lo stream-multiprocessor, Il quale può essere considerato un processore vettoriale, una GPU solitamente ne contiene diversi, e compongono il modello fisico della GPU, i quali a loro volta sono composti principalmente da tre unità di calcolo:

- **SP (scalar processor)** - E' un'unita logica che esegue operazioni in virgola fissa e mobile di un solo thread per volta, è inportante inoltre sapere che gli SP di uno stesso SM hanno una memoria detta shared condivisa che gli permette di scambiarsi dati con estrema velocità a tempo di esecuzione.
- **SFU (special functions unit)** - E' un'unità logica di supporto agli SP che esegue a livello HW funzioni complesse come le funzioni gognometriche e trascendenti che appesantirebbero eccessivamente gli SP.
- **DPU (double proces unit)** - Quest'ultima si occupa delle operazioni su numeri a 64 bit, i double.

Una delle peculiarità più importanti della GPU è la differenza tra il modello fisico e quello logico, infatti il modello al quale il programmatore deve far riferimento per ottimizzare l'esecuzione del codice è una via di mezzo tra quello logico e quello che la macchina implementa fisicamente, cercando il bilanciamento migliore possibile tra comprensibilità dei dati e ottimizzazione del codice. Il modello logico. Il programmatore infatti ha la possibilità di interagire e modellare questa struttura composta da oggetti logici detti griglie (GRID), tali griglie possono avere tre dimensioni e a loro a volta sono composte da altri oggetti logici detti blocchi, i quali possono avere fino a tre dimensioni, che in fine sono composti da thread. Ogni SM generalmente processa un blocco alla volta, mettendo a disposizione la sua shared memory tra i thread dello stesso blocco, tra i vari blocchi e griglie è possibile lo scambio di informazioni esclusivamente attraverso la global memory, molto più grande delle cache degli SM ma anche molto più lenta. Un'ultimo argomento da trattare riguardo il legame tra la struttura logica e quella hardware della GPU è "il flusso di esecuzione" ovvero il metodo con cui lo scheduler, ovvero l'organo interno che si occupa della ripartizione delle operazioni ,strutturate dal programmatore nella griglia, ai vari SM. Tale operazione ha come unità base il warp, ovvero un numero n dipendente dalla specifica architettura, nel nostro caso 32, di thread appartenenti allo stesso blocco che vengono eseguiti contemporaneamente in maniera "lookstepped" (ovvero in sincronizzazione forzata). L'esecuzione del warp si conclude e si passa al successivo solo nel momento in cui tutti i suoi thread arrivano a fine esecuzione indipendentemente dalla "branch" di programma che hanno seguito.

2.2.2 L'utilizzo della GPU per il deep-learning

Nel deep learning come illustrato nella sezione precedente, si effettuano moltissimi calcoli, i quali sono per gran parte scorrelati tra loro, caratteristica questa fondamentale per la parallelizzazione delle operazioni. L'utilizzo di calcolatori paralleli per l'addestramento delle reti neuronali consente di completare addestramenti per modelli molto grandi in tempi molto ridotti. Confrontando l'addestramento di una rete su una CPU desktop rispetto ad una GPU da gaming si possono incrementare le prestazioni da 200 a 500 volte, ciò significa che un addestramento di un'ora su GPU, può richiedere su una CPU un tempo che può andare da una a tre settimane.

(Nel nostro caso da un i5-4670k abbiamo avuto un incremento di prestazioni di 250 volte su una GPU Nvidia gtx760 asus, con un programma che pecca fortemente di ottimizzazione).

2.3 Il kinect

Il kinect è un dispositivo per la rilevazione del movimento, prodotto dall'azienda Microsoft e commercializzato insieme alla console videoludica Xbox. Esso può tuttavia essere interfacciato ad un comune pc tramite l'apposito connettore. Nel kinect è presente una fotocamera 1080p RGB, una fotocamera infrarossi per calcolare la profondità e 4 microfoni. Con questo set di sensori ed il suo SDK (software development kit), consente l'utilizzo dei dati prodotti per scopi più generici, una tra le quali l'utilizzo delle gesture per controllare il pc o la ricostruzione delle espressioni facciali. Questi dati sono resi disponibili attraverso l'apposito SDK e possono essere usati in generici programmi C++, C# ed altri linguaggi di programmazione, nello specifico per il linguaggio C++ si usa la libreria "Kinect.h".

Figura 8: Il kinect.

2.3.1 Modello dello scheletro

In particolare ci siamo interessati alla capacità del kinect di poter ricostruire la posizione delle parti del corpo nello spazio, infatti utilizzando la fotocamera infrarossi è in grado di rilevare 24 punti diversi nel corpo come indicato in figura 9.

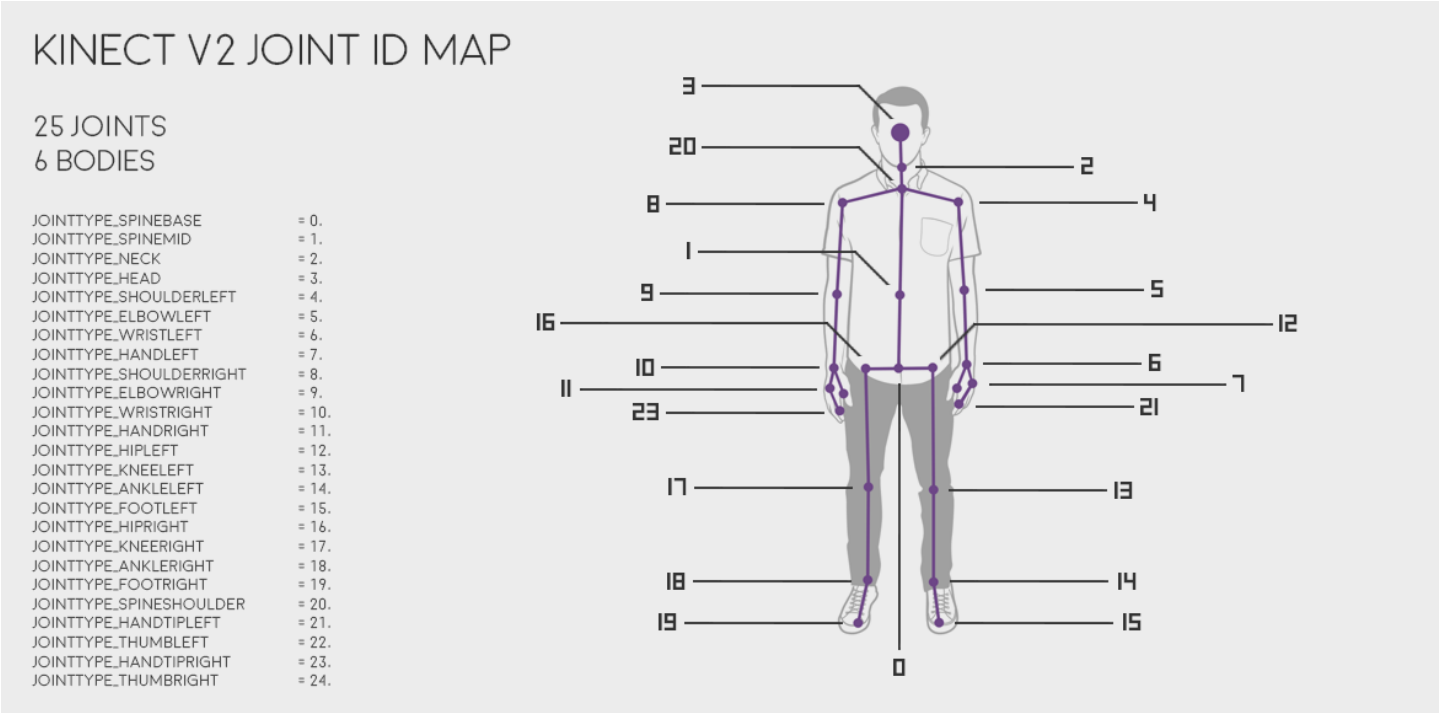


Figura 9: La mappa dei giunti che il kinect è in grado di rilevare.

I dati di posizione e profondità restituiti dal kinect fanno riferimento alla terna di orientazione indicata in figura 10. Per in nostri scopi abbiamo utilizzato delle procedure già definite dalla libreria che permettono di acquisire la posizione nello spazio di ogni singolo giunto dello scheletro e il suo stato di tracciamento, se la sua posizione è stata interpolata o è stato effettivamente tracciato.

Figura 10: Orientazione degli assi rispetto al kinect.

2.4 Scheda enbedded

Per questo progetto abbiamo realizzato una scheda enbedded che comprendesse l'imu, un modulo bluetooth, la batteria con il suo regolatore e un arduino.

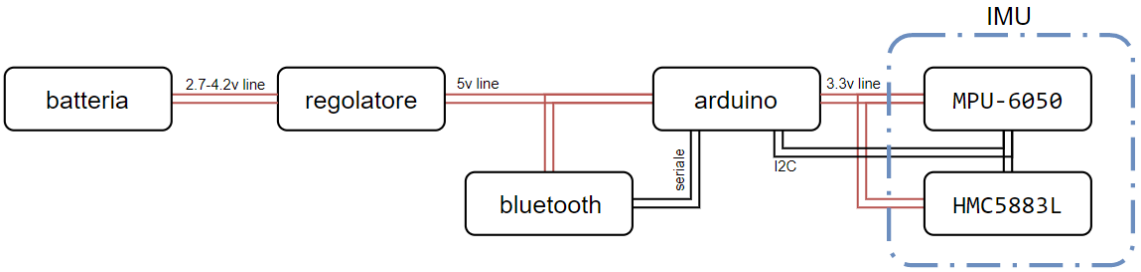


Figura 11: Schema scheda enbedded

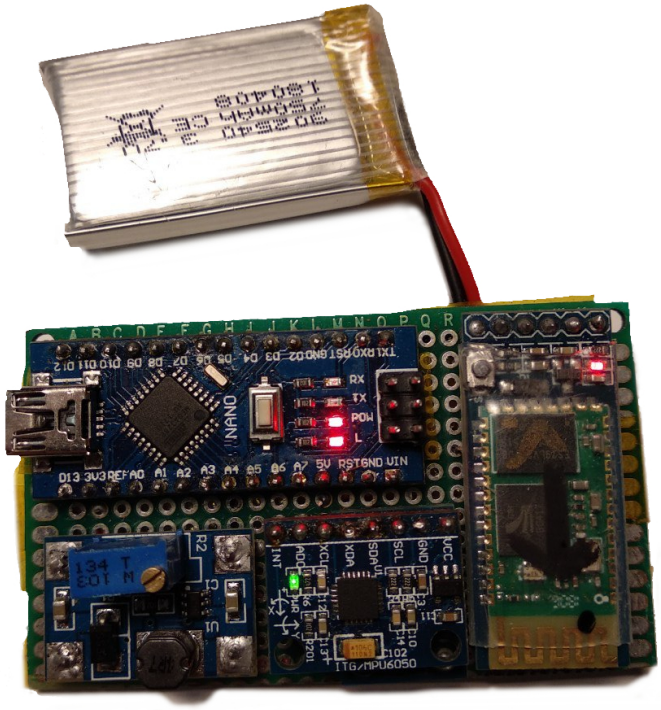


Figura 12: Scheda enbedded

La scheda è stata realizzata su millefori saldando e wrappando i diversi componenti.

2.4.1 Arduino

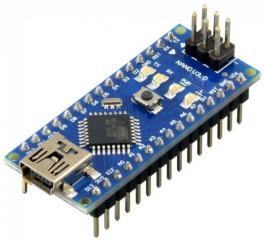


Figura 13: Arduino nano

Arduino è una piattaforma hardware open-source dotata di microcontrollore e tutto il suo ecosistema, questo lo rendono estremamente utile per realizzare progetti che non richiedono specifiche particolari senza doveresi realizzare una scheda apposita. Le schede arduino possono essere programmate attraveso l'ide proprietario "Arduino IDE", che oltre ad includere la compatibilità con tutte le schede arduino offre alcuni strumenti utili come il motitor seriale.
Nel nostro progetto abbiamo usato la scheda "Arduino nano" (fig.13), si alimenta a 5 V e include un regolatore di tensione da 5 V a 3.3 V, una seriale, l'I2C ed altri pin general purpose.
Il regolatore interno è stato usato per alimentare i moduli HMC5883L e MPU-6050 che costituiscono l'imu della scheda, la seriale è stata utilizzata per comunicare con il modulo bluetooth HC-05, il bus di comunicazione I2C è stato usato per interfacciarsi con l'imu ed un pin GPIO è stato usato per controllare lo stato del modulo bluetooth.

2.4.2 Bluetooth

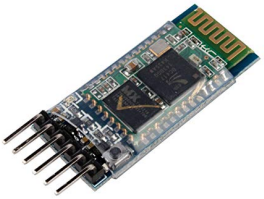


Figura 14: HC-05

Il modulo bluetooth ulilizzato è "HC-05", ha 6 pin per interfacciarsi con le atre periferiche: 2 di alimentazione, 2 per la seriale, state e key.
Questo modulo ha un regolatore da 5 V a 3.3 V integrato, quindi va alimentato a 5 V.
Per comunicare con arduino usa la seriale (RS-232) con livello logico 3.3 V, ciò comporta la necessità di inserire un partitore sul pin rx del modulo (quindi il pin tx di arduino).
Il pin state èstato usato per far conoscere ad arduino quando il modulo ha stabilito la connessione con il

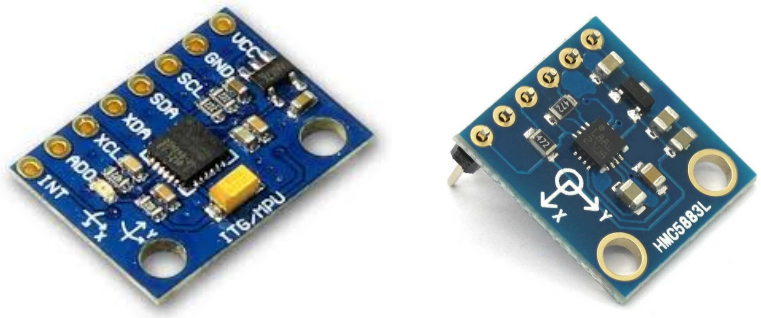
computer.
Il pin key svolge una funzione particolare poichè permette di avviare il modulo in modalità "command mode", in questa modalità si possono usare gli AT commands per programmare il modulo (es. cambiare nome al dispositivo, oppure cambiare il baud rate della seriale).
I comandi AT e la loro descrizione si trovano sul datasheet del modulo.
Per avviare l'HC-05 in modalità AT si deve collegare il modulo all'alimentazione tenendo premuto il pulsante che si trova sulla scheda, dopodiche si può procedere ad inviare gli at command tramite seriale con un baud rate di 38400Bd. Per fare ciò abbiamo collegato la seriale del modulo con 2 pin di arduino con supporto PWM ed abbiamo usato la libreria software serial per poter avere un'altra seriale (la principale connessa al pc e la secondaria connessa al modulo).
Dopodichè basta programmare arduino in modo che reindirizzi ciò che gli viene scritto dal pc al modulo, e viceversa. il programma usato è il seguente:

```
#include <SoftwareSerial.h>

SoftwareSerial BTSerial(10, 11); // RX | TX

void setup()
{
  Serial.begin(9600);
  Serial.println("Enter AT commands:");
  BTSerial.begin(38400);  // HC-05 default speed in AT command more
}

void loop()
{
  // Keep reading from HC-05 and send to Arduino Serial Monitor
  if (BTSerial.available())
    Serial.write(BTSerial.read());

  // Keep reading from Arduino Serial Monitor and send to HC-05
  if (Serial.available())
    BTSerial.write(Serial.read());
}
```

Così facendo abbiamo impostato il baud rate a 115 200 Hz.

2.4.3 L'imu

L'imu (inertial measurement unit) permette di misurare le forze ad esso applicate e l'orientazione dello stesso. Questo viene solitamente fatto combinando i dati di accelerometro, magnetometro e giroscopio. In particolare l'accelerometro misura le accelerazioni, da cui in condizioni di moto inerziale si può estrarre il vettore gravità sui 3 assi determinando quindi l'angolazione rispetto al suolo; Il magnetometro rileva invece il campo magnetico terrestre su 3 assi, dando così indicazione della direzione "nord"; Infine il giroscopio restituisce le accelerazioni angolari. Per questo specifico progetto si sono utilizzati i moduli commerciali "MPU-6050" (fig.15a) e "HMC5883L" (fig.15b), rispettivamente come accelerometro più giroscopio e magnetometro.



(a) MPU-6050 (b) HMC5883L

Figura 15: l'IMU utilizzata in questo progetto

Questi dispositivi comunicano con arduino attraverso il protocollo I2C, sono stati quindi connessi ai pin analogici 5 e 4 di arduino. In particolare l'MPU-6050 integra un accelerometro su 3 assi ed un giroscopio su 3 assi, che vengono convertiti in digitale da 6 ADC a 16 bits. Inoltre può essere programmato su diverse precisioni, il giroscopio tra ±250 °/s e ±2000 °/s, l'accelerometro tra ±2 g e ±16 g. Questo modulo ha anche un sensore di temperatura che per questo progetto non è stato usato. Supporta l'I2C fino a 400 kHz.

2.4.4 Regolatore di tensione

Questo regolatore di tensione è del tipo switching e permette di elevare la tensione da 2 V-24 V a 2 V-28 V con un picco massimo di 2 A. La tensione di uscita viene impostata girando il trimmer e qualunque sia la tensione in ingresso l'uscita rimarrà al valore impostato purchè si rispettino i limiti massimi e la tensione di ingresso sia sempre minore di quella di uscita. Nel nostro caso è stato molto utile perchè la betteria a litio ha una tensione che oscilla tra 2.7 V quando è scarica e 4.2 V quando è carica e questo integrato provvede a stabilizzare l'alimentazione a 5 V.

Figura 16: SX1308

3 Programmi

3.1 Overview generale e main server

Introduciamo i programmi con uno schema generale della relazione logico temporale tra i programmi: per prima cosa abbiamo dovuto acquisire un dataset ed usarlo per addesrare la rete neurale

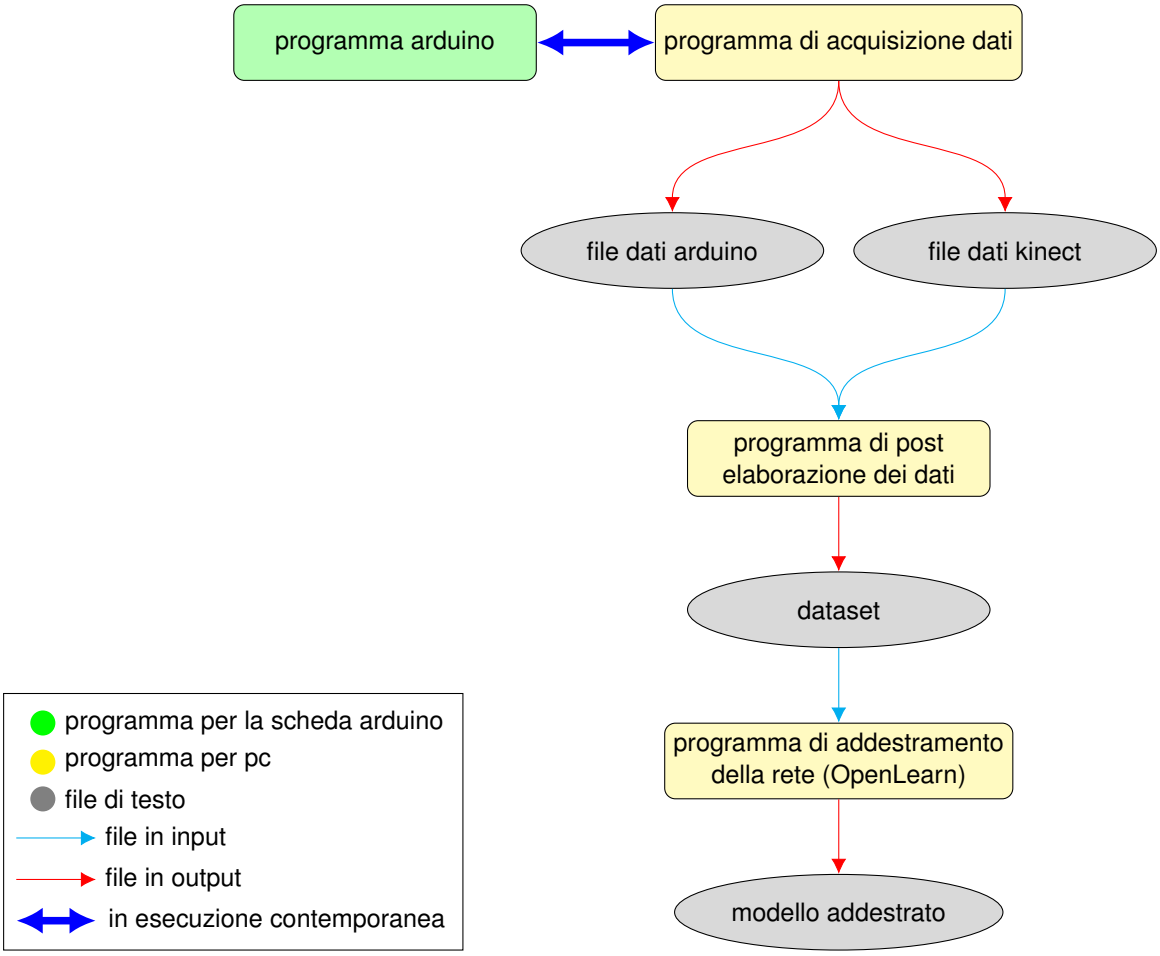


Figura 17: i programmi usati per ottenere i pesi della rete addestrata (?)

una volta addestrata la rete si possono usare i pesi così ottenuti per visualizzare in tempo reale il pupo che si muove(???)

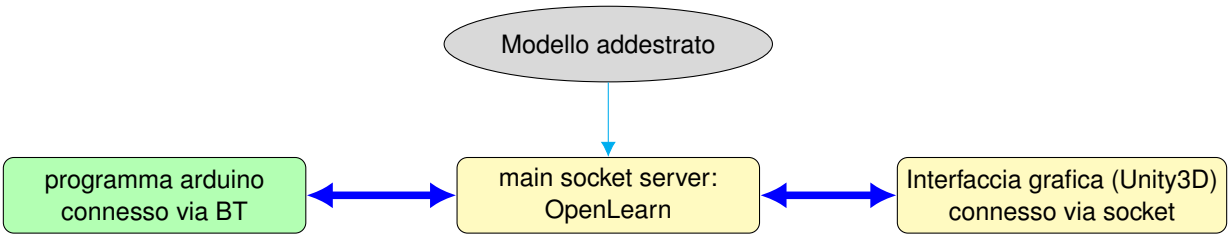


Figura 18: una volta addestrata la rete il manichino viene controllato dal suo output in real-time, controllato dai dati ricevuti via BT da arduino. Nella seguente immagine il data-flow va da sinistra verso destra.

Il programma principale è stato realizzato in C++, con l'ausilio della libreria boost per l'implementazione dei thread e del server socket, in oltre integra la libreria OpenLearn per il caricamento del modello preaddestrato e il processamento dei dati real-time, in tale programma è presente un thread che si occuppa parallelamente della ricezzione dei dati da arduino, tali dati vengono caricati nel circular buffer il quale è eseguito in un altro thread. I dati contenuti nel circular buffer vengono successivamente presi dal thread che esegue il controllo della rete, la quale elabora i dati che vengono poi immagazzinati e inviati al client unity su richiesta.

3.2 Programma arduino

Questo programma si occupa di acquisire i dati dai sensori attraverso l'I2C, elaborarli e scriverli sulla seriale, in modo che il modulo bluetooth li invii al pc. Il programma si scompone nella classe "MPU0_6050" che gestisce l'accelerometro, nella classe "QMC5883" che gestisce il magnetometro, nella classe "serial" per gestire la seriale e nelle 2 funzioni standard di arduino "setup" e "loop". Inoltre abbiamo usato la libreria "Wire.h" per l'I2C.
Nel "setup" si trova soltanto l'inilizializzazione alla seriale per comunicare con il modulo "HC-05", il baud rate impostato è lo stesso che è stato settato nel modulo tramite la modalità AT:

```
1 void setup()
2 {
3   Serial.begin(115200);
4 }
```

Listing 1: funzione "setup"

Nella funzione "loop" si trovano le varie dichiarazioni alle altre classi, un while per aspettare che il modulo "HC-05" sia connesso al pc ed il ciclo infinito che provvede ad acquisire i dati dai sensori e scriverli sulla seriale:

```
1  void loop()
2  {
3    // chiamo i costruttori delle 2 classi che gestiscono i sensori e provvedo al loro settaggio.
4    QMC5883 mm;
5    MPU_6050 aa;
6    aa.setting();
7
8    // accendo un led per il debug
9    digitalWrite(13, HIGH);
10
11   // aspetto finchè il pin "state" del modulo "HC-05" non diventa alto
12   while (!digitalRead(3))
13   {
14     delay(10);
15   }
16
17   // quando il bluetooth è connesso posso procedere ad inizializzare e sincronizzare la seriale
18   serial ser;
19   ser.sinc();
20
21   // spengo il led
22   digitalWrite(13, LOW);
23
24   while (true)
25   {
26     // acquisisco i dati del magnetometro e li normalizzo
27     mm.get_data();
28     mm.normalize(100);
29
30     // acquisisco i dati dell'accelerometro
31     aa.get_data();
32
33     // li sposto in degli array
34     float acc_xyz[] = {aa.get_AcX(), aa.get_AcY(), aa.get_AcZ()};
35     float g_xyz[] = {aa.get_GyX(), aa.get_GyY(), aa.get_GyZ()};
36     float magn[] = {mm.getX(), mm.getY(), mm.getZ()};
37
38     // li scrivo sulla seriale in modo che il modulo bluetooth li trasmetta al pc.
39     ser.send_data(acc_xyz, g_xyz, magn, aa.get_temp());
40   }
41 }
```

Listing 2: funzione "loop"

Qui sono state usate le classi "MPU_6050", "QMC5883" e "serial".
Le classi "MPU_6050" e "QMC5883" si occupano di impostare i registri dei 2 moduli ed inoltre hanno metodi che permettono l'estrazione delle misurazioni da loro effettuate.
L'impostazione dei moduli è stata eseguita facendo riferimento ai rispettivi datasheet.

```
1 class MPU_6050
2 {
```

```
  public:
    int16_t AcX,AcY,AcZ,Tmp,GyX,GyY,GyZ;
    float normalized_AcX, normalized_AcY, normalized_AcZ, normalized_GyX, normalized_GyY,
        ↪ normalized_GyZ;
    float rescaled_AcX, rescaled_AcY, rescaled_AcZ, rescaled_GyX, rescaled_GyY, rescaled_GyZ;

    const uint8_t MPU = 0x68; // I2C address of the MPU-6050

    // costruttore della classe, accende l'MPU_6050
    MPU_6050(){...}

    // prede i dati dall'MPU_6050 e le sposta sulle variabili di classe.
    void get_data(){...}

    // funzioni per ritornare i valori dell'accelerometro
    float get_AcX(){ return (AcX*1.0); }
    float get_AcY(){...}
    float get_AcZ(){...}

    // funzioni per ritornare i valori del giroscopio
    float get_GyX(){ return (GyX*1.0); }
    float get_GyY(){...}
    float get_GyZ(){...}

    // funzioni per ritornare i valori normalizzati dell'accelerometro
    float get_normalized_AcX(){ return normalized_AcX; }
    float get_normalized_AcY(){...}
    float get_normalized_AcZ(){...}

    // funzioni per ritornar i valori normalizzati del giroscopio
    float get_normalized_GyX(){ return normalized_GyX; }
    float get_normalized_GyY(){...}
    float get_normalized_GyZ(){...}

    // funzioni per calcolare e ritornare la norma dell'accelerometro o del magnetometro
    float norma(float x, float y, float z){...}
    float get_norma_Gy(){ return norma(GyX, GyY, GyZ); }
    float get_norma_Ac(){...}

    // funzione per ritornare la temperatura
    float get_temp(){ return Tmp; }

    // funzioni per normalizzare l'accelerometro o il magnetometro
    void normalize_Ac(int Max){...}
    void normalize_Gy(int Max){...}

    // funzione che scrive le impostazioni all'MPU_6050
    void setting(){...}
};
```

Listing 3: classe "MPU_6050"

La seguente è la classe di gestione della bussola 3D

```
class QMC5883
{
  public:
    uint8_t add = 0x0D;
    int nowX = 0;
    int nowY = 0;
    int nowZ = 0;
    float rescaled_x = 0;
    float rescaled_y = 0;
    float rescaled_z = 0;

    // il costruttore provvede ad inizializzare il modulo e settarne i vari registri
    QMC5883(){...}

    // funzioni per ritornare i valori x-y-z del magnetometro
    float getX(){ return (nowX*1.0); }
```

```
    float getY(){...}
    float getZ(){...}

    // funzioni per ritornare i valori del magnetometro dopo averli riscalati
    float getX_rescaled(){ return rescaled_x; }
    float getY_rescaled(){...}
    float getZ_rescaled(){...}

    // funzione che provvede a leggere i regisrti del modulo per acquisire i dati
    // spostandoli nelle variabili di classe
    void get_data(){...}

    // funzioni per calcolare la norma e ritornare i valori normalizzati
    float norma(float x, float y, float z){...}
    float get_norma(){ return norma(nowX, nowY, nowZ); }
    void normalize(int Max){...}
};
```

Listing 4: classe "QMC5883"

Infine la classe serial si occupa di scomporre i dati di tipo float restituiti dai moduli in array di 4 byte che poi vengono trasmessi al modulo bluetooth tramite la seriale.

```
class serial
{
  public:

    // funzione che permette di sincronizzarsi con il pc
    void sinc(){...}

    // union per poter scompattare un float in 4 byte
    union Scomp_float{...};

    // funzione che permette di inviare un intero float scompattandolo in
    // 4 byte che vengono trasmessi serialmente
    void send_float(float n){...}

    // funzione che permette di riceve un float come sopra
    float receive_float(){...}

    // funzione per l'invio di un solo carattere
    void send_char(char ch){...}

    // funzione per la ricezione di un singolo carattere
    char receive_char(){...}

    // funzione che provvede ad inviare i dati necessari.
    void send_data(float* acc_xyz, float* g_xyz, float* magn, float temp){...}
};
```

Listing 5: classe "serial"

Per effettuare l'operazione di serializziopne dei float in array di byte abbiamo fatto ricorso all'uso di una struttura dati chiamata "union":

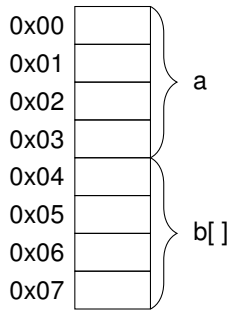
```
union Scomp_float
{
  float f;
  int i;
  unsigned char byte_s[4];
};
```

Listing 6: classe "serial"

Questa struttura dati ha la particolarità che a differenza di una struct, in cui i dati sono memorizzati in modo contiguo nella memoria, l'union memorizza tutti i dati della struttura nelle stesse locazioni di memoria.

```c
struct es_struct
{
  int a;
  unsigned char b[4];
}
```

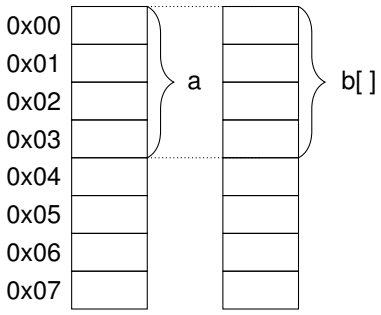
(a) Struct generico



(b) Rappresentazione dei dati dello struct memorizzati in ram

```c
union es_union
{
  int a;
  unsigned char b[4];
}
```

(c) Union generico



(d) Rappresentazione dei dati dell'union memorizzati in ram

Figura 19: Confronto tra struct ed union

Come si può vedere dalla figura 19 lo struct (figura 19a) memorizza i dati in ram facendo partire l'intero dall'indirizzo 0x00 e l'array di byte dall'indirizzo 0x04 (figura 19c), mentre l'union (figura 19d) sovrappone i 2 dati nelle stesse locazioni di memoria (figura 19). Cioò permmette ad esempio di leggere un intero (o un qualunque altro tipo di dato) come un insieme di byte: con riferimento alla figura 19d se pongo:

```
es_union U;
U.a = 0x0514FFAA;
```

Listing 7: classe "serial"

e successivamente vado a vedere il valore di b[ ] ottengo i valori:

```
U.b[0] -> 0x05
U.b[1] -> 0x14
U.b[2] -> 0xFF
U.b[3] -> 0xAA
```

Listing 8: classe "serial"

che sono esattamente i byte costituenti U.a. Ciò ci ha permesso di inviare un float come sequenza di byte al pc, nella classe serial sul pc c'e' esattamente lo stesso tipo di struttura per ricostruire il float.

3.3 programma per l'acquisizione dei dati dal kinect e da arduino

Questo programma è stato usato per acquisire i dati da arduino e dal kinect e scriverli in dei file di testo, ce vengono successivamente elaborati.

3.3.1 librerie usate

La lista completa delle librerie usate è:

```c
// libreria usata da visual studio, da togliere in caso si usi un altro ide
#include "pch.h"

// librerie standard per i file stringhe e altro
#include <cstddef>
#include <cstdlib>
#include <fstream>
#include <iomanip>
#include <iostream>
#include <ostream>
#include <string>
#include <math.h>

// librerie create da noi
#include "real_time.h"
```

```
#include "kin_file_manager.h"
#include "ard_file_manager.h"
#include "data_structure.h"

// libreria per usare i thread
#include <boost/thread.hpp>

// altre librerie per log, alcuni tipi di dato, un buffer FIFO
#include <boost/log/core.hpp>
#include <boost/call_traits.hpp>
#include <boost/circular_buffer.hpp>
#include <boost/container/vector.hpp>

// altre librerie per il logger
#include <boost/log/attributes.hpp>
#include <boost/log/attributes/scoped_attribute.hpp>
#include <boost/log/expressions.hpp>
#include <boost/log/sinks/sync_frontend.hpp>
#include <boost/log/sinks/text_ostream_backend.hpp>
#include <boost/log/sources/basic_logger.hpp>
#include <boost/log/sources/record_ostream.hpp>
#include <boost/log/sources/severity_logger.hpp>
#include <boost/log/utility/setup/common_attributes.hpp>
#include <boost/log/utility/setup/console.hpp>
#include <boost/log/utility/setup/file.hpp>

#include <boost/smart_ptr/make_shared_object.hpp>
#include <boost/smart_ptr/shared_ptr.hpp>

// altre librerie per i thread
#include <boost/thread/condition_variable.hpp>
#include <boost/thread/mutex.hpp>
#include <boost/thread/thread.hpp>

// librereria per il tempo
#include <boost/date_time/posix_time/posix_time.hpp>

// libreria fatta da noi per gestire la seriale
#include "Serial_handler.h"

// librerie di Windows e kinect
#include <Windows.h>
#include <Kinect.h>
#include <Shlobj.h>

// namespace standard
using namespace std;
```

Listing 9: librerie usate

3.3.2 Librerie custom

Per alleggerire il file "main" abbiamo spostato delle classi in file esterni, che sono a tutti gli effetti delle librerie a se stanti. Abbiamo costruito librerie per: la gestione del tempo, i file manager, le strutture dati e la gestione della seriale.

3.3.2.1 Libreria per misurare il tempo

In "real_time.h" ci sono alcune funzioni per prendere il tempo e misurarlo:

```
#pragma once

#ifndef _REAL_TIME_H_
#define _REAL_TIME_H_

#include <boost/chrono.hpp>
#include <boost/timer/timer.hpp>

class real_time
{
```

```
private:
  boost::timer::cpu_timer timer;
  boost::timer::cpu_times t;

public:
  /// <summary>
  /// start the time counting
  /// </summary>
  /// <returns></returns>
  void start();

  /// <summary>
  /// return enlapsed time in millisecond
  /// </summary>
  /// <returns></returns>
  float stop();

  /// <summary>
  /// get the time in millisecond
  /// </summary>
  /// <returns></returns>
  uint64_t get_curr_time();

};

#endif // #ifndef _REAL_TIME_H_
```

Listing 10: librerie usate

3.3.2.2 File manager per il kinect

In "kin_file_manager.h" ci sono le funzioni per gestire il file di testo relativo al kinect:

```
#pragma once

#ifndef _KIN_FILE_MANAGER_H_
#define _KIN_FILE_MANAGER_H_

#include <iostream>
#include <ostream>
#include <fstream>
#include <string>

// in questa libreria sono dichiarate le varie strutture dati
#include "data_structure.h"

class kin_file_manager
{
private:
  std::fstream f;
  std::ios::_Openmode mode;
  unsigned long int line_id = 0;

public:
  // il costruttore provvede a settare il nome del file e la modalit\'a di apertura
  kin_file_manager(std::string file_name, std::ios::_Openmode mode);

  // questa funzione provvede a scrivere un singolo blocco di dati nel file
  void write_data_line(kinect_data dat);

  // funzione che legge un blocco di dati dal file e lo sposta nella struttura dati
  // se si \'e raggiunta la fine del file la funzione ritorna 0, altrimenti ritorna 1.
  bool read_data_line(kinect_data* dat);

  // distruttore della classe, chiude il file
  ~kin_file_manager()
  {
  f.close();
  }
```

```
37
38    // funzione che chiude il file
39    void close()
40    {
41    f.close();
42    }
43
44 };
45
46 #endif // #ifndef _KIN_FILE_MANAGER_H_
```

Listing 11: librerie usate

3.3.2.3 File manager per arduino

La libreria "ard_file_manager.h" che contiene le funzioni per la gestione del file di testo relativo ai dati di arduino:

```
1  #pragma once
2
3  #ifndef _ARD_FILE_MANAGER_
4  #define _ARD_FILE_MANAGER_
5
6  #include <iostream>
7  #include <ostream>
8  #include <fstream>
9  #include <string>
10 #include "data_structure.h"
11
12
13 class ard_file_manager
14 {
15 private:
16   std::fstream f;
17   std::ios::_Openmode mode;
18   unsigned long int line_id = 0;
19
20 public:
21   ard_file_manager(std::string file_name, std::ios::_Openmode mode);
22
23   void write_data_line(arduino_data dat);
24
25   bool read_data_line(arduino_data* dat);
26
27   ~ard_file_manager()
28   {
29     f.close();
30   }
31
32   void close()
33   {
34     f.close();
35   }
36 };
37
38 #endif //#ifndef _ARD_FILE_MANAGER_
```

Listing 12: librerie usate

Questa libreria è strutturata in modo totalmente simile a quella per la gestione dei file del kinect trattata nel paragrafo 3.3.2.2 con l'unica differenza che la struttura dati utilizzata è relativa ad arduino e non al kinect.

3.3.2.4 strutture dati

La libreria "data_structure.h" è fondamentale e definisce le strutture dati utilizzate nel resto del programma:

```
1 #pragma once
2
3
```

```cpp
#ifndef _DATA_STRUCTURE_H_
#define _DATA_STRUCTURE_H_

#include <iostream>
#include <string>
#include <boost/array.hpp>

// struttura dati per gestire i dati provenienti dal kinect
struct kinect_data
{
  static const int number_of_joints = 3;

  // sottostruttura che contiene i dati di ogni giunto: il suo numero, il giunto attaccato ad esso, la posizione e l'angolo rispetto all'altro giunto
  struct joit_data
  {
    int joint_name = -1;
    int to_joint = -1;
    float position[3] = { 0, 0, 0 };
    float angle[3] = { 0, 0, 0 };
  };

  //Tabelle ad accesso diretto utilizzate per ottimizzare la procedura di estrazione dati dalla
  //classe kinect_class e la conversione dai punti nello spazio dello scheletro ad angoli
  //del braccio.
  int needed_joint[number_of_joints] = { 8, 9, 10 };
  int to_joint[24] = { -1, -1, -1, -1, -1, -1, -1, -1, 9, 10, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1 };
  int to_ref_joint[24] = { -1, -1, -1, -1, -1, -1, -1, -1, 0, 1, 2, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1 }; // tabella di accesso (chiave numero joint kinect) -> (restituisce l-indice del numero del giunto in needed_joint)
  float ref_Ang_x[number_of_joints] = { 0, 0, 0 }; //per  8, 9
  float ref_Ang_y[number_of_joints] = { 0, 0, 0 }; //per 8, 9
  float ref_Ang_z[number_of_joints] = { 180, 180, 0 }; //per 8, 9
  float mul_fctor_x[number_of_joints] = { 0, 0, 0 }; //per 8, 9
  float mul_fctor_y[number_of_joints] = { 1, 1, 0 }; //per 8, 9
  float mul_fctor_z[number_of_joints] = { 1, 1, 0 }; //per 8, 9

  joit_data needed_joints[number_of_joints];

  // tempo esatto in cui i dati sono stati acquisiti
  uint64_t frame_time = 0;

  // contatore usato nella post elaborazione dei dati
  uint64_t contatore = 0;

  // funzione per verificare se il frame acquisito \'e parziale o completo
  bool full_frame();

  // funzione per stampare i dati memorizzati
  void print_data();

  float jointAngleX(float *P1, float *P2);

  float jointAngleY(float *P1, float *P2);

  float jointAngleZ(float *P1, float *P2);

  void updateAngles();

  // funzione per convertire il numero del giunto in una stringa
  std::string joint_Enum_ToStr(int n, std::string language);
};

// struttura dati per gestire i dati provenienti da arduino
struct arduino_data
{
  // variabili che rappresentano i dati: dell'accelerometro, del giroscopio, del magnetometro e la temperatura
```

```
  float acc_xyz[3];
  float gy_xyz[3];
  float magn_xyz[3];
  float temp;

  // tempo esatto di acquisizione dei dati
  uint64_t frame_time = 0;

  // ?
  uint64_t contatore = 0;

  // funzione che permette di stampare il set di dati attualmente immagazzinato a schermo
  void print_data();
};

// struttura template che gestisce un set di dati del dataset
template<typename type_in, std::size_t N, typename type_out, std::size_t M>
struct dataset_data
{
  boost::array<type_in, N> in;
  boost::array<type_out, M> out;

  // stamapa i dati attualmente immagazzinati
  void print_data();
};

#endif //#ifndef _DATA_STRUCTURE_H_
```

Listing 13: librerie usate

3.3.2.5 Gestione seriale

Infine, l'ultima libreria fatta da noi ospita la classe che gestisce la seriale per la comunicazione tramite bluetooth con arduino:

```
#pragma once

#ifndef _SERIAL_HADLER_H_
#define _SERIAL_HADLER_H_

#include <iostream>
#include <string>
#include <boost/asio.hpp>
#include <boost/bind.hpp>
#include <boost/asio/serial_port.hpp>


class Serial
{
private:
  // dichiaro la porta seriale che verra inizializzata nel costruttore
  boost::asio::serial_port* port;

  // per ricomporre i float ricevuti uso un union
  union Scomp_float
  {
    float n_float;
    uint32_t n_int;
    uint8_t n_bytes[4];
  };

public:
  // nel costruttore si inizializza la seriale e si effettua la connessione al dispositivo
     ↪ bluetooth
  Serial(std::string com);

  // effettua una sincronizzazione iniziale tra arduino ed il pc
  void sinc();

  // riceve un float da arduino
```

```
35    float receive_float();
36
37    // metodo per inviare un carattere
38    void send_char(char ch);
39
40    // metodo per ricevere un carattere
41    char receive_char();
42
43
44    // riceve i dati di acc, gy, magn da arduino
45    // il metodo ritorna
46    //     -> 0 se la trasmissione \'e andata a buon fine
47    //     -> 1 se \'e fallita
48    int receive_data(float* acc_xyz, float* g_xyz, float* magn, float* temp);
49
50    ~Serial()
51    {
52      port->close();
53    }
54 };
55
56 #endif // #ifndef _SERIAL_HADLER_H_
```

Listing 14: librerie usate

Per trasmettere il blocco di dati da arduino al pc abbiabo sviluppato un "protocollo" di comunicazione, cioè i vari dati sono separati da sei tag di controllo che permettono di verificare se il dato è stato trasmesso correttamente o no.

```
1  void send_data(float* acc_xyz, float
       ↪ * g_xyz, float* magn, float
       ↪ temp)
2  {
3    send_char('a');
4    for (int i = 0; i < 3; i++)
5      send_float(acc_xyz[i]);
6    send_char('g');
7    for (int i = 0; i < 3; i++)
8      send_float(g_xyz[i]);
9    send_char('m');
10   for (int i = 0; i < 3; i++)
11     send_float(magn[i]);
12   send_char('t');
13   send_float(temp);
14   send_char(4);
15 }
```

(a) codice arduino

```
1  int Serial::receive_data(float*
       ↪ acc_xyz, float* g_xyz, float*
       ↪ magn, float* temp)
2  {
3    if (receive_char() != 'a')
4      return 1;
5    for (int i = 0; i < 3; i++)
6      acc_xyz[i] = receive_float();
7    if (receive_char() != 'g')
8      return 1;
9    for (int i = 0; i < 3; i++)
10     g_xyz[i] = receive_float();
11   if (receive_char() != 'm')
12     return 1;
13   for (int i = 0; i < 3; i++)
14     magn[i] = receive_float();
15   if (receive_char() != 't')
16     return 1;
17   temp[0] = receive_float();
18   if (receive_char() != 4)
19     return 1;
20   return 0;
21 }
```

(b) codice che gira sul pc

Figura 20: codice per l'invio e la ricezione dei dati necessari

Nel codice in figura 20a troviamo la funzione per inviare i dati da arduino, mentre in figura 20b troviamo il codice per ricevere gli stessi dati sul pc. Come si può vedere viene inviato un carattere di controllo prima di ogni dato, questo carattere serve per controllare l'integrità della comunicazione e deve essere ricevuto dal pc. Se ciò non accade significa che nella trasmissione è andato perso 1 byte e quindi arduino ed il pc vanno risincronizzati.

3.3.3 programma principale dell'acquisizione dati

Oltre a quste classi/strutture integrate in delle librerie separate ci sono altre classi/altro nel main, infatti abbiamo: "namespace logger", "class bounded_buffer", "class kinect_class", "class arduino_class", "class ard_handler", "class kin_handler", "void start_aquire_data" e "void acquire_data".

3.3.3.1 Logger

Per prima cosa è bene spiegare il logger, che viene usato nel resto del programma. Questa libreria serve esclusivamente per il debug e ha attributi specifici per lavoare con più thread contemporaneamente in modo asincrono. Serve fondamentalmente a stampare un testo a schermo, ma una volta settate le impostazioni si può aggiungere a quasto testo il nome del thread il tempo esatto in cui il messaggio viene stampato ed evitare che 2 thread stampino contemporaneamente nella console (se lo facessero il testo si mischierebbe e non si capirebbe nulla). La documentazione approfondita su come funziona questa libreria si può trovare qui. Nel nostro caso i settaggi usati sono: un id che identifica il thread, un tag che identifica il nome del trhead, un tag che identifica il livello di severità, il tempo assoluto e la stringa da stampare. Ad esempio se questa riga è posta all'inizio del main e il main thread viene chiamato "main":

```
BOOST_LOG_SEV(slg, logger::normal) << "hello log";
```

Listing 15: librerie usate

Allora viene stampato a schermo un messaggio formattato in questo modo:

```
id->(0x24a8): main < normal >   time->[00:00:00.000404]  -> hello log
```

Listing 16: librerie usate

Ciò consente di debuggare facilmente operazioni asincrone su più trhead. Il codice relativo hai settaggi necessari per ottenere quel tipo di formattazione è il "namespace logger":

```
namespace logger
{
  // definizione dei divesti livelli di severit\`a
  enum severity_level
  {
    normal,
    warning,
    error,
    critical
  };

  // definizione degli attributi usati
  BOOST_LOG_ATTRIBUTE_KEYWORD(severity, "Severity", severity_level)
  BOOST_LOG_ATTRIBUTE_KEYWORD(tag_attr, "Tag", std::string)
  BOOST_LOG_ATTRIBUTE_KEYWORD(timeline, "Timeline", boost::log::attributes::timer::value_type)
  BOOST_LOG_ATTRIBUTE_KEYWORD(tread_id, "Tread_id", boost::thread::id)
  BOOST_LOG_ATTRIBUTE_KEYWORD(tread_name, "tread_name", std::string)

  // provvede a sostituire il numero del severity_level con la relativa stringa
  std::ostream& operator<< (std::ostream& strm, severity_level level){...}

  // inizializzazione della formattazione del logger
  void init(){...}

  // definizione dei vari parametri
  boost::log::sources::severity_logger< severity_level > f_init(){...}
  void attr_thread(...){...}
  void attr_time(...){...}
  void attr_tag(...){...}
  void attr_thread_name(...){...}
};
```

Listing 17: librerie usate

Inoltre per funzionare correttamente richiede che all'inizio del main siano posti alcuni settaggi:

```
// inizializzazione logger
logger::init();
auto slg = logger::f_init();
attr_thread(&slg);
attr_time(&slg);

```

```
// imposto il nome del main trhead in "main"
logger::attr_thread_name(&slg, "main");
```

Listing 18: librerie usate

3.3.3.2 Buffer FIFO

Un altra classe usata nel resto del programma è "class bounded_buffer", questa classe ospita funzioni specifiche per immagazzinare temporaneamente dei dati gestendone l'ingresso e l'uscita dal buffer. Fondamentalmente costituisce un registro FIFO indipendente dal tipo di dato che si utilizza. Questo tipo di buffer viene comunemente usato in caso si ha la struttura thread che produce dati - thread che consuma dati: il produttore acquisisce i dati e li inserisce nel buffer, il consumatore estrae i dati e li processa. La dimensione del buffer viene specificata quando lo si inizializza e deve essere sufficentemente grande in modo da garantire la non perdita di dati e il non rallentamento del trhead produttore. Nel nostro caso abbiamo adottato questo tipo di buffer per poter garantire l'asincronicità delle acquisizioni tra arduino e il kinect, così facendo si garantisce la massima velocità tra le acquisizioni dei dati. Il codice relativo a questa classe è stato quasi interamente presto dagli esempi della libreria boost, qui ed è:

```
template <class T>
class bounded_buffer
{
public:

  typedef boost::circular_buffer<T> container_type;
  typedef typename container_type::size_type size_type;
  typedef typename container_type::value_type value_type;
  typedef typename boost::call_traits<value_type>::param_type param_type;

  // nel costruttore si imposta la capacit\'a del buffer
  explicit bounded_buffer(size_type capacity) : m_unread(0), m_container(capacity) {}

  // questa funzione permette di aggiungere un elemeto al buffer
  void push_front(param_type item){...}

  // questa funzione permette di estrarre un elemento dal buffer
  void pop_back(value_type* pItem){...}

  // restituisce la percentuale di riempimento del buffer
  float percentage_of_filling(){...}

  // svuota il buffer
  void flush(){...}

private:
  bounded_buffer(const bounded_buffer&);             // Disabled copy constructor
  bounded_buffer& operator = (const bounded_buffer&); // Disabled assign operator

  bool is_not_empty() const { return m_unread > 0; }

  bool is_not_full() const { return m_unread < m_container.capacity(); }

  size_type m_unread;
  container_type m_container;
  boost::mutex m_mutex;
  boost::condition_variable m_not_empty;
  boost::condition_variable m_not_full;
};
```

Listing 19: librerie usate

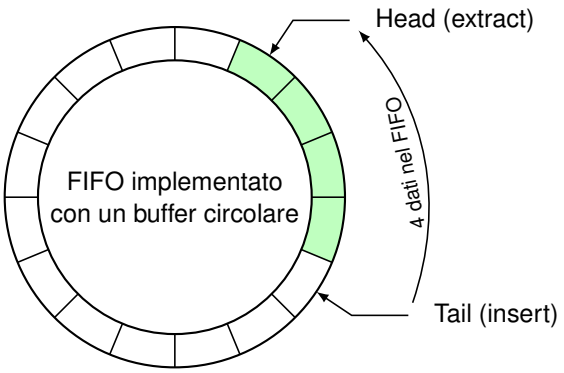


Figura 21: Schema di funzionamento circular buffer

La classe nel listato 19 utilizza un buffer circolare per implementare il registro FIFO, il funzionamento di questo buffer è descritto dalla figura 21: ogni volta che si aggiunge un dato viene occupata una locazione in testa al buffer ed ogni volta che si rimuove un dato viene liberata una locazione in coda, cioò permette di riusare continuamente le stesse locazioni di memoria.

3.3.3.3 Classe gestione kinect

La classe "kinect_class" provvede a leggere i dati dal kinect ed aggiungerli al suo corrispettivo registro FIFO. L'estrazione dei dati dal kinect è stata realizzata sulla base dell'esempio body basic D2D integrato nell'sdk browser fornito da microsoft. Il codice relativo a questa classe è:

```
1  class kinect_class
2  {
3  public:
4
5    // Current Kinect
6    IKinectSensor* m_pKinectSensor;
7    ICoordinateMapper* m_pCoordinateMapper;
8
9    // Body reader
10   IBodyFrameReader* m_pBodyFrameReader;
11
12   real_time t;
13
14   bounded_buffer<kinect_data>* FIFO_kin;
15
16   kinect_class(bounded_buffer<kinect_data>* data_FIFO){...}
17
18   void start()
19   {
20
21     bool flag = false;
22     while (true)
23     {
24       {...}
25
26       // prendo il tempo
27       uint64_t t1 = t.get_curr_time();
28
29       // acquisisco i dati e li sposto in data
30       flag = Update(&data);
31
32       // se l'acquisizione \'e andata a buon fine procedo
33       if (flag)
34       {
35         // prendo un altra volta il tempo
36         uint64_t t2 = t.get_curr_time();
37         // cos\'i da poter impostare che il tempo in cui sono stati acquisiti i dati sia
38         // la media tra il tempo prima di cominciare l'acquisizione ed il tempo dopo che essa \'
               ↪ e finita
39         data.frame_time = (t1 + t2) / 2;
40
41         // inserisco i dati cos\'i acquisiti nel registro fifo
```

```
42          FIFO_kin->push_front(data);
43
44          // faccio aspettare un po di tempo a questo trhead perch\'e altrimenti il kinect da
                ↪ errore,
45          // dato che non supporta un acquisizione a pi\'u di 30fps
46          boost::this_thread::sleep(boost::posix_time::milliseconds(35));  al kinect scoppia
47
48        }
49
50        {...}
51
52      }
53    }
54
55    // inizializza il kinect, ritorna 1 se ha avuto successo, 0 altrimenti
56    int InitializeDefaultSensor(){...}
57
58    // acquisisce i dati e li passa alla funzione per processarli
59    bool Update(kinect_data* data)
60    {
61      {...}
62
63      flag = ProcessBody(BODY_COUNT, ppBodies, data);
64
65      {...}
66
67      return flag;
68
69    }
70
71    // questa funzione provvede a spostare i dati da joints a data sistemandoli nella loro
          ↪ struttura dati,
72    // una spiegazione di come ci\'o viene fatto verra data nel capitolo successivo
73    bool handle_data(Joint* joints, kinect_data* data){...}
74
75
76    // quetsta funzione separa le diverse persone rilevate e trasferisce le informazioni
77    // relative hai giunti della prima persona alla funzione handle_data
78    bool ProcessBody(int nBodyCount, IBody** ppBodies, kinect_data* data)
79    {
80      {...}
81
82      for (int i = 0; i < nBodyCount; ++i)
83      {
84        {...}
85
86        Joint joints[JointType_Count];
87
88        hr = pBody->GetJoints(_countof(joints), joints);
89
90        return handle_data(joints, data);
91
92        {...}
93
94      }
95
96      {...}
97    }
98
99    // provvede a spegnere il kinect e deallocare le sue risorse nel caso la classe venga
          ↪ distrutta
100   ~kinect_class(){...}
101   template<class Interface>
102   inline void SafeRelease(Interface *& pInterfaceToRelease){...}
103
104 };
```

Listing 20: librerie usate

3.3.3.4 Classe gestione arduino

La classe "arduino_class" provvede a caricare continuamente i dati provenienti dal kineck nel registro fifo, infatti la funzione "start" viene lanciata come thread e continua a caricare i dati fino al termine del programma:

```
class arduino_class
{
public:
  Serial* ser;
  real_time t;
  bounded_buffer<arduino_data>* FIFO_ard;

  // il costruttore provvede ad inizializzare la seriale e sincronizzare i 2 dispositivi
  arduino_class(bounded_buffer<arduino_data>* data_FIFO, string com_port)
  {
    ser = new Serial(com_port);
    cout << "serial ok" << endl;
    FIFO_ard = data_FIFO;
    ser->sinc();
  }

  // questa funzione viene lanciata all'avvio del thread e provvede ad acquisire i dati ed
      ↪ inserirli nel registro fifo
  void start()
  {
    while (true)
    {
      arduino_data data;

      uint64_t t1 = t.get_curr_time();
      ser->receive_data(data.acc_xyz, data.gy_xyz, data.magn_xyz, &data.temp);
      uint64_t t2 = t.get_curr_time();

    // per ottenere un accurata misurazione del tempo viene preso prima e dopo l'acquisizione e
        ↪ poi fatta la media dei valori ottenuti
      data.frame_time = (t1 + t2) / 2;

      FIFO_ard->push_front(data);
    }
  }
};
```

Listing 21: librerie usate

3.3.3.5 Gestore di arduino e del kinect

Dopo i 2 thread producer si hanno i 2 thread consumer: "class ard_handler" e "class kin_handler", che si occupano rispettivamente di salvare su file di testo le informazioni di arduino e del kinect.

```
/// <summary>
/// handle the data from arduino and write it on a file
/// </summary>
class ard_handler
{
private:
  bounded_buffer<arduino_data>* ard;
  ard_file_manager* f;

public:
  // il costruttore provvede ad inizializzare la classe per gestire i file di testo di arduino
  ard_handler(bounded_buffer<arduino_data>* ard, string file_name)
  {
    this->ard = ard;
    f = new ard_file_manager(file_name, std::ios::out);
  }

  // questa funzione \'e quella che viene effettivamente lanciata all'avvio del thread e
      ↪ provvede a scaricare il buffer scrivendo i dati sul file fino al raggiungimento della
      ↪ quota predichiarata "tot_esempi"
```

```
  void start()
  {
  // inizializzazione logger
    auto slg = logger::f_init();
    attr_thread(&slg);
    logger::attr_thread_name(&slg, "ard_handler");

    BOOST_LOG_SEV(slg, logger::normal) << "hello log";

    for (int i = 0; i < tot_esempi; i++)
    {
      arduino_data data_ard;
      ard->pop_back(&data_ard);

      f->write_data_line(data_ard);

    }

  // segnalo sulla console qunado il thread ha finito il suo lavoro
    BOOST_LOG_SEV(slg, logger::normal) << "end";
  }
};
```

Listing 22: librerie usate

La classe "kin_handler" esegue esattamente le stesse operazioni della precedente con la sola differenza che usa "kin_file_manager" al posto di "ard_file_manager".

```
/// <summary>
/// handle the data from the kinect and write it in the file
/// </summary>
class kin_handler
{
private:
  bounded_buffer<kinect_data>* kin;
  kin_file_manager* f;

public:
  kin_handler(bounded_buffer<kinect_data>* kin, string file_name){...}

  void start(){...}
};
```

Listing 23: librerie usate

3.3.3.6 Funzioni di lancio dei thead per l'acquisizione ed il salvataggio dei dati

Dopo tutte queste classi si hanno 2 funzioni che si occupano di inizializzare le altre classi, lanciare tutti i thread neccessari ed acquisire dall'utente i nomi dei file etc.

```
void start_aquire_data(string out_ard, string out_kin, string com_port)
{
  // in inizializzo il logger per questa funzione
  auto slg = logger::f_init();
  attr_thread(&slg);
  logger::attr_thread_name(&slg, "main");

  // dichiaro i 2 buffer FIFO
  bounded_buffer<arduino_data> data_FIFO_ard(20);
  bounded_buffer<kinect_data> data_FIFO_kin(20);

  // chiamo i costruttori delle classi che gestiscono il kinect e arduino
  // al costruttore di arduino gli devo passare la com a cui il bluetooth \'e collegato
  arduino_class a(&data_FIFO_ard, com_port);
  kinect_class k(&data_FIFO_kin);

  // chiamo i costruttori delle classi per la scrittura nei file di testo
  ard_handler a_h(&data_FIFO_ard, out_ard);
  kin_handler k_h(&data_FIFO_kin, out_kin);
```

```
20
21    BOOST_LOG_SEV(slg, logger::normal) << "costructor end" << std::endl;
22
23    // lancio i thread che "producono i dati"
24    boost::thread ard([&a]() { a.start(); });
25    boost::thread kin([&k]() { k.start(); });
26
27    BOOST_LOG_SEV(slg, logger::normal) << "producer thread launced" << std::endl;
28
29    // lancio i thread che "consumnano dati"
30    boost::thread ard_h([&a_h]() { a_h.start(); });
31    boost::thread kin_h([&k_h]() { k_h.start(); });
32
33    BOOST_LOG_SEV(slg, logger::normal) << "consumer thread launched" << std::endl;
34
35    // aspetto all'infinito (finch\'e il programma non viene chiuso
36    kin.join();
37    ard.join();
38    kin_h.join();
39    ard_h.join();
40 }
```

Listing 24: librerie usate

infine c'è la funzione che si occupa di acquisire i dati dall'utente e passarli alla funzione appena vista.

```
1  void acquire_data()
2  {
3    string ard_file_mane = "";
4    string kin_file_mane = "";
5    string arduino_com_port = "";
6
7    // acquisisco il nome dei 2 file che conterranno i dati di arduino e i dati del kinect
8    cout << "insert the file names: " << endl << "\t arduino -> ";
9    std::getline(std::cin, ard_file_mane);
10   cout << "\t kinekt -> ";
11   std::getline(std::cin, kin_file_mane);
12
13   // acquisisco il numero della com a cui \'e collegato il bluetooth del computer
14   cout << "insert the arduino com port -> : ";
15   std::getline(std::cin, arduino_com_port);
16
17   // lancio la funzione passandogli i dati appena acquisiti
18   start_aquire_data(ard_file_mane, kin_file_mane, arduino_com_port);
19 }
```

Listing 25: librerie usate

3.3.3.7 Main

Infine il main chiama semplicemente la funzione appena trattata.

```
1  int main()
2  {
3    logger::init();
4
5    auto slg = logger::f_init();
6    attr_thread(&slg);
7    attr_time(&slg);
8    logger::attr_thread_name(&slg, "main");
9
10   BOOST_LOG_SEV(slg, logger::normal) << "hello log";
11
12   acquire_data();
13
14   return 0;
15 }
```

Listing 26: librerie usate

3.3.4 Riassunto

Riassumendo la struttura generale di questo programma si ha:

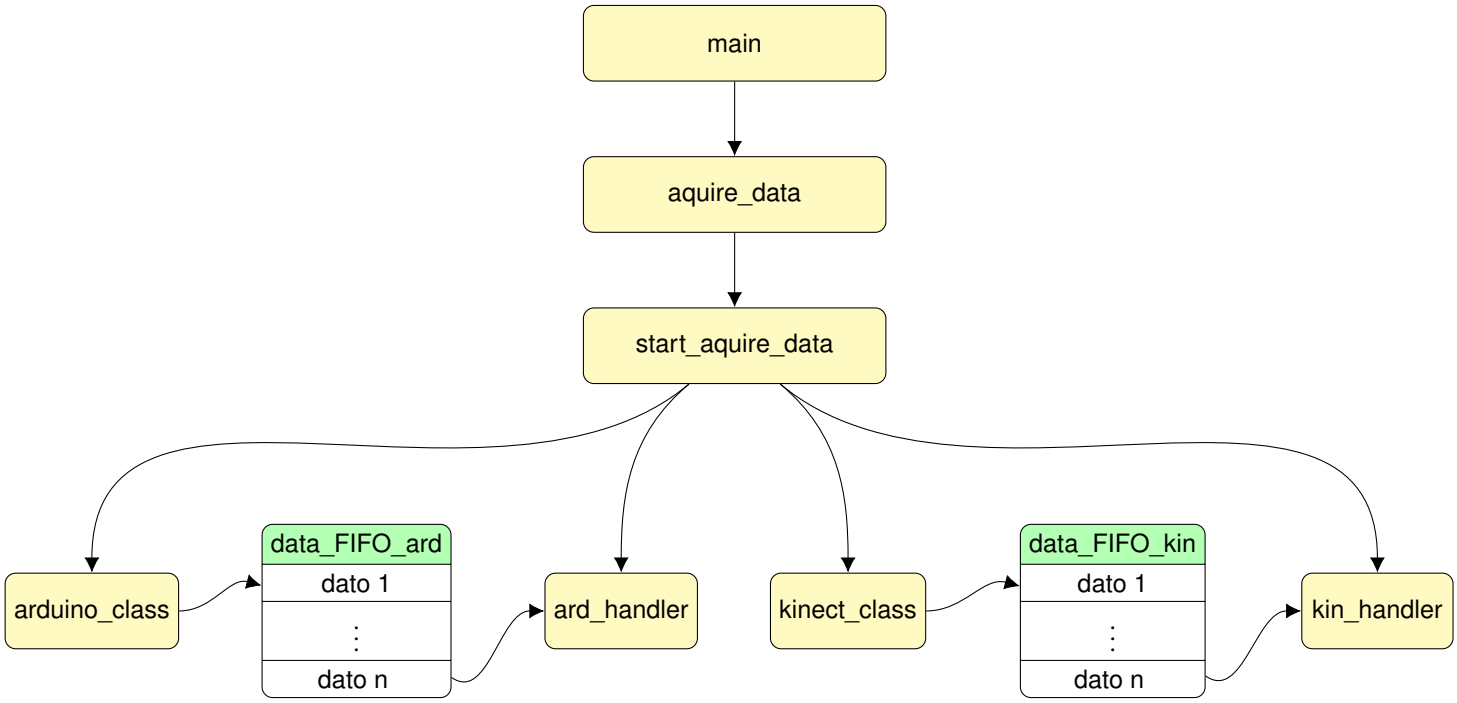


Figura 22: programmi che si occupano dell'immagazzinamento dei dati per la creazione del dataset

Cioè il "main" chiama "aquire_data" che chiama a sua volta "start_aquire_data". Dopodichè vengono lanciati i 4 thread ("arduino_class", "ard_handler", "kinect_class", "kin_handler"). I thread che si occupano di prelevare i dati dai dispositivi sono "arduino_class" e "kinect_class", che inseriscono i dati nel corrispettivo buffer FIFO. Infine i thread che si occupano di scrivere i dati su i file di testo sono "ard_handler" e "kin_handler", che scrivono i dati prelevati dal buffer nel corrispettivo file di testo.

3.4 Programma che si occupa di post-elaborare i dati

Dopo aver acquisito i dati di arduino e del kinect bisogna effettuare una post elaborazione per ottenere coppie di valori riferiti a circa lo stesso istante temporale. Il programma usato per fare ciò riprende molte cose dal codice del programma "prg_get_data", le librerie usate sono le stesse ed anche le classi "kin_file_manager", "kin_file_manager" e "data_structure.h". L'unica classe nuova è "make_dataset":

```
1  class make_dataset
2  {
3  private:
4    // dichiaro i puntatori alle classi per la gestione dei file cos\'i da renderle disponibili
       ↪ per tutta la classe
5    ard_file_manager *f_a;
6    kin_file_manager *f_k;
7
8    // delta di accettazione in millisecondi
9    const int range = 35;
10
11   // dichiaro 2 liste per conservare i dati
12   list<arduino_data> a_dat;
13   list<kinect_data> k_dat;
14
15   // devo dichiararare il numero di ingressi e di uscite della rete
16   static const std::size_t n_in = 9;
17   static const std::size_t n_out = 4;
18
19   // dichiaro una lista di tipo dataset_data che verr\'a riempita con i dati accoppiati
20   list<dataset_data<float, n_in, float, n_out>> dataset;
21
22   // questa funzione si occupa di scrivere il dataset sul file
23   void write_dataset(string dataset_file_name){...}
24
25   // restituisce 1 solo se i 2 tempi che gli vengono passati differiscono meno della soglia
       ↪ impostata sopra
```

```
  bool check_sync(uint64_t t1, uint64_t t2){...}

  // questo metodo sposta i dati dalle strutture "arduino_data" e "kinect_data" alla struttura "dataset" dichiarata globalmente nella classe
  void transfer_data_to_dataset(arduino_data* ard_iter, kinect_data* kin_iter){...}

  // restituisce la differenza temporale tra i tempi passati alla funzione
  uint32_t time_distance(uint32_t t1, uint32_t t2){...}

  // sincronizza i dati caricati in "a_dat", "k_dat" e li sposta in "dataset" attraverso la funzione "transfer_data_to_dataset". Questa funzione non tiene conto dei "cloni", nella sezione successiva vi \'e una spiegazione pi\'u dettagliata.
  void sync_with_clones(){...}

  // stessa cosa della funzione precedente ma eliminando i cloni.
  void sync_without_clones(){...}

  // carica i dati dai file usando "f_a" e "f_k", sposta i dati acquisiti in "a_dat" e k_dat"
  void load_data(){...}

public:
  // l'unica funzione accessibile \'e il costruttore che provvede ad inizializzare i  file manager, caricare i dati, sincronizzarli (con o senza cloni) e scrivere il file del dataset con i dati appena generati.
  make_dataset(string ard_file_name, string kin_file_name, string dataset_file_name, bool want_clones)
  {
    f_a = new ard_file_manager(ard_file_name, std::ios::in);
    f_k = new kin_file_manager(kin_file_name, std::ios::in);

    load_data();

    if (want_clones)
    {
      sync_with_clones();
    }
    else
    {
      sync_without_clones();
    }

    write_dataset(dataset_file_name);

  }

  // distruttore della classe
  ~make_dataset(){...}

};
```

Listing 27: classe make_dataset

In questa classe le 2 funzioni principali sono "sync_with_clones" e "sync_without_clones" che provvedono a costruire le coppie di dati ed a salvarle nella struttura dati "dataset". Analizziamo la funzione "sync_with_clones":

```
void sync_with_clones()
{
  uint64_t time_old_good_pair = 0;

  uint64_t time_old_ard = 0;
  uint64_t time_old_kin = 0;

  auto ard_iter = a_dat.begin();
  auto kin_iter = k_dat.begin();

  uint32_t ard_cont = 0;
  uint32_t kin_cont = 0;

  uint32_t old_ard_cont = 0;
  uint32_t old_kin_cont = 0;
```

```
16
17    while (true)
18    {
19      // controllo la sincronia
20      if (check_sync(ard_iter->frame_time, kin_iter->frame_time))
21      {
22        // aggiungo il dato al dataset
23        transfer_data_to_dataset(&(*ard_iter), &(*kin_iter));
24
25        // stampo delle info utili a capire la qualit\'a dei dati acquisiti
26        cout << abs((long long)(ard_iter->frame_time - kin_iter->frame_time)) << "  kin: " <<
              ↪ kin_cont << " ard: " << ard_cont << endl;
27        cout << "time since another good pair: " << ((ard_iter->frame_time + kin_iter->frame_time)
              ↪  / 2 - time_old_good_pair) << endl;
28        cout << "ard frame rate: " << (1.0f / ((ard_iter->frame_time - time_old_ard) / 1000.0f)) <
              ↪ < "Hz" << endl;
29        cout << "kin frame rate: " << (1.0f / ((kin_iter->frame_time - time_old_kin) / 1000.0f)) <
              ↪ < "Hz" << endl;
30
31        // salvo i tempi
32        time_old_good_pair = (ard_iter->frame_time + kin_iter->frame_time) / 2;
33        time_old_ard = ard_iter->frame_time;
34        time_old_kin = kin_iter->frame_time;
35
36        // controllo se ci sono pi\'u dati che si accoppiano con uno solo (solo debug)
37        if ((kin_cont == old_kin_cont) || (ard_cont == old_ard_cont))
38        {
39          cout << "dati multipli nello stesso range" << endl;
40        }
41        else
42        {
43          old_kin_cont = kin_cont;
44          old_ard_cont = ard_cont;
45        }
46
47        cout << endl;
48
49      }
50      else
51      {
52        cout << "dato scartato" << endl << endl;
53      }
54
55      // incremento chi tra i 2 dati \'e il pi\'u vecchio
56      if (ard_iter->frame_time >= kin_iter->frame_time)
57      {
58        kin_iter++;
59        kin_cont++;
60      }
61      else
62      {
63        ard_iter++;
64        ard_cont++;
65      }
66
67      // se sono arrivato alla fine esco dal ciclo
68      if ((kin_iter == k_dat.end()) || (ard_iter == a_dat.end())) { break; }
69    }
70  }
```

Listing 28: funzione sync_with_clones

Questa prima funzione effettua il pair dei dati senza però tener conto che, dati i differenti frame rate di arduino e del kinect, un singolo dato del kinect potrebbe accoppiarsi con più di un dato di arduino e viceversa. Questi "cloni" interferiscono con il processo di apprendimento della rete e quindi si è passati alla funzione "sync_without_clones" che effettua la stessa operazione ma tenendo conto che potrebbero esserci più dati di un tipo accoppiati con uno dell'altro e salva solo la coppia che ha la differenza di tempo minore.

```
1  void sync_without_clones()
2  {
3    uint64_t time_old_good_pair = 0;
```

```
4
5   uint64_t time_old_ard = 0;
6   uint64_t time_old_kin = 0;
7
8   auto ard_iter = a_dat.begin();
9   auto kin_iter = k_dat.begin();
10
11  uint32_t ard_cont = 0;
12  uint32_t kin_cont = 0;
13
14  uint32_t old_ard_cont = 0;
15  uint32_t old_kin_cont = 0;
16
17  kinect_data k_tmp = *kin_iter;
18  arduino_data a_tmp = *ard_iter;
19
20  for(;;)
21  {
22    // controllo la sincronia
23    if (check_sync(ard_iter->frame_time, kin_iter->frame_time))
24    {
25      // stampo i tempi per valutere la qualit\'a del dataset
26      cout << "common " << abs((long long)(ard_iter->frame_time - kin_iter->frame_time)) << "
          ↪ kin: " << kin_iter->contatore << " ard: " << ard_iter->contatore << endl;
27      cout << "ard frame rate: " << (1.0f / ((ard_iter->frame_time - time_old_ard) / 1000.0f)) <
          ↪ < "Hz" << endl;
28      cout << "kin frame rate: " << (1.0f / ((kin_iter->frame_time - time_old_kin) / 1000.0f)) <
          ↪ < "Hz" << endl;
29
30      // salvo i tempi attuali
31      time_old_ard = ard_iter->frame_time;
32      time_old_kin = kin_iter->frame_time;
33
34      // se il contatore del kinect non \'e variato dall'ultima volta significa che ci sono pi\'
          ↪ u dati riferiti ad arduino che si accoppiano ad un solo dato del kinect
35      if (kin_cont == old_kin_cont)
36      {
37
38        cout << "multi ard in the same kin" << endl;
39        if (time_distance(kin_iter->frame_time, ard_iter->frame_time) < time_distance(k_tmp.
              ↪ frame_time, a_tmp.frame_time))
40        {
41          a_tmp = *ard_iter;
42        }
43      }
44      // se il contatore dei arduino non \'e variato dall'ultima volta significa che ci sono pi
          ↪ \'u dati riferiti al kinect che si accoppiano ad un solo dato di arduino
45      else if (ard_cont == old_ard_cont)
46      {
47        cout << "multi kin in the same ard" << endl;
48        if (time_distance(kin_iter->frame_time, ard_iter->frame_time) < time_distance(k_tmp.
              ↪ frame_time, a_tmp.frame_time))
49        {
50          k_tmp = *kin_iter;
51        }
52      }
53      \\ se entrambi sono variati significa che la sequenza \'e finita e posso salvare i dati
54      else
55      {
56        cout << "sequence reset" << endl;
57
58        // salvo la coppia salvata
59        transfer_data_to_dataset(&a_tmp, &k_tmp);
60
61        // stampo i tempi per le valutazioni
62        cout << "best " << abs((long long)(a_tmp.frame_time - k_tmp.frame_time)) << "  kin: " <<
              ↪  k_tmp.contatore << " ard: " << a_tmp.contatore << endl;
63        cout << "time since another good pair: " << ((a_tmp.frame_time + k_tmp.frame_time) / 2 -
              ↪  time_old_good_pair) << endl;
```

```
        time_old_good_pair = (a_tmp.frame_time + k_tmp.frame_time) / 2;

        // salvo i dati correnti e i contatori correnti
        old_kin_cont = kin_cont;
        old_ard_cont = ard_cont;
        a_tmp = *ard_iter;
        k_tmp = *kin_iter;
      }

      cout << endl;

    }
    else
    {
      cout << "dato scartato" << endl << endl;
    }

    // incremento chi tra i 2 dati \'e il pi\'u vecchio
    if (ard_iter->frame_time >= kin_iter->frame_time)
    {
      kin_iter++;
      kin_cont++;
    }
    else
    {
      ard_iter++;
      ard_cont++;
    }

    // se sono arrivato alla fine dei dati esco
    if ((kin_iter == k_dat.end()) || (ard_iter == a_dat.end())) { break; }
  }
}
```

Listing 29: funzione sync_without_clones

La funzione che si occupa di acquisire i nomi dei file e altro dall'utente è "post_elaborate":

```
void post_elaborate()
{
  string ard_file_mane = "";
  string kin_file_mane = "";
  string dataset_file_name = "";
  string tmp = "";
  bool clones = false;

  // acquisisco i nomi dei file dei dati di arduino e del kinect
  cout << "insert the input file names: " << endl
    << "\t arduino -> ";
  std::getline(std::cin, ard_file_mane);
  cout << "\t kinekt -> ";
  std::getline(std::cin, kin_file_mane);

  // acquisisco il nome del file in cui verr\'a scritto il dataset
  cout << "insert the ouptut file name -> : ";
  std::getline(std::cin, dataset_file_name);

  // chiedo se si vuole il dataset con o senza cloni
  cout << "do you want clones in the data? [y/n] -> : ";
  std::getline(std::cin, tmp);

  if (tmp == "y")
  {
    clones = true;
  }

  // chiamo la classe make_dataset passandogli i dati appena acquisiti
  make_dataset(ard_file_mane, kin_file_mane, dataset_file_name, clones);
}
```

Listing 30: funzione "post_elaborate"

3.5 OpenLearn documentazione

Questa parte della relazione è dedicata all'illustrazione e al commento della struttura e delle parti principali del componente principale del progetto, la libreria di machine learning sviluppata per la creazione, l'addestramento e l'utilizzo dei modelli neuronali. La libreria è stata sviluppata per un utilizzo general-purpose, quindi non finalizzata alla risoluzione dello specifico problema proposto, il quale è stato utilizzato per testare la risposta di varie strutture neurali alla risoluzione di un problema con soluzione nota ma attraverso l'elaborazione di dati di ingresso fortemente affetti da rumore.
Nella documentazione verranno mostrate soltanto le caratteristiche principali della libreria, le classi, le strutture e i metodi più importanti effettivamente utilizzati benche la libreria ne presenti altri predisposti per future revisioni ed espansioni.

3.5.1 Le librerie importate

Le prime librerie importate sono le librerie per l'utilizzo del linguaggio C/CUDA, proprietario di Nvidia, per la programmazione delle schede della stessa azienda.

```
1 //////////////CUDA INCLUDES///////////////
2 #include "cuda.h"
3 #include "cuda_runtime.h"
4 #include <device_functions.h>
5 #include "device_launch_parameters.h"
```

Listing 31: librerie cuda

Sono state importate poi diverse librerie standard del linguaggio C++, principalmente:
la classe vector è stata usata per l'incapsulamento e la manipolazione dei dati.
le classi *stream sono state utilizzate per il salvataggio e il caricamento dei modelli su file di testo.
la libreria windows per l'utilizzo delle funzioni di timing ad alta precisione.

```
1
2 #include "stdafx.h"
3 #include <iostream>
4 #include <stdio.h>
5 #include <fstream>
6 #include <sstream>
7 #include <string>
8 #include <windows.h>
9 #include <limits>
10 #include <math.h>
11 #include <vector>
12 #include <list>
13 #include <time.h>
14 #include <algorithm>
15 #include <assert.h>
```

Listing 32: librerie STD

3.5.2 classi di creazione e addestramento modelli su CPU

Di seguito sono illustrate le strutture che formano lo scheletro dei modelli, tali strutture sono state pensate per essere completamente modulari e per permettere la realizzazione di diversi modelli, dalle più semplici feed-forward, alle più complesse ricorrenti. Tale approccio permette la massima maneggevolezza della struttura sacrificando l'efficienza che deriva da una computazione matriciale.

```
1 //prototipo struttura neuron
2 struct Neuron;
3 //dichiarazione puntatore a neuron
4 typedef Neuron *ptNeuron;
5
6 //La struttura arc, presente in array nella struttura Neuron
7 //conserva i dati delle connessioni e il puntatore alla struttura Neuron puntata
8 struct arc {
9   ptNeuron target = nullptr;
10   float weight = 0;
11   float oldDelta = 0;
12   //bool enabled = true;
```

```
};

//La struttura interArc è stata creata per l'addestramento MLP supportato da un rete ricorrente
//fornisce un arco di interconnessione tra i neuroni di 2 reti
struct interArc {
  ptNeuron target = nullptr;
  ptNeuron base = nullptr;
};

//La struttura Neuron è l'unità base delle strutture delle reti e ne conserva tutti i parametri
struct Neuron {
  //OutArcs è il vettore di connessioni ai neuroni degli strati successivi
  // to inizialize: = new vector<arc>(10);
  vector<arc> OutArcs;
  // numero di archi in uscita
  u_int numOutArcs = 0;
  //numero di archi in ingresso
  u_int numInArcs = 0;
  //indice riga del neurone
  u_int layer = 0;
  //indice della colonna del neurone
  u_int column = 0;
  //peso del bayes
  float bayes = 0.01f;
  //ultima variazione del peso
  float oldBayesDelta = 0;
  // vettore delle interconnessioni temporali
  //vector<float> timeBayes;
  // vettore contenente la percentuale di influenza relativa ad ogni input
  vector<float> influenceInput;
  // vettore contenente la percentuale di influenza relativa all'errore  retropropagato da ogni
      ↪ output
  vector<float> influenceOutput;
  //potenziale attuale del neurone
  float output = 0;
  //somma in valore assoluto degli input del neurone
  float absOutSum = 0;
  //somma in valore assoluto delle variazioni dei pesi
  float absDeltaSum = 0;
  //errore di retropropagazione
  float BPerr = 0;
  //ogni neurone è contraddistinto da un indice unico che si riferisce alla sua posizione
  int neurIdx = 0;
};

//La struct Layer è stata realizzata per contenere un layer di neuroni
struct Layer {
  vector<Neuron> Neurons;
  u_int numNeurons = 0;
};

//Omap conserva la scala di conversione tra l'output
struct Omap {
  float maxValue = 1;
  float minValue = 0;
};

// struttura necessaria per l'inserimento di un nuovo neurone
struct conMap {
  u_int startLyr;
  u_int startCol;
  u_int arcRef;
  u_int targetLyr;
  u_int targetCol;
};

//conserva una serie temporale prima dell'accomulazione in una struct Dataset
struct timeSeries {
  list<float> evento;
```

```
};

//struttura contenente il singolo esempio
struct example {
  vector<float> input;
  vector<float> Doutput;
};

//Struttura contenente l'intero Dataset di una rete
struct Dataset {
  vector<example> trainingSet; // vettore esempi del training set
  vector<example> validationSet; // vettore esempi del validation set
  float triningErr = 0; //percentuale di errore training set
  float validationErr = 0; //percentuale di errore validation set
};
```

Listing 33: struct strutturali del modello

La classe DatasetCore è stata realizzata per permettere l'estrazione dei dataset da vari formati, riportandoli nello standard di utilizzato dalle classi che li utilizzano per effettuare gli addestramenti.

```
class DatasetCore {
public:
  // lista dei dataset
  list<Dataset> Datasets;

  // costruttore DatasetCore
  DatasetCore() { ... }

  ////////////////////MANIPOLAZIONE DATASET/////////////////////////////////////
  //funzione che consente la lettura di serie temporali da csv e le ristruttura in un dataset
  void readTimeSeriesCsv(string filename, int outStep, int inEx, float trainingPerc) { ... }

  //estrae un dataset dalla lista interna della classe, la parte training se il flag e true
  // o la parte validation se il flag è false
  vector<example> getDataset(int n, bool training = true) { ... }
  ////////////////////////////////////////////////////////////////////////////////

  ////////////////////////ALTRE FUNZIONI/////////////////////////////////////////
  //metodo interno per contare le righe di un file
  int coutnRow(string filename) { ... }
```

Listing 34: class DatasetCore

La classe network fornisce le variabili e i metodi generali che la maggior parte degli oggetti rete utilizza.

```
class Network {

public:
  // vettore di struct layer
  vector<Layer> Layers;
  // vettore di esempi per l'apprendimento
  vector<example> examples;
  // vettore contenente i valori di rimappatura del'output della rete
  vector<Omap> map;

  // nome del file contenente la struttura della rete
  string genoma = "";
  // Layer nella rete compresi strato input e strato output
  u_int nLayers = 0;
  // numero totale neuroni nella rete
  u_int nNeurons = 0;
  //numero totale di tutti gli archi presenti nella rete
  u_int nArc = 0;

  //costruttore
  Network(string filename) {
    genoma = filename;
  }

  ////////////////FUNZIONE COSTRUZIONE RETE DA FILE/////////////////////////////////
```

```
//dato il nome del file contenete la struttura carica l'oggetto corrispondente
void getNetParams() { ... }
////////////////////////////////////////////////////////////////////////////////////

////////////////////FUNZIONE COSTRUZIONE DATASET DA FILE//////////////////////////////
//dato il file contenente un dataset prestrutturato correttamente lo carica nel modello
void getDataset(string filename) { ... }
////////////////////////////////////////////////////////////////////////////////////

////////////////////SALVA RETE SU FILE//////////////////////////////////////////////
//salva la rete sul file specificato
void saveNet(string filename = "") { ... }
////////////////////////////////////////////////////////////////////////////////////

////////////////////////SALVA DATASET SU FILE///////////////////////////////////////
//salva il datset sul file specificato
void saveDataset(string filename) { ... }
////////////////////////////////////////////////////////////////////////////////////

//////////////////////// FUNZIONI MODIFICA RETE////////////////////////////////////
//elimina un arco dati i parametri diriferimento
void deleteArc(int Nlayer, int Ncolumn, int targetLayer, int targetColumn) { ... }

//elimina un neurone, compresi tutti gli archi ad esso connesso
void deleteNeuron(int Nlayer, int Ncolumn) { ... }

//aggiunge un arco dati i riferimenti al neurone di partenza e di arrivo
void addArc(int Nlayer, int Ncolumn, int targetLayer, int targetColumn) { ... }
////////////////////////////////////////////////////////////////////////////////////

//////////////////////////FUNZIONI VARIE////////////////////////////////////////////
float DsigOut(int Layer, int Neuron) { ... }
float sigmoid(float x) { return 1 / (1 + exp(-x)); } // Sigmoide
float logit(float x) { return log(x / (1 - x)); } // funzione sigmoide inversa
float gaussian(float x, float mu, float var) { ... }
void WeightsStamp(string mode = "a") { ... }
  //m - stampa le medie dei pesi di ogni layer
  //a - stampa tutti i pesi della rete con alcuni parametri di apprendimento
  //w - stampa tutti i pesi con il riferimento riga colonna al target
  //fc - stampa le medie dei gruppi di pesi tra due layer

//funzione sigmoide
void sigLayer(int lyr) { ... }

//applica il bayes all'output di ogni neurone del dato layer
void bayesLayer(int lyr, bool absSum = false) { ... }

// esegue il reset del potenziale di tutti i neuroni della rete
void resetPotential() { ... }

// esegue il reset della sommatoria di ogni input in valore assoluto di ogni neurone
void resetAbsSumPotenzial() { ... }

// esegue il reset della sommatoria di ogni input in valore assoluto di ogni neurone
void resetAbsSumDelta() { ... }

// esegue il reset dell'errore retropropagato in ogni neurone
void resetBPerr() { ...}

//resetta il valore ID nei neuroni di una rete (necessario per modifiche strutturali)
void resetNeuronsID() { ... }

// salva su vettore i riferimenti numerici delle connesioni verso il layer specificato
vector<conMap> saveConsTowardsLyr(int Layer) { ... }

// ricarica i riferimenti numerici delle connesioni verso il layer specificato
void loadConsTowardsLyr(vector<conMap> con) { ... }

//elimina il contenuto l'oggetto dataset dell'oggetto rete
```

```
  void ClearDataset() { ... }

  //generazione di una serie storica del seno (DEBUG)
  void genTestDataset(int nExe, int nIn, int nOut, float step, int type, float offset) { ... }

  //esegue il settaggio della mappatura di output dell'oggetto
  void setNetMap(float max, float min) { ... }

  //restituisce il valore normalizzato dell'output della rete
  float reverseMap(int neur, float val) { ... }

  // crea un vettore di n elementi successivi e li disordina
  // creazione di una tabella di accesso casuale per un secondo vettore
  vector<u_int> casualVector(int in, int start = 0) { ... }

  //esegue il mescolamento degli elementi all'interno di un oggetto vector
  /*vector<T, A>*/
  template<typename T, typename A>
  void shackeVector(vector<T, A> const& vec) { ... }

  //ricalcola gli ID dei neuroni dopo il reset (da eseguire dopo delle modifiche strutturali)
  void refreshNeurIdx() { ... }

  //applica all'intero dataset un valore di offset
  void datasetOffset(float offset) { ... }
  ////////////////////////////////////////////////////////////////////////////////

  ////////////////////////ACCESSO A VARIABILI PRIVATE////////////////////////////////
  int numLayers() { ... }
  int numNeurons(int Layer) { ... }
  int numCon(int Layer, int Neuron) { ... }
  int numInCon(int Layer, int Neuron) { ... }
  int getConTargetLyr(int Layer, int Neuron, int Arc) { ... }
  int getConTargetCol(int Layer, int Neuron, int Arc) { ... }
  float getWeight(int Layer, int Neuron, int Arc) { ... }
  float getDeltaWeight(int Layer, int Neuron, int Arc) { ... }
  float getOutput(int Layer, int Neuron) { ... }
  float getBPerr(int Layer, int Neuron) { ... }
  ptNeuron getTarget(int Layer, int Neuron) { ... }
  ptNeuron getConTarget(int Layer, int Neuron, int Conn) { ... }
  int getOutConTargetID(ptNeuron base, ptNeuron target) { ... }
  ////////////////////////////////////////////////////////////////////////////////

  ////////////////////////WINDOWS HIGH PRECISION TIMING////////////////////////////
  BOOL WINAPI QueryPerformanceCounter(_Out_ LARGE_INTEGER *lpPerformanceCount);
  BOOL WINAPI QueryPerformanceFrequency(_Out_ LARGE_INTEGER *lpFrequency);
  inline long long PerformanceCounter() noexcept
  {...}
  inline long long PerformanceFrequency() noexcept
  {...}
  ////////////////////////////////////////////////////////////////////////////////
};
```

Listing 35: class Network

La classe MLP (multi-layer perceptron) figlia della classe Network completa il corredo della madre con tutta una serie di strumenti e variabili che supporta l'addestramento e l'utilizzo di questa tipologia di reti.

```

class MLP : public Network {

public:
  // tempo di esecuzione medio in millisecondi
  float NetPerformance = 0;

  //errore percentuale medio associato alla rete
  float NetErrPercent = 0;

  //costruttore
  MLP(string filename) :Network(filename) {};
```

```
13
14  ////////////////////FUNZIONI CREAZIONE RETE/////////////////////////////////////////
15  //permette la costruzione rapida di una rete MLP quadrata con dimensioni date
16  void qubeNet(int Nlayers, int Ncolumns, int input, int output, bool c, float initValue = 0.01f
        ↪ ) { ... }
17
18  //CREAZIONE RETE QUADRATA COMPLETAMENTE CONNESSA
19  //permette la costruzione di una rete MLP quadrata con dimensioni date in cui ogni neurone
20  //è connesso a tutti i neuroni di tutti gli strati successivi.
21  //(a parità di neuroni tale modello è più pesante ma tipicamente ha un aprrendimento più
        ↪ rapido)
22  void qubeNetFC(int Nlayers, int Ncolumns, int input, int output, bool c, float initValue = 0.
        ↪ 01f) { ... }
23
24  //CREAZIONE RETE CUSTOM
25  //permette la creazione di una rete MLP con la larghezza di ogni layer variabile
26  //indicando tali larghezze con un apposito vettore dato come argomento
27  void customNet(int Nlayers, vector<int> Ncolumns, float conFill) { ... }
28  ////////////////////////////////////////////////////////////////////////////////////
29
30  ////////////////////STIMOLAZIONE RETE/////////////////////////////////////////////////
31  //procedura di propagazione dell'informazione
32  //dati i riferimenti al vettore contenete gli input e ad un vettore di output esegue
33  //il processamento degli input e carica l'output derivante sul vettore indicato
34  void inputNet(vector<float> &input, vector<float> &output) { ... }
35
36  //esegue una propagazione dell'informazione salvando lo storico di propagazione degli input
37  //tale funzione viene utilizzata per l'apprendiment strutturale
38  void inputNetProfiler(vector<float> &input, vector<float> &output) { ... }
39  ////////////////////////////////////////////////////////////////////////////////////
40
41  ////////////////////////FUNZIONI DI ADDESTRAMENTO MLP/////////////////////////////////
42  //Algoritmo di addestramento Back-propagation su tutto il dataset caricato
43  void BP(int iter, float eps, float beta, float ErrPercent) { ... }
44
45  //esecuzione del backpropagation per un solo esempio
46  //specificando il particolare esempio
47  void oneBP(float eps, float beta, example e) { ... }
48  ////////////////////////////////////////////////////////////////////////////////////
49
50  ////////////////////////ALTRE FUNZIONI MLP/////////////////////////////////////////////
51  //inizializza i vettori di benchmark presenti nei neuroni per l'apprendimento strutturale
52  void initVectorsProfiler() { ... }
53
54  //resetta a zero tutti i vettori di profilazione esclusi layer input e output
55  void resetVectorsProfiler(bool inInfl, bool outInfl) { ... }
56
57  //stampa a schermo l'influenza degli input per ogni uscita
58  void stampInputInfluences(bool all = false) { ... }
59
60  //stampa a schermo l'influenza degli errori degli output per ogni ingresso
61  void stampOutputErrorPropagation(bool all = false) { ... }
62
63  //dal riferimento al nodo target e dall'indice del vettore restituisce il puntatore al neurone
        ↪  base
64  ptNeuron basePosfromtarget(ptNeuron target, int k) { ... }
65
66  //dato un neurone e l'indice di un suo arco restituisce l'indice di quella connessione all'
        ↪ interno del        //vettore influenceInput all'interno del neurone target
67  int idBaseConReftoTargetInfl(ptNeuron base, int arc) { ... }
68  ////////////////////////////////////////////////////////////////////////////////////
69
70 };
```

Listing 36: class MLP

3.5.3 altre classi non utilizzate nel progetto

Le seguenti classi sono state iniziate e verranno completate in futuro, La classe Hopfield supporta (ancora in parte) la creazione l'addetramento e l'utilizzo delle reti ricorrenti, ovvere reti con connessioni tra i neuroni che si retropropagano, ovvero che portano potenziali di uscita di neuroni nello strato output e hidden al tempo (t), ad altri neuroni nel tempo (t+1).
La classe structural learning è stata realizzata e testa, ma presenta notevoli inefficienze che non permettono di utilizzarla in questa forma su modelli di grandi dimensioni, tale classe comunque è stata realizzata per effettuare un addestramento che mira a modificare la struttura della rete durante l'addestramento eliminando le connessioni che risultano essere mediamente quelle che persistono nell'introduzione di errore rispetto alle altre.

```
class Hopfield : public Network { ... };

class StructuralLearning { ... };
```

Listing 37: altre classi

3.5.4 Classi e kernel per l'interfacciamento con la GPU

In questa sezione sono stati implementati i kernel della GPU, ovvero le funzioni scritte in C/CUDA, che vengono eseguite sulla scheda grafica. tali funzioni sono contraddistinte dalla keyword __global__, e richiedono come argomenti gli indirizzi di memoria corrispondenti ai dati precaricati sulla global memory della GPU. Il richiamo dei kernel dal codice C++ richiedono la sintassi speciale:
CUDAapplyWeightCorrections «<numOfBlocksA, ThxBlock »> (...);
si vede infatti la sequenza di caratteri «<A,B»> dove A è il numero di blocchi su cui deve essere eseguita la funzione mentre B rappresenta il numero di thread per ogni blocco.

```
////////////////////////////////////CUDA Kernels/////////////////////////////////////

//resetta il valore di una variabile all'interno della scheda grafica
__global__ void CUDAresetVar(float *val) {
  *val = 0;
}

//applica ad ogni arco della rete la correzione del peso
__global__ void CUDAapplyWeightCorrections(float eps, float *NeuronOut, float *BPerr, float *
    ↪ weights, int *ArcIn, int *ArcOut, int nArcs) {
  unsigned int i = (blockIdx.x * blockDim.x) + threadIdx.x;
  if (i < nArcs) {
    weights[i] += -eps * BPerr[ArcIn[i]] * NeuronOut[ArcOut[i]];
  }
}

//applica le correzioni hai bayes dei neuroni
__global__ void CUDAapplyBayesCorrections(float eps, float *BPerr, float *Bayes, int startN, int
    ↪  endN) {
  unsigned int i = startN + (blockIdx.x * blockDim.x) + threadIdx.x;
  if (i <= endN) {
    Bayes[i] += -eps * BPerr[i];
  }
}

//retropropaga l'errore nella rete
__global__ void CUDAPropagationErr(float *BPerr, float *weights, float *NeuronOut, int *ArcIn,
    ↪ int *ArcOut, int startA, int endA) {
  unsigned int i = startA + (blockIdx.x * blockDim.x) + threadIdx.x;

  //retropropago l'errore dai neuroni successivi
  if (i <= endA) {
    //BPerr[ArcOut[i]] += BPerr[ArcIn[i]] * weights[i];
    atomicAdd(&BPerr[ArcOut[i]], BPerr[ArcIn[i]] * weights[i]);
  }
}

//moltiplica l'errore delle uscite per la derivata puntuale della sigmoide
__global__ void CUDAoutDiff(float *BPerr, float *NeuronOut, int startN, int endN) {
  int i = startN + (blockIdx.x * blockDim.x) + threadIdx.x;
  if (i <= endN) {
    BPerr[i] *= NeuronOut[i] * (1 - NeuronOut[i]);
  }
```

```
}

//calcola l'errore dei neuroni dello strato output
__global__ void CUDAoutputErr(float *NeuronOut, int OutputRef, int numNeurons, int inputN, float *BPerr, float *examples, int exampleRef, float *mapMaxOut, float *mapMinOut, float *MeanErr) {

  unsigned int i = (OutputRef) + (blockIdx.x * blockDim.x) + threadIdx.x; //indice di scorrimento vettori: NeuronOut, BPerr,
  unsigned int e = (exampleRef + inputN) + (blockIdx.x * blockDim.x) + threadIdx.x; //indice di scorrimento vettori: examples
  unsigned int m = (blockIdx.x * blockDim.x) + threadIdx.x; // indice di scorrimento vettori: mapMaxOut, mapMinOut
  //if (i == 0) *MeanErr = 0;
  if (i < numNeurons) {

    float delta = mapMaxOut[m] - mapMinOut[m];
    BPerr[i] = (NeuronOut[i] - ((examples[e] - mapMinOut[m]) / delta)) * NeuronOut[i] * (1 - NeuronOut[i]); // formula valida solo per i neuroni di uscita
    //atomicAdd(MeanErr, (abs((((NeuronOut[i] * delta) + mapMinOut[m]) - examples[e]) / examples[e]))*100.0f);
    atomicAdd(MeanErr, abs((((NeuronOut[i] * delta) + mapMinOut[m]) - examples[e])/ examples[e]) * 100.0f);
    // calcolo l'errore percentuale sulla singola uscita e lo sommo
    //questo rappresenta uno dei punti più inefficienti dell'algoritmo,
    //la funzione AtomicAdd infatti blocca la parallizzazione dell'algoritmo per tutti
    //i thread che cercheranno di accedere a tale variabile riportando solo per questi
    //l'esecuzione ad un modello sequenziale mettendoli in attesa
  }
}

//resetta un dato vettore
__global__ void CUDAresetVector(float *vect, int size) {
  unsigned int i = (blockIdx.x * blockDim.x) + threadIdx.x;
  if (i < size) vect[i] = 0.0f;
}

//imposta i valori di output dei neuroni di input al valore dell'esempio del dataset
__global__ void CUDAsetInput(float *NeuronOut, int inputN, int exampleRef, float *example) {
  unsigned int i = (blockIdx.x * blockDim.x) + threadIdx.x;
  if (i < inputN)NeuronOut[i] = example[exampleRef + i];
}

//imposta i valori di output dei neuroni di input al valore specificato
__global__ void CUDAsetSingleInput(float *NeuronOut, int inputN, float *example) {
  unsigned int i = (blockIdx.x * blockDim.x) + threadIdx.x;
  if (i < inputN)NeuronOut[i] = example[i];
}

//applica la sigmoide ai potenziali dei neuroni in un dato intervallo (uno strato)
__global__ void CUDAsigLayer(float *NeuronOut, int start, int end) {
  unsigned int i = start + (blockIdx.x * blockDim.x) + threadIdx.x;
  if (i <= end) {
    NeuronOut[i] = 1 / (1 + expf(-NeuronOut[i]));
  }
}
//aggiunge all'output del neurone il contributo del bayes
__global__ void CUDAbayesInput(float *NeuronOut, float *Bayes, int start, int end) {
  unsigned int i = start + (blockIdx.x * blockDim.x) + threadIdx.x;
  if (i <= end) {
    NeuronOut[i] += Bayes[i];
  }
}

//propaga l'informazione dai neuroni dello strato input a quello di output
__global__ void CUDAlayerInput(float *weights, int *ArcIn, int *ArcOut, float *NeuronOut, int start, int end) {
  unsigned int i = start + (blockIdx.x * blockDim.x) + threadIdx.x;
  if (i <= end) {
```

```
    atomicAdd(&NeuronOut[ArcIn[i]], NeuronOut[ArcOut[i]] * weights[i]);
    //addizione bloccante non permette ad altri thread di sovrascrivere il falore
    // finche l'operazione non è completata
  }
}

////////////////////////////////////////////////////////////////////////////////
```

Listing 38: cuda kernels

Questa classe è l'interfaccia di collegamento con la GPU, la quale conserva i metodi di caricamento dei dati strutturali della rete, e gli esempi di addestramento sulla GPU.

```
//api di interfacciamento alla GPU
class CUDAcore {
public:
  //Device specs struct (contiene le specifiche della GPU)
  cudaDeviceProp prop;
  int GpuID = 0;

  //struttura contenente i puntatori alle aree di memoria conenenti i parametri della rete nella GPU
  struct devNetParams {
    float *weights = 0;
    int *ArcIn = 0;
    int *ArcOut = 0;
    float *examples = 0;
    float *NeuronOut = 0;
    float *Bayes = 0;
    float *BPerr = 0;
    float *mapMaxOut = 0;
    float *mapMinOut = 0;
    int *NeurInLyr = 0;
    int *priority = 0;
    float *MeanErr = 0;
    float *InputRT = 0;
  }gpuNetParams;

  vector<float> weights; //pesi della rete
  vector<int> ArcIn; //target dell'n-esimo arco
  vector<int> ArcOut; //base dell'n-esimo arco
  vector<float> NeuronOut; //vettore contenente l'output dei neuroni
  vector<float> Bayes; //vettore contenente i bayes dei neuroni
  vector<float> BPerr; // vettore contenete gli errori retropropagati
  vector<float> mapMaxOut; //vettore contenente il massimo valore degli output
  vector<float> mapMinOut; //vettore contenente il minimo valore degli output
  vector<int> priority; // vettore contenente i punti di sincronizazione dei thread
  vector<int> NeurInLyr; //vettore contenente gli indici dell'ultimo neurone di ogni layer
  vector<float>examples; //vettore degli esempi
  float MeanErr = 0; //veriabile contenente l'errore medio percentuale della rete
  int inputN, outputN; //passo di esecuzione elementi del vettore esempi

  CUDAcore(int nGpu) {
    GpuID = nGpu;
    checkCuda(cudaGetDeviceProperties(&prop, nGpu)); // carica lo struct cudaDeviceProp prop con le caratteristiche della GPU con indice 0
  }
```

Listing 39: classe di interfaccia alla GPU

La seguenti funzioni interne della classe di interfaccia alla GPU, rappresentano il metodo di conversione della struttura della rete dalla classe MLP alla classe CudaCore, e viceversa, tale conversione porta il modello da una struttura reticolare a puntatori in una struttura a vettori lineare.
Tale struttura divide i vari parametri della rete in vettori lineari, ogni tipo di elemento viene raggruppato in un unico vettore, mentre le connessioni precedentemente rappresentate da puntatori divengono vettori a loro volta che fungono da tabelle di accesso ordinate, scorrendo dal primo elemento all'ultimo si trovano tutti gli elementi su cui eseguire sequenzialmente le operazioni da svolgere per eseguire la propagazione. La complessità di tale operazione sta proprio nello svolgimento della struttura della rete per poter eseguire con efficienza la computazione sulla scheda grafica, la quale deve essere più "machine-friendly" possibile, data l'enorme quantità di operazioni che deve essere svolta per eseguire l'addestramento. Sono indicate anche altre funzioni di conversione per Hopfield e per il dataset il quale ha bisogno anch'esso di serializzazione per l'utilizzo nelle successive funzioni.

```
//copia del modello da MLP class a CudaCore class
void cudaNetCopyMLP(MLP *pt) {
  cout << "copying the net into CUDAcore.." << endl;
  weights.resize(pt->nArc);
  ArcIn.resize(pt->nArc);
  ArcOut.resize(pt->nArc);
  NeuronOut.resize(pt->nNeurons);
  Bayes.resize(pt->nNeurons);
  BPerr.resize(pt->nNeurons);
  priority.resize(pt->nLayers + 1);
  NeurInLyr.resize(pt->nLayers + 1);
  mapMaxOut.resize(pt->map.size());
  mapMinOut.resize(pt->map.size());
  inputN = pt->numNeurons(0);
  outputN = pt->numNeurons(pt->nLayers - 1);

  int NeuronIdx = 0;
  int ArcIdx = 0;
  vector<int> neurons(pt->nLayers);
  //carico il vettore di mappatura dell'output della rete
  for (int i = 0; i < pt->map.size(); i++) {
    mapMaxOut[i] = pt->map[i].maxValue;
    mapMinOut[i] = pt->map[i].minValue;
  }

  NeurInLyr[0] = -1; // setto il primo valore
  priority[0] = -1; // setto il primo valore

  //carico i parametri della rete
  for (int i = 0; i < pt->nLayers; i++) {

    for (int j = 0; j < pt->numNeurons(i); j++) {

      Bayes[NeuronIdx] = pt->getTarget(i, j)->bayes;

      for (int k = 0; k < pt->numCon(i, j); k++) {

        weights[ArcIdx] = pt->getTarget(i, j)->OutArcs[k].weight;
        ArcIn[ArcIdx] = pt->getTarget(i, j)->OutArcs[k].target->neurIdx;
        ArcOut[ArcIdx] = pt->getTarget(i, j)->neurIdx;
        ArcIdx++;
      }
      NeuronIdx++;
    }
    NeurInLyr[i + 1] = NeuronIdx - 1; // salvo l'indice dell'ultimo neurone del layer corrente
    priority[i + 1] = ArcIdx - 1; // salvo l'indice dell'ultimo arco del layer corrente
  }
}

//copia del modello da CudaCore class a MLP class
void cudaNetPasteMLP(MLP *pt) {
  int idx = 0;
  int Nidx = 0;
  for (int i = 0; i < pt->nLayers; i++) {
    for (int j = 0; j < pt->numNeurons(i); j++) {
      pt->getTarget(i, j)->bayes = Bayes[Nidx++];
      for (int k = 0; k < pt->numCon(i, j); k++) {
        pt->getTarget(i, j)->OutArcs[k].weight = weights[idx++];
      }
    }
  }
}

void cudaNetCopyHopfield(Hopfield* pt) { ... }

void cudaNetCopyExamples(MLP *pt) { ... }
```

Listing 40: classe di interfaccia alla GPU

3.5.5 metodi che lanciano i kernel sulla GPU

i seguenti metodi interagiscono direttamente con la GPU allocando memoria su di essa, caricando i dati, e lanciando i kernel che eseguono le effettive operazioni sulla GPU.

```
1
2  ////////////////////////////////CUDA Kernel functions//////////////////////////
3  //esegue le operazioni di allocamento memoria e preparazione al lancio del kernel di
       ↪ propagazione della rete
4  cudaError_t hostCUDAtrainingNet(float eps, int Niter, int ThxBlock) {
5    cout << "learning is started!" << endl;
6    //host variables
7    float *Cweights = &weights[0];
8    int *CArcIn = &ArcIn[0];
9    int *CArcOut = &ArcOut[0];
10   float *CNeuronOut = &NeuronOut[0];
11   float *CBayes = &Bayes[0];
12   float *CBPerr = &BPerr[0];
13   float *CmapMaxOut = &mapMaxOut[0];
14   float *CmapMinOut = &mapMinOut[0];
15   float *Cexamples = &examples[0];
16   int *CNeurInLyr = &NeurInLyr[0];
17   int *Cpriority = &priority[0];
18   float *CMeanErr = &MeanErr;
19
20   //device variables
21   float *dev_weights = 0;
22   int *dev_ArcIn = 0;
23   int *dev_ArcOut = 0;
24   float *dev_examples = 0;
25   float *dev_NeuronOut = 0;
26   float *dev_Bayes = 0;
27   float *dev_BPerr = 0;
28   float *dev_mapMaxOut = 0;
29   float *dev_mapMinOut = 0;
30   int *dev_NeurInLyr = 0;
31   int *dev_priority = 0;
32   float *dev_MeanErr = 0;
33
34   //int ThxBlock = 1024;
35
36   cudaError_t cudaStatus;
37
38   // Choose which GPU to run on, change this on a multi-GPU system.
39   cudaStatus = cudaSetDevice(GpuID);
40   if (cudaCheckStatus(cudaStatus) == true) goto Error;
41
42   // Allocate GPU buffers for vectors
43   cudaStatus = cudaMalloc((void**)&dev_weights, weights.size() * sizeof(float));
44   if (cudaCheckStatus(cudaStatus) == true) goto Error;
45   cudaStatus = cudaMalloc((void**)&dev_ArcIn, ArcIn.size() * sizeof(float));
46   if (cudaCheckStatus(cudaStatus) == true) goto Error;
47   cudaStatus = cudaMalloc((void**)&dev_ArcOut, ArcOut.size() * sizeof(float));
48   if (cudaCheckStatus(cudaStatus) == true) goto Error;
49   cudaStatus = cudaMalloc((void**)&dev_NeuronOut, NeuronOut.size() * sizeof(float));
50   if (cudaCheckStatus(cudaStatus) == true) goto Error;
51   cudaStatus = cudaMalloc((void**)&dev_Bayes, Bayes.size() * sizeof(float));
52   if (cudaCheckStatus(cudaStatus) == true) goto Error;
53   cudaStatus = cudaMalloc((void**)&dev_BPerr, BPerr.size() * sizeof(float));
54   if (cudaCheckStatus(cudaStatus) == true) goto Error;
55   cudaStatus = cudaMalloc((void**)&dev_mapMaxOut, mapMaxOut.size() * sizeof(float));
56   if (cudaCheckStatus(cudaStatus) == true) goto Error;
57   cudaStatus = cudaMalloc((void**)&dev_mapMinOut, mapMinOut.size() * sizeof(float));
58   if (cudaCheckStatus(cudaStatus) == true) goto Error;
59   cudaStatus = cudaMalloc((void**)&dev_examples, examples.size() * sizeof(float));
60   if (cudaCheckStatus(cudaStatus) == true) goto Error;
61   cudaStatus = cudaMalloc((void**)&dev_NeurInLyr, NeurInLyr.size() * sizeof(int));
62   if (cudaCheckStatus(cudaStatus) == true) goto Error;
63   cudaStatus = cudaMalloc((void**)&dev_priority, priority.size() * sizeof(int));
```

```
64      if (cudaCheckStatus(cudaStatus) == true) goto Error;
65      cudaStatus = cudaMalloc((void**)&dev_MeanErr, sizeof(float));
66      if (cudaCheckStatus(cudaStatus) == true) goto Error;
67
68
69      // Copy input vectors from host memory to GPU buffers.
70      cudaStatus = cudaMemcpy(dev_weights, Cweights, weights.size() * sizeof(float),
            ↪ cudaMemcpyHostToDevice);
71      if (cudaCheckStatus(cudaStatus) == true) goto Error;
72      cudaStatus = cudaMemcpy(dev_ArcIn, CArcIn, ArcIn.size() * sizeof(int),
            ↪ cudaMemcpyHostToDevice);
73      if (cudaCheckStatus(cudaStatus) == true) goto Error;
74      cudaStatus = cudaMemcpy(dev_ArcOut, CArcOut, ArcOut.size() * sizeof(int),
            ↪ cudaMemcpyHostToDevice);
75      if (cudaCheckStatus(cudaStatus) == true) goto Error;
76      cudaStatus = cudaMemcpy(dev_NeuronOut, CNeuronOut, NeuronOut.size() * sizeof(float),
            ↪ cudaMemcpyHostToDevice);
77      if (cudaCheckStatus(cudaStatus) == true) goto Error;
78      cudaStatus = cudaMemcpy(dev_Bayes, CBayes, Bayes.size() * sizeof(float),
            ↪ cudaMemcpyHostToDevice);
79      if (cudaCheckStatus(cudaStatus) == true) goto Error;
80      cudaStatus = cudaMemcpy(dev_BPerr, CBPerr, BPerr.size() * sizeof(float),
            ↪ cudaMemcpyHostToDevice);
81      if (cudaCheckStatus(cudaStatus) == true) goto Error;
82      cudaStatus = cudaMemcpy(dev_mapMaxOut, CmapMaxOut, mapMaxOut.size() * sizeof(float),
            ↪ cudaMemcpyHostToDevice);
83      if (cudaCheckStatus(cudaStatus) == true) goto Error;
84      cudaStatus = cudaMemcpy(dev_mapMinOut, CmapMinOut, mapMinOut.size() * sizeof(float),
            ↪ cudaMemcpyHostToDevice);
85      if (cudaCheckStatus(cudaStatus) == true) goto Error;
86      cudaStatus = cudaMemcpy(dev_examples, Cexamples, examples.size() * sizeof(float),
            ↪ cudaMemcpyHostToDevice);
87      if (cudaCheckStatus(cudaStatus) == true) goto Error;
88      cudaStatus = cudaMemcpy(dev_NeurInLyr, CNeurInLyr, NeurInLyr.size() * sizeof(int),
            ↪ cudaMemcpyHostToDevice);
89      if (cudaCheckStatus(cudaStatus) == true) goto Error;
90      cudaStatus = cudaMemcpy(dev_priority, Cpriority, priority.size() * sizeof(int),
            ↪ cudaMemcpyHostToDevice);
91      if (cudaCheckStatus(cudaStatus) == true) goto Error;
92      cudaStatus = cudaMemcpy(dev_MeanErr, CMeanErr, sizeof(float), cudaMemcpyHostToDevice);
93      if (cudaCheckStatus(cudaStatus) == true) goto Error;
94
95
96
97      ////////////////////lancio dei kernel all'interno della gpu///////////////
98      int startA = 0;
99      int endA = 0;
100     int startN = 0;
101     int endN = 0;
102     int numLayerArcs = 0;
103     int numLayerNeur = 0;
104     int numOfBlocksMax = 0;
105     int numOfBlocksA = 0;
106     int numOfBlocksN = 0;
107     int numOfBlocksOut = floorf(outputN / ThxBlock) + 1;
108     int exampleRef = 0;
109     int outputRef = NeuronOut.size() - outputN;
110     long long t0 = 0, t1 = 0;
111     long long t0in = 0, t1in = 0;
112     double elapsedMilliseconds = 0;
113     double elapsedInMilliseconds = 0;
114
115     for (int it = 0; it < Niter; it++) { //scorro le iterazioni
116
117       t0 = PerformanceCounter();
118
119       for (int t = 0; t < (examples.size() / (inputN + outputN)); t++) { //scorro gli esempi
120         //imposto il riferimento per l'esempio di input
121         exampleRef = t * (inputN + outputN);
```

```
122         t0in = PerformanceCounter();
123         //resetto il vettore contenente lo stato di attivazione dei neuroni
124         numOfBlocksA = (floorf(NeuronOut.size() / ThxBlock) + 1);
125         CUDAresetVector <<<numOfBlocksA, ThxBlock >>> (dev_NeuronOut, NeuronOut.size());
126 
127         //imposto i valori di input ai neuroni dello strato input
128         numOfBlocksA = (floorf(inputN / ThxBlock) + 1);
129         CUDAsetInput <<<numOfBlocksA, ThxBlock >>> (dev_NeuronOut, inputN, exampleRef,
                ↪ dev_examples);
130 
131         //propagazione dell'input nella rete
132 
133         startA = 0; // indice di partenza dei vettori archi
134         endA = 0; // ultimo indice dei vettori archi
135         startN = 0; // indice di partenza dei vettori neuroni
136         endN = 0; // ultimo indice dei vettori neuroni
137 
138         for (int i = 0; i < priority.size() - 1; i++) { //NB non viene applicata la sigmoide
                ↪ allo strato di input eventulmente correggi
139 
140           startA = priority[i] + 1;
141           endA = priority[i + 1];
142 
143           if (i < priority.size() - 2) {
144             startN = NeurInLyr[i + 1] + 1;
145             endN = NeurInLyr[i + 2];
146           }
147 
148           numLayerArcs = endA - startA + 1;
149           numLayerNeur = endN - startN + 1;
150 
151           numOfBlocksA = floorf(numLayerArcs / ThxBlock) + 1;
152           numOfBlocksN = floorf(numLayerNeur / ThxBlock) + 1;
153 
154           if (i < priority.size() - 2) {
155             //propago l'output dei neuroni al prossimo/i layer
156             CUDAlayerInput <<<numOfBlocksA, ThxBlock >>> (dev_weights, dev_ArcIn, dev_ArcOut,
                    ↪ dev_NeuronOut, startA, endA);
157              //applico il contributo dei bayes all output dei neuroni del layer corrente
158             CUDAbayesInput <<< numOfBlocksN, ThxBlock >>> (dev_NeuronOut, dev_Bayes, startN,
                    ↪ endN);
159             //applico la sigmoide allo stato di attivazione dei neuroni
160             CUDAsigLayer <<<numOfBlocksN, ThxBlock >>> (dev_NeuronOut, startN, endN);
161           }
162         }
163 
164         t1in = PerformanceCounter();
165         elapsedInMilliseconds += ((t1in - t0in) * 1000.0) / PerformanceFrequency();
166 
167 
168         //resetto il vettore contenente l'errore  dei neuroni
169         numOfBlocksN = (floorf(BPerr.size() / ThxBlock) + 1);
170         CUDAresetVector <<<numOfBlocksN, ThxBlock >>> (dev_BPerr, BPerr.size());
171 
172         CUDAresetVar <<<1, 1 >>> (dev_MeanErr);
173         CUDAoutputErr <<<numOfBlocksOut, ThxBlock >>> (dev_NeuronOut, outputRef, NeuronOut.size
                ↪ (), inputN, dev_BPerr, dev_examples, exampleRef, dev_mapMaxOut, dev_mapMinOut,
                ↪ dev_MeanErr);
174         cudaMemcpy(CMeanErr, dev_MeanErr, sizeof(float), cudaMemcpyDeviceToHost);
175 
176         MeanErr += *CMeanErr / outputN;
177 
178 
179         //retropropagazione dell'errore
180 
181         for (int i = priority.size() - 2; i > 1; i--) {
182 
183           startA = priority[i - 1] + 1;
184           endA = priority[i];
```

```
185           startN = NeurInLyr[i - 1] + 1;
186           endN = NeurInLyr[i];
187
188           numLayerArcs = endA - startA + 1;
189           numLayerNeur = endN - startN + 1;
190
191           numOfBlocksA = floorf(numLayerArcs / ThxBlock) + 1;
192           numOfBlocksN = floorf(numLayerNeur / ThxBlock) + 1;
193           //numOfBlocksMax = maxOf(numOfBlocksA, numOfBlocksN);
194
195           CUDAPropagationErr <<<numOfBlocksA, ThxBlock >>> (dev_BPerr, dev_weights,
                  ↪ dev_NeuronOut, dev_ArcIn, dev_ArcOut, startA, endA);
196           CUDAoutDiff <<<numOfBlocksN, ThxBlock >>> (dev_BPerr, dev_NeuronOut, startN, endN);
197           cudaStatus = cudaMemcpy(CBPerr, dev_BPerr, BPerr.size() * sizeof(float),
                  ↪ cudaMemcpyDeviceToHost);
198           if (cudaCheckStatus(cudaStatus) == true) goto Error;
199           copy(CBPerr, CBPerr + BPerr.size(), BPerr.begin());
200         }
201
202         //applico a ogni peso la sua correzione
203
204         startN = NeurInLyr[1] + 1; // la correzione dei bais va applicata dal primo layer
                ↪ nascosto in poi
205         endN = NeurInLyr[NeurInLyr.size() - 1];
206
207         numLayerNeur = endN - startN + 1;
208
209         numOfBlocksA = floorf(weights.size() / ThxBlock) + 1;
210         numOfBlocksN = floorf(numLayerNeur / ThxBlock) + 1;
211
212         CUDAapplyWeightCorrections <<<numOfBlocksA, ThxBlock >>> (eps, dev_NeuronOut, dev_BPerr,
                ↪  dev_weights, dev_ArcIn, dev_ArcOut, weights.size());
213         CUDAapplyBayesCorrections <<<numOfBlocksN, ThxBlock >>> (eps, dev_BPerr, dev_Bayes,
                ↪ startN, endN);
214       }
215
216       t1 = PerformanceCounter();
217       elapsedMilliseconds = ((t1 - t0) * 1000.0) / PerformanceFrequency(); // calcolo il tempo
              ↪ di esecuzione di una iterazione di addestramento (tutto il set)
218       MeanErr = MeanErr / (examples.size() / (inputN + outputN)); //calcolo l'errore percentuale
              ↪  medio sul dataset
219       elapsedInMilliseconds = elapsedInMilliseconds / (examples.size() / (inputN + outputN));
220       cout << "Iterazione: " << it << "  " << MeanErr << " %Err  " << "execution time:" <<
              ↪ elapsedMilliseconds << "ms" << endl;
221       cout << "mean InputTime: " << elapsedInMilliseconds << "ms" << endl;
222       printNetSpecs();
223       MeanErr = 0;
224     }
225
226     cudaStatus = cudaMemcpy(Cweights, dev_weights, weights.size() * sizeof(float),
            ↪ cudaMemcpyDeviceToHost);
227     if (cudaCheckStatus(cudaStatus) == true) goto Error;
228     copy(Cweights, Cweights + weights.size(), weights.begin());
229
230     cudaStatus = cudaMemcpy(CBayes, dev_Bayes, Bayes.size() * sizeof(float),
            ↪ cudaMemcpyDeviceToHost);
231     if (cudaCheckStatus(cudaStatus) == true) goto Error;
232     copy(CBayes, CBayes + Bayes.size(), Bayes.begin());
233     //checkpoint di errore (se la GPU richiama un qualunque errore ripare da qui)
234   Error:
235
236     //libero la memoria nella scheda grafica
237     cudaFree(dev_weights);
238     cudaFree(dev_ArcIn);
239     cudaFree(dev_ArcOut);
240     cudaFree(dev_NeuronOut);
241     cudaFree(dev_examples);
242     cudaFree(dev_BPerr);
243     cudaFree(dev_mapMaxOut);
```

```
    cudaFree(dev_mapMinOut);
    cudaFree(dev_priority);
    cudaFree(dev_NeurInLyr);

    //ritorno lo stato della GPU
    return cudaStatus;
}

//esegue il caricamento nella gpu dei parametri della rete
cudaError_t hostCUDAuploadNetParams() {

    cudaError_t cudaStatus;

    cudaStatus = cudaSetDevice(GpuID);
    if (cudaCheckStatus(cudaStatus) == true) goto Error;

    //host variables
    float *Cweights = &weights[0];
    int *CArcIn = &ArcIn[0];
    int *CArcOut = &ArcOut[0];
    float *CNeuronOut = &NeuronOut[0];
    float *CBayes = &Bayes[0];
    float *CBPerr = &BPerr[0];
    float *CmapMaxOut = &mapMaxOut[0];
    float *CmapMinOut = &mapMinOut[0];
    float *Cexamples = &examples[0];
    int *CNeurInLyr = &NeurInLyr[0];
    int *Cpriority = &priority[0];
    float *CMeanErr = &MeanErr;

    // Choose which GPU to run on, change this on a multi-GPU system.
    cudaStatus = cudaSetDevice(GpuID);
    if (cudaCheckStatus(cudaStatus) == true) goto Error;

    // Allocate GPU buffers for vectors
    cudaStatus = cudaMalloc((void**)&gpuNetParams.weights, weights.size() * sizeof(float));
    if (cudaCheckStatus(cudaStatus) == true) goto Error;
    cudaStatus = cudaMalloc((void**)&gpuNetParams.ArcIn, ArcIn.size() * sizeof(float));
    if (cudaCheckStatus(cudaStatus) == true) goto Error;
    cudaStatus = cudaMalloc((void**)&gpuNetParams.ArcOut, ArcOut.size() * sizeof(float));
    if (cudaCheckStatus(cudaStatus) == true) goto Error;
    cudaStatus = cudaMalloc((void**)&gpuNetParams.NeuronOut, NeuronOut.size() * sizeof(float));
    if (cudaCheckStatus(cudaStatus) == true) goto Error;
    cudaStatus = cudaMalloc((void**)&gpuNetParams.Bayes, Bayes.size() * sizeof(float));
    if (cudaCheckStatus(cudaStatus) == true) goto Error;
    cudaStatus = cudaMalloc((void**)&gpuNetParams.BPerr, BPerr.size() * sizeof(float));
    if (cudaCheckStatus(cudaStatus) == true) goto Error;
    cudaStatus = cudaMalloc((void**)&gpuNetParams.mapMaxOut, mapMaxOut.size() * sizeof(float));
    if (cudaCheckStatus(cudaStatus) == true) goto Error;
    cudaStatus = cudaMalloc((void**)&gpuNetParams.mapMinOut, mapMinOut.size() * sizeof(float));
    if (cudaCheckStatus(cudaStatus) == true) goto Error;
    cudaStatus = cudaMalloc((void**)&gpuNetParams.examples, examples.size() * sizeof(float));
    if (cudaCheckStatus(cudaStatus) == true) goto Error;
    cudaStatus = cudaMalloc((void**)&gpuNetParams.NeurInLyr, NeurInLyr.size() * sizeof(int));
    if (cudaCheckStatus(cudaStatus) == true) goto Error;
    cudaStatus = cudaMalloc((void**)&gpuNetParams.priority, priority.size() * sizeof(int));
    if (cudaCheckStatus(cudaStatus) == true) goto Error;
    cudaStatus = cudaMalloc((void**)&gpuNetParams.InputRT, inputN * sizeof(float));
    if (cudaCheckStatus(cudaStatus) == true) goto Error;
    cudaStatus = cudaMalloc((void**)&gpuNetParams.MeanErr, sizeof(float));
    if (cudaCheckStatus(cudaStatus) == true) goto Error;


    // Copy input vectors from host memory to GPU buffers.
    cudaStatus = cudaMemcpy(gpuNetParams.weights, Cweights, weights.size() * sizeof(float),
        ↪ cudaMemcpyHostToDevice);
    if (cudaCheckStatus(cudaStatus) == true) goto Error;
    cudaStatus = cudaMemcpy(gpuNetParams.ArcIn, CArcIn, ArcIn.size() * sizeof(int),
        ↪ cudaMemcpyHostToDevice);
```

```
311     if (cudaCheckStatus(cudaStatus) == true) goto Error;
312     cudaStatus = cudaMemcpy(gpuNetParams.ArcOut, CArcOut, ArcOut.size() * sizeof(int),
            ↪ cudaMemcpyHostToDevice);
313     if (cudaCheckStatus(cudaStatus) == true) goto Error;
314     cudaStatus = cudaMemcpy(gpuNetParams.NeuronOut, CNeuronOut, NeuronOut.size() * sizeof(float)
            ↪ , cudaMemcpyHostToDevice);
315     if (cudaCheckStatus(cudaStatus) == true) goto Error;
316     cudaStatus = cudaMemcpy(gpuNetParams.Bayes, CBayes, Bayes.size() * sizeof(float),
            ↪ cudaMemcpyHostToDevice);
317     if (cudaCheckStatus(cudaStatus) == true) goto Error;
318     cudaStatus = cudaMemcpy(gpuNetParams.BPerr, CBPerr, BPerr.size() * sizeof(float),
            ↪ cudaMemcpyHostToDevice);
319     if (cudaCheckStatus(cudaStatus) == true) goto Error;
320     cudaStatus = cudaMemcpy(gpuNetParams.mapMaxOut, CmapMaxOut, mapMaxOut.size() * sizeof(float)
            ↪ , cudaMemcpyHostToDevice);
321     if (cudaCheckStatus(cudaStatus) == true) goto Error;
322     cudaStatus = cudaMemcpy(gpuNetParams.mapMinOut, CmapMinOut, mapMinOut.size() * sizeof(float)
            ↪ , cudaMemcpyHostToDevice);
323     if (cudaCheckStatus(cudaStatus) == true) goto Error;
324     cudaStatus = cudaMemcpy(gpuNetParams.examples, Cexamples, examples.size() * sizeof(float),
            ↪ cudaMemcpyHostToDevice);
325     if (cudaCheckStatus(cudaStatus) == true) goto Error;
326     cudaStatus = cudaMemcpy(gpuNetParams.NeurInLyr, CNeurInLyr, NeurInLyr.size() * sizeof(int),
            ↪ cudaMemcpyHostToDevice);
327     if (cudaCheckStatus(cudaStatus) == true) goto Error;
328     cudaStatus = cudaMemcpy(gpuNetParams.priority, Cpriority, priority.size() * sizeof(int),
            ↪ cudaMemcpyHostToDevice);
329     if (cudaCheckStatus(cudaStatus) == true) goto Error;
330     cudaStatus = cudaMemcpy(gpuNetParams.MeanErr, CMeanErr, sizeof(float),
            ↪ cudaMemcpyHostToDevice);
331     if (cudaCheckStatus(cudaStatus) == true) goto Error;
332
333     if (false) {
334       Error:
335       //libero la memoria nella scheda grafica
336       cudaFree(gpuNetParams.weights);
337       cudaFree(gpuNetParams.ArcIn);
338       cudaFree(gpuNetParams.ArcOut);
339       cudaFree(gpuNetParams.NeuronOut);
340       cudaFree(gpuNetParams.examples);
341       cudaFree(gpuNetParams.BPerr);
342       cudaFree(gpuNetParams.mapMaxOut);
343       cudaFree(gpuNetParams.mapMinOut);
344       cudaFree(gpuNetParams.priority);
345       cudaFree(gpuNetParams.NeurInLyr);
346       cout << "ERRORE: libero la memoria della gpu. " << endl;
347     }
348
349     return cudaStatus;
350   }
351
352   //esegue il download dalla gpu dei parametri della rete
353   cudaError_t hostCUDAdownloadNetParams() {
354
355     cout << "downloading net params from gpu.." << endl;
356
357     float *Cweights = &weights[0];
358     float *CBayes = &Bayes[0];
359
360     cudaError_t cudaStatus;
361
362     cudaStatus = cudaSetDevice(GpuID);
363     if (cudaCheckStatus(cudaStatus) == true) goto Error;
364
365     cudaStatus = cudaMemcpy(Cweights, gpuNetParams.weights, weights.size() * sizeof(float),
            ↪ cudaMemcpyDeviceToHost);
366     if (cudaCheckStatus(cudaStatus) == true) goto Error;
367
368     cudaStatus = cudaMemcpy(CBayes, gpuNetParams.Bayes, Bayes.size() * sizeof(float),
```

```
        ↪ cudaMemcpyDeviceToHost);
369     if (cudaCheckStatus(cudaStatus) == true) goto Error;
370
371
372     if (false) {
373     Error:
374       //libero la memoria nella scheda grafica
375       cudaFree(gpuNetParams.weights);
376       cudaFree(gpuNetParams.ArcIn);
377       cudaFree(gpuNetParams.ArcOut);
378       cudaFree(gpuNetParams.NeuronOut);
379       cudaFree(gpuNetParams.examples);
380       cudaFree(gpuNetParams.BPerr);
381       cudaFree(gpuNetParams.mapMaxOut);
382       cudaFree(gpuNetParams.mapMinOut);
383       cudaFree(gpuNetParams.priority);
384       cudaFree(gpuNetParams.NeurInLyr);
385       cout << "ERRORE: libero la memoria della gpu. " << endl;
386     }
387
388     return cudaStatus;
389   }
390
391   //esegue l'input della rete gia addestrata prendendo in input l'esempio dato
392   cudaError_t hostCUDAInputNet(float *input, int ThxBlock) {
393     //inportante verificare che l'input abbia la stessa dimansione dell'input della rete
394
395     cudaError cudaStatus;
396
397     cudaStatus = cudaSetDevice(GpuID);
398     if (cudaCheckStatus(cudaStatus) == true) goto Error;
399
400     ////////////////////lancio dei kernel all'interno della gpu///////////////
401     //int ThxBlock = 1024;
402     int startA = 0;
403     int endA = 0;
404     int startN = 0;
405     int endN = 0;
406     int numLayerArcs = 0;
407     int numLayerNeur = 0;
408     int numOfBlocksMax = 0;
409     int numOfBlocksA = 0;
410     int numOfBlocksN = 0;
411     int numOfBlocksOut = floorf(outputN / ThxBlock) + 1;
412     int outputRef = NeuronOut.size() - outputN;
413     long long t0in = 0, t1in = 0;
414     double elapsedInMilliseconds = 0;
415
416     t0in = PerformanceCounter();
417     //resetto il vettore contenente lo stato di attivazione dei neuroni
418     numOfBlocksA = (floorf(NeuronOut.size() / ThxBlock) + 1);
419     CUDAresetVector <<<numOfBlocksA, ThxBlock >>> (gpuNetParams.NeuronOut, NeuronOut.size());
420
421     cudaStatus = cudaMemcpy(gpuNetParams.InputRT, input, inputN * sizeof(float),
        ↪ cudaMemcpyHostToDevice);
422     if (cudaCheckStatus(cudaStatus) == true) goto Error;
423
424     //imposto i valori di input ai neuroni dello strato input
425     numOfBlocksA = (floorf(inputN / ThxBlock) + 1);
426     CUDAsetSingleInput <<<numOfBlocksA, ThxBlock>>> (gpuNetParams.NeuronOut, inputN,
        ↪ gpuNetParams.InputRT);
427
428     //propagazione dell'input nella rete
429
430     startA = 0; // indice di partenza dei vettori archi
431     endA = 0; // ultimo indice dei vettori archi
432     startN = 0; // indice di partenza dei vettori neuroni
433     endN = 0; // ultimo indice dei vettori neuroni
434
```

```
435     for (int i = 0; i < priority.size() - 1; i++) { //NB non viene applicata la sigmoide allo
            ↪ strato di input eventulmente correggi
436
437       startA = priority[i] + 1;
438       endA = priority[i + 1];
439
440       if (i < priority.size() - 2) {
441         startN = NeurInLyr[i + 1] + 1;
442         endN = NeurInLyr[i + 2];
443       }
444
445       numLayerArcs = endA - startA + 1;
446       numLayerNeur = endN - startN + 1;
447
448       numOfBlocksA = floorf(numLayerArcs / ThxBlock) + 1;
449       numOfBlocksN = floorf(numLayerNeur / ThxBlock) + 1;
450
451       if (i < priority.size() - 2) {
452         CUDAlayerInput <<<numOfBlocksA, ThxBlock >>> (gpuNetParams.weights, gpuNetParams.ArcIn,
              ↪ gpuNetParams.ArcOut, gpuNetParams.NeuronOut, startA, endA); //propago l'output dei
              ↪  neuroni al prossimo/i layer
453         CUDAbayesInput <<<numOfBlocksN, ThxBlock >>> (gpuNetParams.NeuronOut, gpuNetParams.Bayes
              ↪ , startN, endN); //applico il contributo dei bayes all output dei neuroni del
              ↪ layer corrente
454         CUDAsigLayer <<<numOfBlocksN, ThxBlock >>> (gpuNetParams.NeuronOut, startN, endN); //
              ↪ applico la sigmoide allo stato di attivazione dei neuroni
455
456       }
457     }
458
459     float delta;
460     cout << "input time: " << elapsedInMilliseconds << " ms" << endl;
461
462     for (int on = 0; on < outputN; on++) {
463       delta = mapMaxOut[on] - mapMinOut[on];
464       cout << "Y" << on << ": " << (NeuronOut[NeuronOut.size() - outputN + on] * delta) +
            ↪ mapMinOut[on] << endl;
465     }
466     cout << endl;
467
468     ////////////////////////////////////////////////////////////////////////////////
469
470     if (false) {
471     Error:
472       //libero la memoria nella scheda grafica
473       cudaFree(gpuNetParams.weights);
474       cudaFree(gpuNetParams.ArcIn);
475       cudaFree(gpuNetParams.ArcOut);
476       cudaFree(gpuNetParams.NeuronOut);
477       cudaFree(gpuNetParams.examples);
478       cudaFree(gpuNetParams.BPerr);
479       cudaFree(gpuNetParams.mapMaxOut);
480       cudaFree(gpuNetParams.mapMinOut);
481       cudaFree(gpuNetParams.priority);
482       cudaFree(gpuNetParams.NeurInLyr);
483     }
484
485     return cudaStatus;
486   }
487   ////////////////////////////////////////////////////////////////////////////////
488
489   ////////////////////////////////CUDA UTILITY////////////////////////////////////////
490   //verifica la corretta esecuzione di un operazione
491   inline cudaError_t checkCuda(cudaError_t result){ ... }
492
493   //verifica la corretta esecuzione di un operazione restituendo un bool
494   bool cudaCheckStatus(cudaError_t cudaStatus) { ... }
495
496   //stampa a schermo le principali proprietà della scheda
```

```
497   void printDeviceSpecs() { ... }
498
499   //stampa i parametri della rete che vengono passati alla scheda
500   void printNetSpecs() { ... }
501
502   //calcola il peso del modello
503   float sizeOfModel(string mesureUnit = "B") { ... }
504
505   template<typename T, typename A>
506   float sizeOfVector(vector<T, A> const& vect,string mesureUnit = "B") { ... }
507   ////////////////////////////////////////////////////////////////////////////
508 };
```

Listing 41: classe di interfaccia alla GPU

3.6 Interfaccia grafica

L'interfaccia grafica dell'applicativo è stata realizzata in Unity, un popolare motore grafico per videogiochi e applicazioni android e ios. E' stato realizzato un rig (un manichino) attraverso blender con uno scheletro interfacciabile con Unity, in modo da poterne controllare i movimenti attraverso l'inserimento degli angoli dei vari "ossi" dello scheletro. Lo script dell'interfaccia grafica è stato scritto in C# e si basa su un client che richiede continuamente al server i dati di uscita della rete neurale.

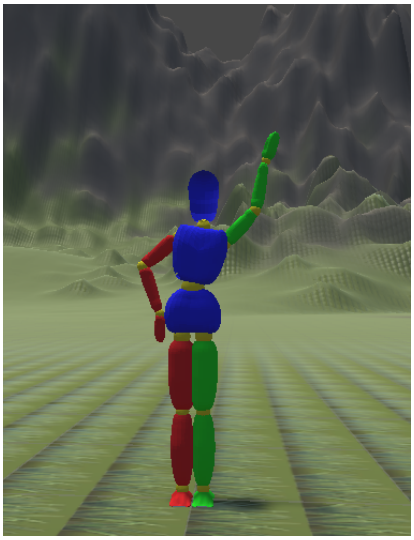


Figura 23: rig utilizzato per la renderizzazione

3.6.1 classe syncSocketServer

Questa parte illustra la classe utilizzata per far effettuare allo script in unity la richieta al server centrale dei dati elaborati.

3.6.1.1 librerie utilizzate

tale classe è stata implementata nella parte unity e ha richiesto l'importazione di UnityEngine, oltre alle librerie per la comunicazione socket e per la manipolazione dei byte ricevuti dal main server.

```
1 using System;
2 using System.Net;
3 using System.Net.Sockets;
4 using System.Runtime.InteropServices;
5 using System.Text;
6 using System.Collections;
7 using System.Collections.Generic;
8 using UnityEngine;
```

Listing 42: librerie class client C#

3.6.1.2 classe client C# per la comunicazione

In questa sezione è illustrato la classe che permette la comunicazione con il main server. Si noti come l'utilizzo della classe in unity richiede che essa sia figlia della classe MonoBehaviour, interna di unity.

```
1
2  public class SyncSocketClient : MonoBehaviour {
3      //struttura union per la conversione Bytes -> float
4      [StructLayout(LayoutKind.Explicit)]
5      public struct Ubones
6      {
7          [FieldOffset(0)] public float fBones;
8          [FieldOffset(0)] public byte strBones1;
9          [FieldOffset(1)] public byte strBones2;
10         [FieldOffset(2)] public byte strBones3;
11         [FieldOffset(3)] public byte strBones4;
12     }
13
14     //Oggetti socket
15     public IPHostEntry ipHostInfo;
16     public IPAddress ipAddress;
17     public IPEndPoint remoteEP;
18     public Socket sender;
19     private const int number_of_joints = 2;
20     private const int number_of_float = number_of_joints * 3;
21
22
23     // alloco la struttura di conversione
24     public Ubones[] bones = new Ubones[number_of_float]; //4 joints
25
26     // dichiaro il vettore di buffer
27     public byte[] bytes;
28
29     //settaggio parametri e inizzializzazione comunicazione
30     public SyncSocketClient(int Port){ ... }
31
32     //chiusura comunicazione
33     public void CloseConnection(){ ... }
34
35     // funzione per la ricezione e la conversione float
36     public void ReciveFloat(){ ... }
37
38   //funzione per la ricezione della stringa dal server
39     public string ReciveString(){ ... }
40
41     //trasmissione stringa
42     public void sendMessage(string msg){ ... }
43
44     //funzione di comunicazione, invia il codice di richiesta, riceve i byte li carica nell'
           ↪ union
45     public void TestComunication(){ ... }
46
47   //funzione per la reinizializzazione delle variabili bones
48     public void initBones(){ ... }
49     }
50 }
```

Listing 43: class client C#

3.6.2 classe di controllo del manichino

In questa sezione viene illustrata la classe interna al motore grafico realizzata per il caricamento dei dati ricevuti dal server nella struttura di dell'oggetto

3.6.2.1 librerie della classe controllo

```
1 using System.Collections.Generic;
2 using System.IO;
```

```
using System.Text;
using UnityEngine;
```

Listing 44: Librerie classe controllo C#

3.6.2.2 corpo della classe controllo

Anche questa classe è stata resa figlia di MonoBehaviour, il settaggio delle variabili viene effettuat dal metodo Start() richiamato da Unity all'avvio del programma, in questo caso però l'utilizzo della classe è richiamato attivamente da Unity attraverso il metodo Update() che viene richiamato una volta per ogni aggiornamento dello schermo.

```
public class ProvaControllo : MonoBehaviour {
  //dichiarazione della classe socketClient sincrono
  //e dell'oggetto di controllo del manichino
    private SyncSocketClient socket;
    public GameObject[] bones;

    private List<string> Tags = new List<string>();

    /*{ 4, 6, 3, 5, 8};*/ // indici degli oggetti da aggiornare nella lista GameObject
    public int[] neededObjects = new int[] {3, 5};

    //ref kinect{ 4, 5, 8, 9}; // indici degli oggetti da aggiornare nella struct Ubones della
        ↪ com socket
    public int[] neededBonesF = new int[] {0, 1};

  //vettori da tre per il passaggio degli angoli
    public Vector3 BraccioDx;
    public Vector3 avanBraccioDx;
    public Vector3 BraccioSx;
    public Vector3 avanBraccioSx;

    // Metodo di inizializzazzione della classe, richiamato all'avvio
    void Start () {

      //attivazione della comunicazione
        socket = new SyncSocketClient(9999);

        //inizializzazzione della struttura di conversione byte/float
        socket.initBones();

    //creazione della lista dei tag
        Tags.Add("Root"); //0
        Tags.Add("Bacino"); //1
        Tags.Add("schiena"); //2
        Tags.Add("BraccioDx"); //3
        Tags.Add("BraccioSx"); //4
        Tags.Add("AvanbraccioDx"); //5
        Tags.Add("AvanbraccioSx"); //6
        Tags.Add("testa"); //7
        Tags.Add("schiena"); //8

    //inizializzazzione dell'array di oggetti del gioco
        bones = new GameObject[10];

        //link degli oggetti creati alle ossa del manichino
        int i = 0;
        foreach (string Tag in Tags) {
            bones[i] = GameObject.FindGameObjectWithTag(Tag);
            i++;
        }
    }

  // Update is called once per frame
  void Update () {

    //richiesta al server centrsle dei nuovi dati e aggiornamento
    //della struttura interna alla classe socket
        socket.TestComunication();
```

```
58
59      //caricamento dei dati ricevuti sulla struttura unity del manichino
60          for (int i = 0; i < 2; i++)
61          {
62              bones[neededObjects[i]].transform.rotation = Quaternion.Euler(socket.bones[
                    ↪ neededBonesF[i]*3].fBones, socket.bones[(neededBonesF[i] * 3) + 1].fBones,
                    ↪ socket.bones[(neededBonesF[i] * 3) + 2].fBones);
63          }
64  }
```

Listing 45: class client C#

4 Conclusioni

Il progetto in fine si è concluso con esito negativo per quanto riguarda la buona generalizzazione del modello, principalmente a causa della cattiva qualità della bussola 3D, la quale anche durante i test sull'output non filtrato restituiva valori non coincidenti con la realtà. In compenso la libreria sviluppata per l'addestramento dei modelli ha dato buoni risultati in termini di ottimizzazione dell'addestramento dei modelli, incrementando su GPU la velocità di addestramento di ben 250 volte rispetto alla CPU, risultati soddisfacenti considerando che il programma è affetto fortemente da "race-condition", una problematica che affligge gli approcci al calcolo che si basano fortemente su multithreading. Tale problematica si presenta quando processi paralleli cercano di eseguire calcoli che mirano a modificare una stessa variabile contemporaneamente, e tale variabile è anche un input di queste funzioni, in particolare un processo prende tale variabile come input copiandola, un altro thread la modifica, e il primo la sovrascrive con il suo risultato non considerando la modifica fatta dal secondo. Per far fronte a tale problema è stato necessario utilizzare le funzioni atomiche che causano un blocco della variabile e mettono in attesa i thread che cercano di utilizzarla, causando una serializzazione delle operazioni. Un possibile altro approccio sarebbe stato effettuare queste operazioni con uno specifico ordine che mirava a fare più operazioni su più variabili evitando però la race-condition, con tale ottimizzazione si possono raggiungere velocità di computazione molto maggiori, in quanto un solo thread in attesa in un warp in cui tutti gli altri hanno concluso il proprio lavoro, li blocca tutti, e non permette allo stream-multiprocessor di lanciarne un altro. Abbiamo comunque deciso di non intraprendere questa strada in quanto richiedeva molto tempo e ingenti modifiche alla struttura del programma. In oltre con modelli a molti neuroni per layer tendeva ad attenuarsi in quanto si verificavano meno collisioni. Concludendo il problema non è stato risolto con i risultati sperati, ma la libreria realizzata è perfettamente funzionante e pronta per essere applicata per l'apprendimento di nuove task.