



UNIVERSITÀ POLITECNICA DELLE MARCHE

INGEGNERIA INFORMATICA E DELL'AUTOMAZIONE

Corso di Sistemi Operativi Dedicati

**Implementazione di una rete neurale per la
Human Activity Recognition su scheda STM con
sensore accelerometrico esterno**

Professore:

Dragoni Aldo Franco

Studenti:

Brutti Marco

Giannini Maria Cristina

Indice

1.	Introduzione	2
2.	STM X-CUBE-AI	3
2.1.	Core e funzionalità dell'ApplicationTemplate	3
2.2.	Guida all'uso dell'ApplicationTemplate	5
3.	Rete neurale per l'HAR a 6 classi	10
3.1.	Rete neurale	10
3.2.	Validation del modello utilizzato	11
3.3.	SystemPerformance del modello utilizzato	12
4.	Hardware	14
4.1.	Scheda STM32F429I-DISC1	14
4.2.	Sensore MPU-6050	14
4.2.1.	Protocollo di comunicazione I2C	18
5.	Codice dell'applicazione HAR	21
5.1.	STM32CubeMX	22
5.2.	Funzioni per la lettura e la scrittura dei registri con l'I2C	25
5.3.	Funzioni per il sensore	26
5.4.	Gestione dell'interrupt per l'overflow del buffer FIFO	32
5.5.	Funzione per l'acquisizione dei dati in input alla rete neurale	33
5.6.	Funzione per l'elaborazione dei dati in output dalla rete neurale	36
6.	Test dell'applicazione HAR	38
6.1.	Setup utilizzato per i test	38
6.2.	Test effettuati e risultati	41
7.	Conclusioni e sviluppi futuri	47
	Bibliografia	48

1. Introduzione

Il contesto in cui si colloca il progetto descritto in questo elaborato è quello dell'edge-AI applicato alle schede della STMicroelectronics. Nello specifico il progetto consiste nella realizzazione di un'applicazione che implementa una rete neurale per la Human Activity Recognition (HAR) su una scheda della STMicroelectronics utilizzando un sensore accelerometrico esterno ad essa. In particolare, l'applicazione permette di classificare in 6 classi (Sitting, Standing, Walking, Jogging, Upstairs e Downstairs) l'attività di un utente che ha il sensore nella tasca dei pantaloni utilizzando una rete neurale in esecuzione sulla scheda STM32F429I-DISC1.

Per la realizzazione del progetto si utilizza il tool X-CUBE-AI disponibile come estensione del software STM32CubeMX integrato all'interno dell'ambiente di sviluppo STM32CubeIDE. Il tool permette attraverso una semplice interfaccia grafica di convertire una rete neurale pre-addestrata in codice ottimizzato per l'esecuzione su scheda STM, generando una vera e propria libreria che l'utente può successivamente impiegare per utilizzare la rete all'interno del proprio progetto. Il tool X-CUBE-AI non si limita soltanto a convertire il modello originale della rete neurale in un modello in C, ma offre delle vere e proprie applicazioni generate automaticamente una volta selezionate dall'utente. In particolare, il progetto sfrutta l'applicazione ApplicationTemplate, la quale fornisce un template per l'utilizzo della rete sulla scheda STM.

Nella prima parte di questo elaborato è riportata una descrizione delle principali funzionalità del tool X-CUBE-AI, una breve guida all'uso dell'applicazione ApplicationTemplate ed è approfondita la rete utilizzata nel progetto.

Nella seconda parte viene descritto l'hardware impiegato, la scheda STM32F429I-DISC1 e il sensore accelerometrico MPU-6050, e il protocollo di comunicazione I2C utilizzato per l'interfacciamento scheda-sensore.

Infine, nell'ultima parte, è commentato il codice dell'applicazione e sono riportati i test effettuati e i risultati ottenuti.

2. STM X-CUBE-AI

La STMicroelectronics mette a disposizione un intero ecosistema di hardware e software per facilitare l'utilizzo di reti neurali sui suoi sistemi embedded, coprendo tutte le fasi: dall'acquisizione dei dati per il training fino alla classificazione in tempo reale di nuovi input (Figura 1).

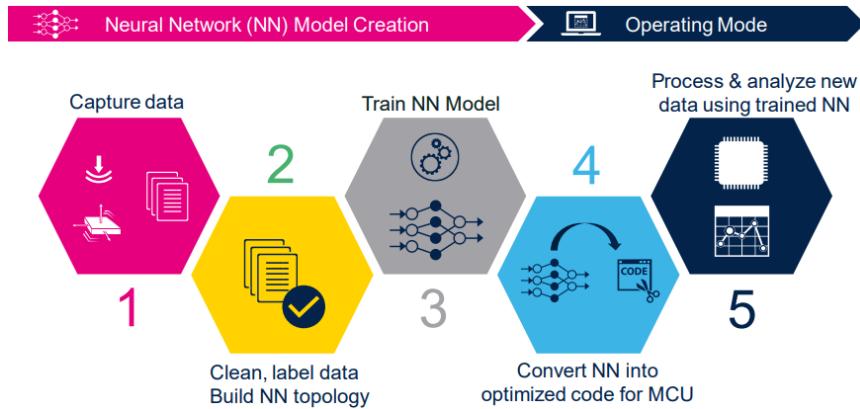


Figura 1: Fasi per l'utilizzo di una rete neurale su sistemi embedded

In questo capitolo viene approfondita la fase 4 e lo strumento X-CUBE-AI che permette di convertire una rete neurale pre-addestrata in codice ottimizzato per l'esecuzione su scheda STM. X-CUBE-AI è un tool fornito dalla STMicroelectronics che estende le funzionalità del software STM32CubeMX permettendo di mappare una rete neurale addestrata con una delle più popolari librerie (come Keras, Lasagne e Caffe) in una libreria che l'utente può facilmente importare ed utilizzare all'interno del proprio progetto. STM32CubeMX è uno strumento grafico che consente di generare automaticamente il codice per l'inizializzazione e la configurazione di microcontrollori e microprocessori STM32. Tale strumento è direttamente integrato nel software STM32CubeIDE [1] che è l'ambiente di sviluppo della STMicroelectronics utilizzato per lo sviluppo del presente progetto.

2.1. Core e funzionalità dell'ApplicationTemplate

In questo paragrafo è riportata una breve descrizione del core e delle funzionalità del tool; per maggiori approfondimenti si rimanda a [2].

Lo schema di funzionamento generale del tool X-CUBE-AI è visibile in Figura 2.

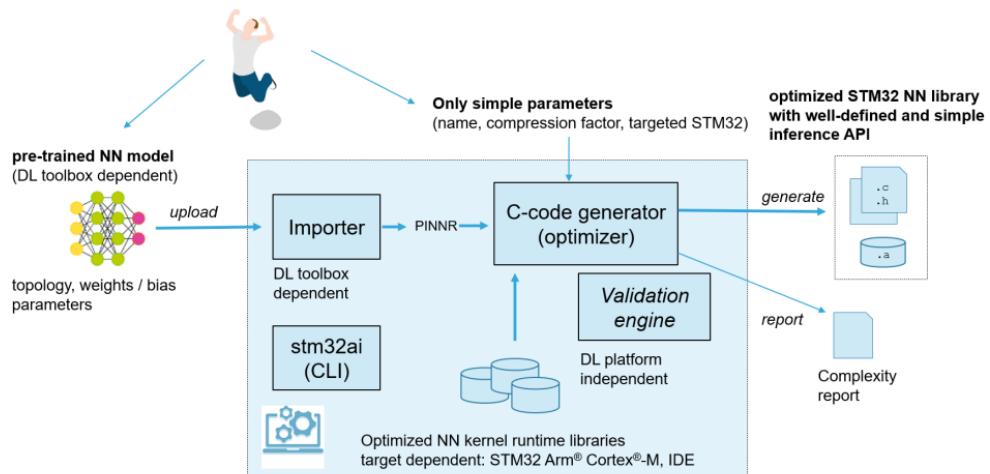


Figura 2: Funzionamento di X-CUBE-AI

Il tool permette attraverso una semplice interfaccia grafica di generare la libreria che implementa in C la rete neurale che si desidera eseguire su una scheda STM. Gli input che l'utente deve fornire al tool sono:

- il modello pre-addestrato della rete neurale;
- il nome del modello in C della rete neurale generato dal tool;
- il fattore di compressione che permette di ridurre le dimensione del modello;
- la versione della scheda.

Per quanto riguarda il modello pre-addestrato della rete neurale, il tool accetta soltanto alcune tipologie di modello. In particolare, attualmente, sono supportati modelli generati con i seguenti toolbox di Deep Learning: Keras, Lasagne, Caffe e ConvNetJS. Inoltre, per ogni toolbox, X-CUBE-AI è in grado di elaborare solo un sottoinsieme di layer e dei loro parametri. La lista di layer supportati per ogni toolbox si può trovare nel capitolo 12 del manuale di X-CUBE-AI riportato in [3].

In Figura 3 si possono osservare le principali proprietà che deve rispettare la rete neurale pre-addestrata in ingresso al tool e le caratteristiche che possiede la libreria generata dalla conversione. In particolare, il modello della rete neurale deve avere un tensore in ingresso e un tensore in uscita, entrambi di dimensione 4 (n° campioni, altezza, larghezza, n° canali), mentre il modello in C generato dalla conversione è rappresentato con valori a virgola mobile a 32 bit.

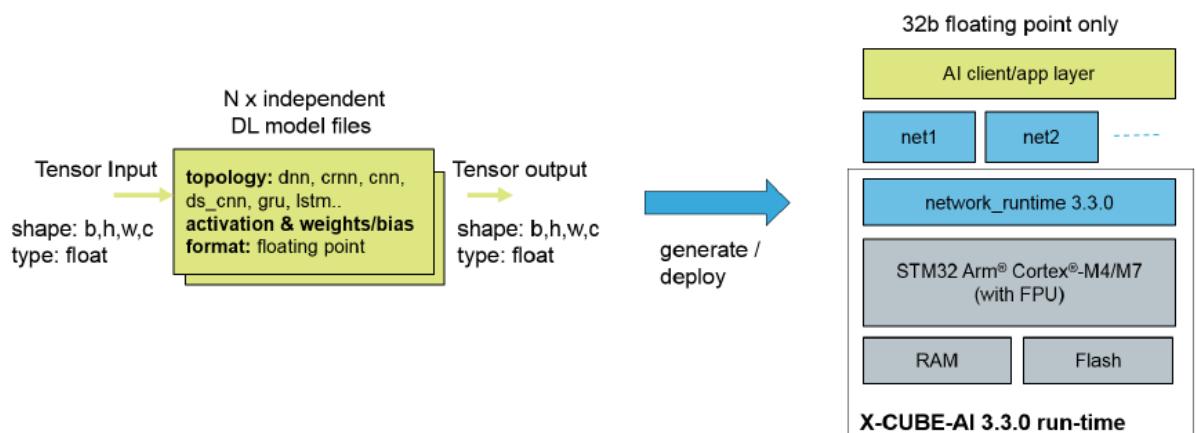


Figura 3: Conversione della rete neurale nel modello in C

Questa conversione è effettuata dal C Code Generator di Figura 2 che deve non soltanto generare la libreria in C, ma anche ottimizzarla per l'utilizzo della rete su dispositivi con risorse hardware limitate, come le schede STM. La problematica principale è rappresentata dall'occupazione di memoria richiesta da una rete neurale sia per la memorizzazione della rete stessa (Flash) sia per la sua esecuzione (RAM), che è normalmente superiore a quella disponibile su un sistema embedded. Per ovviare a tale problematica il C Code Generator applica i seguenti algoritmi di compressione ed ottimizzazione:

- compressione dei parametri della rete;
- riduzione del numero di layer;
- ottimizzazione dell'utilizzo della RAM.

Questi algoritmi sono basati su un approccio che non prevede l'utilizzo del dataset, in quanto non si preoccupano di preservare l'accuratezza del modello iniziale.

Da evidenziare è la compressione dei parametri che permette di ridurre la dimensione della rete limitando di conseguenza l'occupazione della memoria Flash, ma riducendo l'accuratezza del modello iniziale. In particolare, X-CUBE-AI offre la possibilità di scegliere fra tre diversi fattori di compressione (none, 4, 8). Il tool mette a disposizione dell'utente un'applicazione di tipo Validation che permette di valutare la perdita di accuratezza e l'errore generato dalla compressione.

Il tool X-CUBE-AI non si limita a convertire la rete neurale in una libreria in C ottimizzata come descritto in precedenza, ma offre delle vere e proprie applicazioni generate automaticamente una volta selezionate dall'utente. Nello specifico le applicazioni disponibili sono le seguenti:

- SystemPerformance: permette di calcolare il tempo medio di esecuzione di un'inferenza quando la rete è in esecuzione sulla scheda STM;
- Validation: permette di valutare la perdita di accuratezza del modello in C rispetto al modello originale;
- ApplicationTemplate: fornisce un template per un facile utilizzo della rete sulla scheda STM.

Nel seguente elaborato si è utilizzato l'ApplicationTemplate come punto di partenza per sviluppare l'applicazione HAR e nel paragrafo successivo ne è riportata una guida all'uso.

2.2. Guida all'uso dell'ApplicationTemplate

Per generare il modello in C della rete neurale e utilizzare l'applicazione ApplicationTemplate sono stati seguiti i seguenti passaggi:

1. scaricare ed installare l'ambiente di sviluppo STM32CubeIDE come riportato in [1];
2. creare un nuovo progetto “STM32 Project” dal menu File -> New -> STM32 Project;
3. selezionare la scheda sulla quale si intende utilizzare la rete neurale dalla scheda Board Selector, così come mostrato in Figura 4;

*	Overview	X	Commercial Part No	Type	X	Marketing Status	X	Unit Price (US\$)	X	Mounted Device	X
★		X	STM32F429I-DISC1	Discovery Kit	Active		X	29.9	X	STM32F429ZITx	X

Figura 4: Schermata di selezione della scheda STM

4. nominare il file del progetto e terminare la procedura di creazione del progetto senza inizializzare le periferiche;
5. aprire il file “NomeProgetto.ioc” e da Software Packs -> Select Components scaricare il tool X-CUBE-AI;
6. dallo stesso menu abilitare il Core del tool da STMicroelectronics.X-CUBE-AI -> Artificial Intelligence X-CUBE-AI -> Core e selezionare l'applicazione ApplicationTemplate da STMicroelectronics.X-CUBE-AI -> Device Application -> Application, come mostrato in Figura 5;

Packs				
RoweBots.I-CUBE-UNISONRTOS	Status	Version	Selection	-
> RoweBots.I-CUBE-UNISONRTOS	5.5.0-4		Install	
> SEGGER.I-CUBE-embOS	1.2.0		Install	
STMicroelectronics.X-CUBE-AI	7.0.0			
Artificial Intelligence X-CUBE-AI	7.0.0			
Core			<input checked="" type="checkbox"/>	
Device Application	7.0.0			
Application			ApplicationTemplate	
STMicroelectronics.X-CUBE-ALGOBUILD	1.2.1		Install	

Figura 5: Schermata di selezione dei tool di sviluppo forniti da STMicroelectronics

7. per accedere all'interfaccia grafica del tool, visibile in Figura 6, aprire il pacchetto STMicroelectronics.X-CUBE-AI dal menu Software Packs;

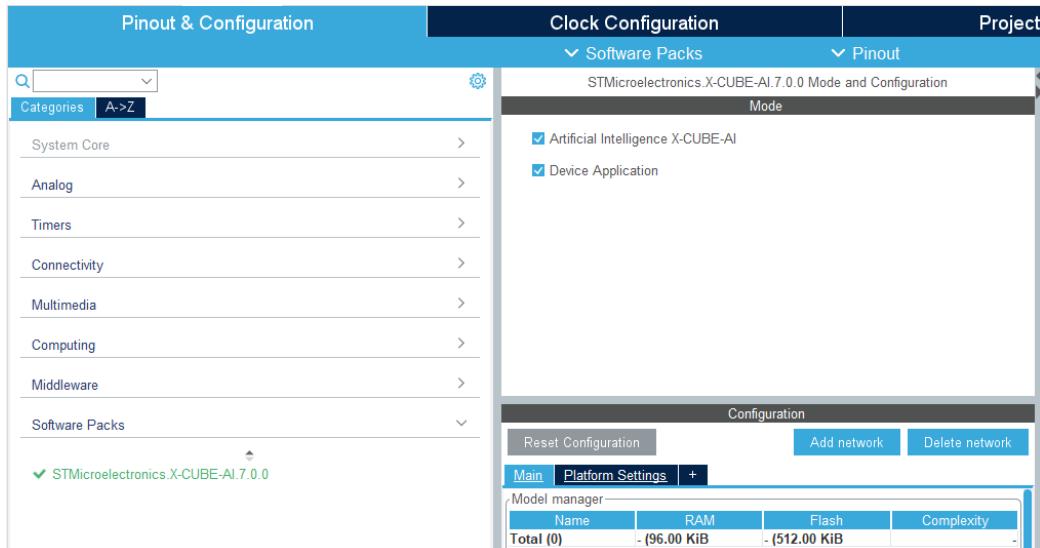


Figura 6: Interfaccia grafica del tool X-CUBE-AI

8. aggiungere una nuova rete tramite il bottone Add network nella scheda Configuration;
9. dalla nuova scheda creata dal tool, rinominare la rete appena creata e caricarne il modello, come mostrato in Figura 7;

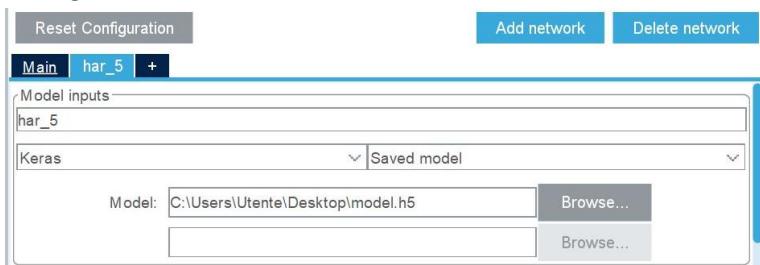


Figura 7: Interfaccia per la creazione di una nuova rete

10. analizzare il modello caricato per verificare se la quantità di memoria richiesta è compatibile con quella disponibile sulla scheda. In caso di esito negativo dell'analisi, come riportato in Figura 8, verificare se il problema è rappresentato dai limiti della Flash o dai limiti della RAM della scheda correntemente utilizzata. Nel caso in cui la scheda non abbia sufficiente RAM è necessario scegliere una scheda con più RAM, mentre se il problema è dato dalla Flash, come in Figura 9, sarà sufficiente selezionare un fattore di compressione maggiore e procedere nuovamente con l'analisi;

```
Analyzing Network
-----
model/c-model: macc=875,232/875,232  weights=2,955,800/2,955,800  activations=--/24,576 io=--/1,104
Complexity report per layer - macc=875,232 weights=2,955,800 act=24,576 ram_io=1,104
-----
id    name      c_macc          c_rom          c_id
-----
0    conv2d_1    |||      15.6% |      0.1% [0]
4    dense_1     ||||||||| 82.4% ||||||||| 97.6% [1]
4    dense_1_nl   |      0.0% |      0.0% [2]
5    dense_2     |      1.9% |      2.2% [3]
5    dense_2_nl   |      0.0% |      0.0% [4]
6    dense_3     |      0.1% |      0.1% [5]
6    dense_3_nl   |      0.0% |      0.0% [6]
Creating txt report file C:\Users\Utente\stm32cubemx\har_5_analyze_report.txt
elapsed time (analyze): 0.722s
Analyze complete on AI model
Required Ram or Flash size for har_5 is bigger than available Ram or Flash
```

OK

Figura 8: Report negativo dell'analisi della rete

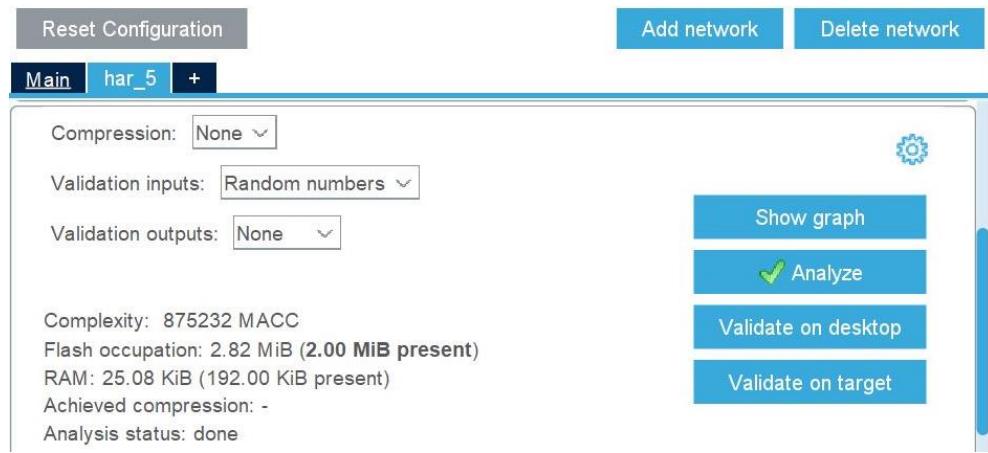


Figura 9: Risultato dell'analisi della rete

11. salvare il progetto e generare il codice. Con questa operazione il tool genera in automatico il modello in C della rete caricata, la libreria e l'applicazione ApplicationTemplate;
12. modificare l'ApplicationTemplate in base all'applicazione che si vuole sviluppare.

In Figura 10 è riportato il main dell'ApplicationTemplate generato automaticamente dal tool, che si trova in NOME_PROGETTO\Core\Src\main.c. Come si può notare, nel main sono chiamate una serie di funzioni per l'inizializzazione delle periferiche e della rete, a cui segue un ciclo infinito per l'esecuzione della rete neurale.

```

67 int main(void)
68 {
69     /* USER CODE BEGIN 1 */
70
71     /* USER CODE END 1 */
72
73     /* MCU Configuration-----*/
74
75     /* Reset of all peripherals, Initializes the Flash interface and the Systick. */
76     HAL_Init();
77
78     /* USER CODE BEGIN Init */
79
80     /* USER CODE END Init */
81
82     /* Configure the system clock */
83     SystemClock_Config();
84
85     /* USER CODE BEGIN SysInit */
86
87     /* USER CODE END SysInit */
88
89     /* Initialize all configured peripherals */
90     MX_GPIO_Init();
91     MX_CRC_Init();
92     MX_X_CUBE_AI_Init();
93
94     /* USER CODE BEGIN 2 */
95
96     /* USER CODE END 2 */
97
98     /* Infinite loop */
99     /* USER CODE BEGIN WHILE */
100    while (1)
101    {
102        /* USER CODE END WHILE */
103
104        MX_X_CUBE_AI_Process();
105        /* USER CODE BEGIN 3 */
106    }
107    /* USER CODE END 3 */
108 }
```

Figura 10: Main dell'ApplicationTemplate

In particolare, in Figura 11 è riportato il codice della funzione MX_X_CUBE_AI_Process() relativo alla classificazione della rete neurale che si trova in NOME_PROGETTO\X-CUBE-AI\App\app_x-cube-ai.c.

```

198 void MX_X_CUBE_AI_Process(void)
199 {
200     /* USER CODE BEGIN 6 */
201     int res = -1;
202     uint8_t *in_data = NULL;
203     uint8_t *out_data = NULL;
204
205     printf("TEMPLATE - run - main loop\r\n");
206
207     if (har) {
208
209         if ((har_info.n_inputs != 1) || (har_info.n_outputs != 1)) {
210             ai_error err = {AI_ERROR_INVALID_PARAM, AI_ERROR_CODE_OUT_OF_RANGE};
211             ai_log_err(err, "template code should be updated\r\n to support a model with multiple IO");
212             return;
213         }
214
215         /* 1 - Set the I/O data buffer */
216
217 #if AI_HAR_INPUTS_IN_ACTIVATIONS
218     in_data = har_info.inputs[0].data;
219 #else
220     in_data = in_data_s;
221 #endif
222
223 #if AI_HAR_OUTPUTS_IN_ACTIVATIONS
224     out_data = har_info.outputs[0].data;
225 #else
226     out_data = out_data_s;
227 #endif
228
229     if ((!in_data) || (!out_data)) {
230         printf("TEMPLATE - I/O buffers are invalid\r\n");
231         return;
232     }
233
234     /* 2 - main loop */
235     do {
236
237         /* 1 - acquire and pre-process input data */
238         res = acquire_and_process_data(in_data);
239         /* 2 - process the data - call inference engine */
240         if (res == 0)
241             res = ai_run(in_data, out_data);
242         /* 3- post-process the predictions */
243         if (res == 0)
244             res = post_process(out_data);
245     } while (res==0);
246
247     if (res) {
248         ai_error err = {AI_ERROR_INVALID_STATE, AI_ERROR_CODE_NETWORK};
249         ai_log_err(err, "Process has FAILED");
250     }
251     /* USER CODE END 6 */
252 }

```

Figura 11: Codice della funzione MX_X_CUBE_AI_Process()

Sono da notare le righe di codice che vanno da 234 a 244, le quali rappresentano il ciclo infinito di esecuzione della rete neurale:

- `acquire_and_process_data()`: acquisisce e pre-processa i dati di input che vengono assegnati alla variabile `in_data`;
- `ai_run()`: esegue la classificazione dell'ingresso `in_data` e restituisce le uscite della rete nella variabile `out_data`;
- `post_process()`: post-processa i dati di output per determinare la classe di appartenenza dell'input.

In Figura 12 sono riportate le funzioni sopracitate, si fa notare che `acquire_and_process_data()` e `post_process()` non sono realmente implementate ma vanno modificate in base all'applicazione.

```

154 static int ai_run(void *data_in, void *data_out)
155 {
156     ai_i32 batch;
157
158     ai_buffer *ai_input = har_info.inputs;
159     ai_buffer *ai_output = har_info.outputs;
160
161     ai_input[0].data = AI_HANDLE_PTR(data_in);
162     ai_output[0].data = AI_HANDLE_PTR(data_out);
163
164     batch = ai_har_run(har, ai_input, ai_output);
165     if (batch != 1) {
166         ai_log_err(ai_har_get_error(har),
167                    "ai_har_run");
168         return -1;
169     }
170
171     return 0;
172 }
173
174 /* USER CODE BEGIN 2 */
175 int acquire_and_process_data(void * data)
176 {
177     return 0;
178 }
179
180 int post_process(void * data)
181 {
182     return 0;
183 }

```

Figura 12: Codici delle funzioni del ciclo infinito

Nel Capitolo 5 viene descritto in dettaglio come l'ApplicationTemplate è stato modificato per ottenere l'applicazione HAR sviluppata nel seguente progetto.

3. Rete neurale per l'HAR a 6 classi

In questo capitolo viene descritta la rete neurale pre-addestrata utilizzata per l'HAR. In particolare, vengono approfondite le caratteristiche di tale modello sia in termini di topologia che di performance nella classificazione dell'attività.

3.1. Rete neurale

La rete neurale utilizzata per l'HAR classifica l'attività dell'utente a partire dalle misure di un accelerometro triassiale nelle seguenti 6 classi: Sitting, Standing, Walking, Jogging, Downstairs e Upstairs. Il modello pre-addestrato di tale rete è reperibile al repository GitHub, riportato in [4], il quale contiene i seguenti file:

- HAR.py: codice Python per l'addestramento della rete, la cui esecuzione genera il modello addestrato della rete (model.h5) e il test set (testData.npy e groundTruth.npy);
- actitracker_raw.txt: file di testo contenente l'intero dataset;
- model.h5: modello pre-addestrato della rete neurale;
- evaluate_model.py: codice Python per valutare le performance della rete pre-addestrata sul set test;
- testData.npy: file di dati NumPy contenente gli ingressi del test set;
- groundTruth.npy: file di dati NumPy contenente le uscite corrette del test set.

Per questo progetto non è stato eseguito il file HAR.py ma sono stati utilizzati direttamente i file model.h5, testData.npy e groundTruth.npy già presenti nel repository GitHub sopracitato. In particolare, il file model.h5 è utilizzato come ingresso del tool X-CUBE-AI, mentre i file relativi al test set sono impiegati per valutare la perdita di accuratezza della rete mediante l'applicazione Validation. Il dataset utilizzato per l'addestramento di tale rete, ovvero il file actitracker_raw.txt sopracitato, contiene le misure di un sensore accelerometrico triassiale di uno smartphone, con frequenza di campionamento pari a 20Hz, ottenute facendo svolgere le varie attività a 36 utenti con lo smartphone nella tasca dei pantaloni.

In Figura 13 sono riportati un esempio degli andamenti delle accelerazioni lungo gli assi X, Y e Z che si ottengono con le diverse attività.

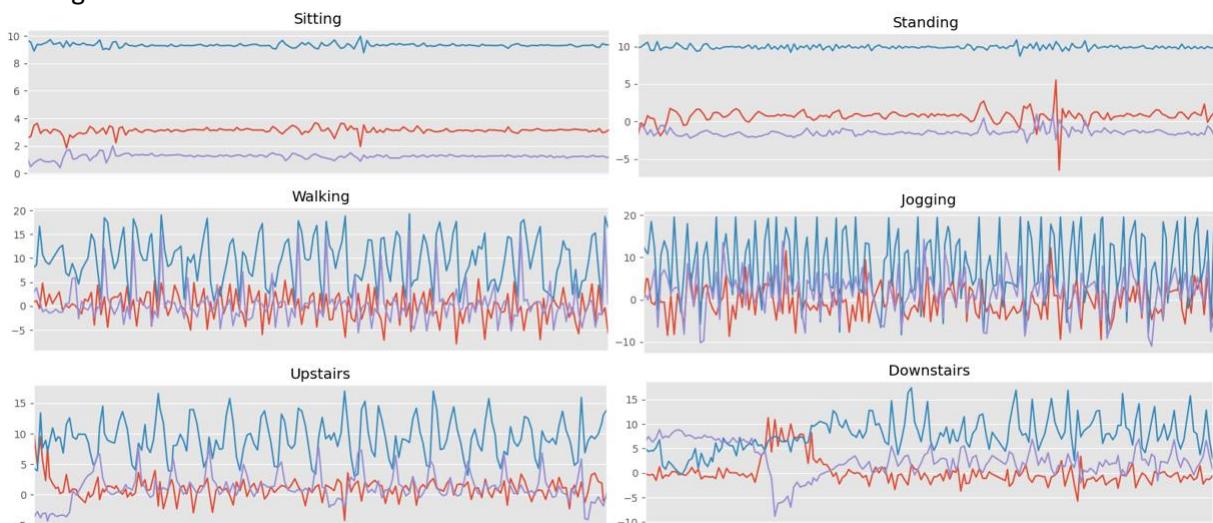


Figura 13: Andamenti delle accelerazioni lungo gli assi x, y e z per ognuna delle 6 classi

In Figura 14 è riportata la distribuzione totale del dataset caratterizzato da:

- 424400 misurazioni per la classe Walking;
- 342177 misurazioni per la classe Jogging;
- 122869 misurazioni per la classe Upstairs;
- 100427 misurazioni per la classe Downstairs;
- 59939 misurazioni per la classe Sitting;
- 48395 misurazioni per la classe Standing.

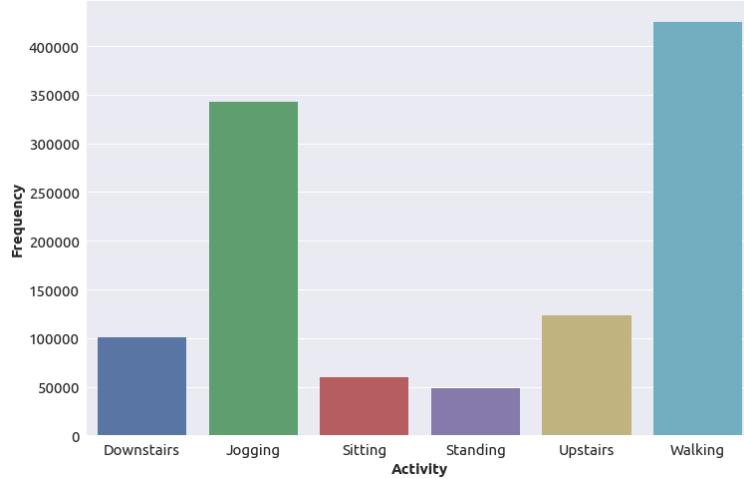


Figura 14: Distribuzione del dataset fra le 6 classi

Il dataset non viene utilizzato direttamente per addestrare la rete ma subisce un pre-processamento: i dati vengono suddivisi in finestre da 90 campioni l'una, al fine di avere un tensore di ingresso della rete neurale avente dimensioni ridotte rispetto a quelle dell'intera acquisizione di ogni classe. Inoltre, viene introdotto un overlapping delle finestre ottenute pari al 50% in modo da aumentare la dimensione del dataset.

In Figura 15 è infine riportata in dettaglio la struttura della rete neurale addestrata, specificando i layer che la compongono e i relativi parametri. Si fa notare che si tratta di una rete neurale convoluzionale che prende in ingresso una matrice 90x3 (90 acquisizioni per ogni asse X, Y e Z) e restituisce in uscita un vettore di 6 elementi; ogni elemento rappresenta la probabilità che l'ingresso appartenga a quella specifica classe.

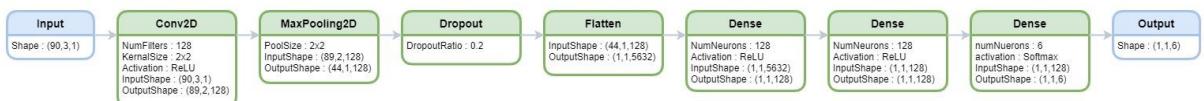


Figura 15: Struttura della rete neurale per la classificazione su 6 classi

3.2. Validation del modello utilizzato

Il primo step per implementare la rete neurale, precedentemente descritta, sulla scheda STM32F429I-DISC1, o in generale una qualsiasi rete neurale su una scheda STM32, è quello di analizzare se la quantità di memoria richiesta è compatibile con quella disponibile sulla scheda e di valutare la perdita di accuratezza causata dalla compressione che il tool X-CUBE-AI esegue. Questo step è ampiamente descritto in [2]; in questo paragrafo ci si limita a riportare i risultati ottenuti eseguendo la Validation con la rete e la scheda utilizzate per il progetto.

In Tabella 1 è riportata l'occupazione di memoria RAM e Flash richiesta dalla rete presa in esame per l'HAR al variare del fattore di compressione, che può assumere valore nullo, 4 o 8. Si fa notare che, dati i valori della memoria RAM e della memoria Flash della scheda utilizzata, rispettivamente pari a 256KB

e 2MB e descritti successivamente al Paragrafo 4.1, non è possibile utilizzare la rete senza compressione; questo aspetto è evidenziato in arancione nella Tabella 1.

		Compressione		
		None	4	8
Occupazione RAM	25.08KB	25.08KB	25.08KB	
	2.82MB	775.52KB	366.65KB	

Tabella 1: Occupazione di memoria RAM e Flash della rete

Sulla base di questo risultato, è necessario valutare anche la perdita di accuratezza della rete causata dalla compressione. Per fare ciò viene utilizzata l'applicazione Validation fornita dal tool scegliendo come ingressi i due file relativi al test set, testData.npy e groundTruth.npy, descritti nel paragrafo precedente, dopo averli convertiti in due file con estensione .csv.

In Tabella 2 sono quindi riportati i risultati della Validation in termini di accuratezza del:

- modello in C, ovvero l'accuratezza del modello in C generato dal tool X-CUBE-AI rispetto alla classificazione corretta;
- modello originale, ovvero l'accuratezza del modello in Keras della rete pre-addestrata rispetto alla classificazione corretta;
- modello in C vs modello originale, ovvero l'accuratezza del modello in C generato dal tool calcolata prendendo il modello originale come riferimento.

		Compressione		
		None	4	8
Accuratezza	Modello in C	92.09%	92.07%	91.32%
	Modello originale	92.09%	92.09%	92.09%
	C vs Originale	100%	99.98%	98.04%

Tabella 2: Accuratezza della rete

Alla luce dei risultati ottenuti, si sceglie un fattore di compressione pari a 4 in quanto il modello in C così generato richiede un'occupazione di memoria sufficientemente minore rispetto alla disponibilità della scheda con una minima perdita di accuratezza se confrontata con quella della rete senza compressione.

3.3. SystemPerformance del modello utilizzato

In questo paragrafo sono riportati i risultati di un'ultima analisi effettuata sul modello in C della rete attraverso l'applicazione SystemPerformance offerta dal tool. Questa applicazione permette di valutare le performance della rete in esecuzione su una scheda STM calcolando il tempo medio di esecuzione di un'inferenza su 16 ingressi randomici. Si fa notare che l'applicazione esegue questo calcolo all'infinito per permettere all'utente di avere una migliore stima dell'effettivo tempo di esecuzione. In Figura 16 sono riportati i risultati ottenuti con la scheda e la rete scelte per questo

specifico progetto; si fa notare che il tempo medio di una singola inferenza di esecuzione è pari a circa 69ms.

```
Running PerfTest on "har" with random inputs (16 iterations)...
.....
Results for "har", 16 inferences @180MHz/180MHz (complexity: 875232 MACC)
duration   : 69.149 ms (average)
CPU cycles : 12446893 (average)
CPU Workload : 6% (duty cycle = 1s)
cycles/MACC : 14.22 (average for all layers)
used stack  : 784 bytes
used heap   : 0:0 0:0 (req:allocated,req:released) max=0 cur=0 (cfg=0)
observer res : 136 bytes used from the heap (7 c-nodes)

Inference time by c-node
kernel   : 69.119ms (time passed in the c-kernel fcts)
                           user     : 0.039ms (time passed in the user cb)

c_id  type      id    time (ms)
-----
0     OPTIMIZED_CONV2D 1     31.117 45.02 %
1     DENSE        4     37.154 53.75 %
2     NL           4     0.014  0.02 %
3     DENSE        5     0.765  1.11 %
4     NL           5     0.014  0.02 %
5     DENSE        6     0.040  0.06 %
6     NL           6     0.011  0.02 %

-----
```

69.119 ms

Figura 16: Risultato della SystemPerformance

4. Hardware

In questo capitolo viene descritto l'hardware impiegato nel progetto, ovvero la scheda STM32F429I-DISC1 e il sensore accelerometrico MPU-6050. Viene inoltre approfondito il protocollo di comunicazione I2C utilizzato per l'interfacciamento scheda-sensore.

4.1. Scheda STM32F429I-DISC1

La STM32F429I-DISC1, visibile in Figura 17, è una scheda della STMicroelectronics che ha come core il microcontrollore STM32F429I, le cui caratteristiche principali sono:

- processore Arm Cortex-M4 con architettura RISC a 32 bit, unità a virgola mobile a singola precisione (FPU) e frequenza massima di 180MHz;
- 256KB di memoria RAM;
- 2MB di memoria Flash.

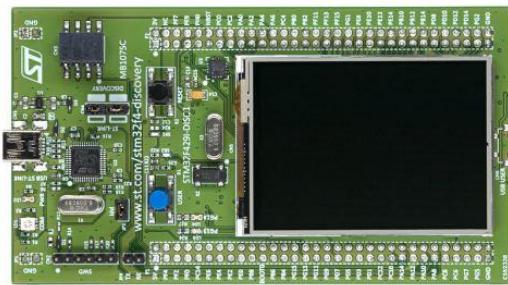


Figura 17: Scheda STM32F429I-DISC1

Ai fini del progetto si fa notare che la scheda è dotata di:

- 3 interfacce di comunicazione I2C;
- 4 interfacce di comunicazioni USART e 4 UART;
- 2 connettori USB;
- ST-LINK/V2-B.

Per maggiori informazioni si rimanda alla documentazione ufficiale riportata in [5].

4.2. Sensore MPU-6050

Il sensore MPU-6050, visibile in Figura 18, combina al suo interno più dispositivi, tra i quali:

- un accelerometro MEMS a 3 assi;
- un giroscopio MEMS a 3 assi;
- un sensore di temperatura;
- un processore di movimento digitale.



Figura 18: Sensore MPU-6050

In particolare, per il presente progetto, il sensore viene utilizzato soltanto come accelerometro. Di seguito sono quindi elencati esclusivamente gli aspetti utilizzati per lo sviluppo dell'applicazione (per la documentazione completa del sensore si rimanda a [6]):

- alimentazione a +3.3V;
- protocollo di comunicazione I2C;
- ADC a 16 bit;
- frequenza di campionamento dell'accelerometro pari a 1KHz;
- range accelerometro modificabile: $\pm 2g$, $\pm 4g$, $\pm 8g$ e $\pm 16g$;
- interrupt programmabili;
- buffer di memoria FIFO di 1024Byte.

Dalla Figura 18 si possono notare i pin del sensore, in particolare si ha:

- VCC, alimentazione a +3.3V;
- GND, massa;
- SCL e SDA, rispettivamente Serial Clock e Serial Data, gestiscono il protocollo di comunicazione I2C per la lettura/scrittura dei registri del sensore;
- XDA e XCL, rispettivamente Auxiliary Serial Data e Auxiliary Serial Clock, gestiscono il protocollo di comunicazione I2C per la comunicazione con altri sensori (ad esempio con un magnetometro);
- AD0, permette di modificare il bit meno significativo (LSB) dell'indirizzo I2C del sensore (se è connesso a +VCC l'indirizzo del sensore è b1101001, se invece non è connesso è b1101000);
- INT, utilizzato dal sensore per inviare segnali di interrupt verso l'esterno.

Per utilizzare il sensore è necessario interagire con i suoi registri attraverso il protocollo di comunicazione I2C; in particolare occorre scrivere su di essi per configurare il sensore in base alle specifiche di progetto e leggerli per accedere ai suoi dati.

Di seguito sono descritti soltanto i registri utilizzati in questo specifico progetto (per la lista completa si rimanda a [7]):

- Sample Rate Divider (SMPRT_DIV), questo registro permette di configurare la frequenza di campionamento del sensore che viene utilizzata per aggiornare i vari registri di output, tra cui il buffer FIFO. In particolare, il registro specifica il fattore di scala SMPRT_DIV da applicare alla frequenza del giroscopio per ottenere la frequenza di campionamento, secondo la seguente formula: $Sample\ Rate = \frac{Gyroscope\ Output\ Rate}{1+SMPLRT_DIV}$. La frequenza del giroscopio è pari a 8KHz quando il filtro passa basso DLPF è disabilitato e pari a 1KHz quando invece è abilitato dal registro Configuration (CONFIG). Per modificare la frequenza di campionamento è quindi sufficiente scrivere sul registro il valore desiderato del fattore di scala con il parametro SMPLRT_DIV intero, senza segno ed a 8 bit.

Register (Hex)	Register (Decimal)	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
19	25						SMPLRT_DIV[7:0]		

Figura 19: Registro Sample Rate Divider (SMPRT_DIV)

- Configuration (CONFIG), nel presente progetto viene usata solo la parte di questo registro, DLPF_CFG, che permette di abilitare il filtro passa basso applicato alle uscite dell'accelerometro.

Register (Hex)	Register (Decimal)	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
1A	26	-	-	EXT_SYNC_SET[2:0]			DLPF_CFG[2:0]		

Figura 20: Registro Configuration (CONFIG)

In Figura 21 sono riportati i valori che è possibile assegnare ai 3 bit del parametro DLPF_CFG in modo da ottenere il filtraggio desiderato.

DLPF_CFG	Accelerometer ($F_s = 1\text{kHz}$)	
	Bandwidth (Hz)	Delay (ms)
0	260	0
1	184	2.0
2	94	3.0
3	44	4.9
4	21	8.5
5	10	13.8
6	5	19.0
7	RESERVED	

Figura 21: Parametro DLPF_CFG del registro Configuration (CONFIG)

- Accelerometer Configuration (ACCEL_CONFIG), come per il registro precedente, anche in questo registro si è usata solo una parte di esso, AFS_SEL, quella che consente di selezionare il range dell'accelerometro.

Register (Hex)	Register (Decimal)	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
1C	28	XA_ST	YA_ST	ZA_ST	AFS_SEL[1:0]			-	

Figura 22: Registro Accelerometer Configuration (ACCEL_CONFIG)

In Figura 23 sono riportati i valori assegnabili al parametro a 2 bit AFS_SEL per settare il range di misura dell'accelerometro.

AFS_SEL	Full Scale Range
0	$\pm 2g$
1	$\pm 4g$
2	$\pm 8g$
3	$\pm 16g$

Figura 23: Parametro AFS_SEL del registro Accelerometer Configuration (ACCEL_CONFIG)

- FIFO Enable (FIFO_EN), questo registro permette di selezionare quali misure del sensore vanno caricate sul buffer FIFO, settando a uno il bit corrispondente alla misura desiderata. Le misure sono scritte sul buffer FIFO seguendo l'ordine del numero dei registri. Nello specifico, per il progetto preso in esame, il bit ACCEL_FIFO_EN abilita la scrittura dei valori dei registri relativi all'accelerometro (Accelerometer Measurements) sul buffer FIFO. Si fa notare che il buffer serve per mantenere temporaneamente in memoria le misure del sensore permettendo alla scheda di leggere direttamente una sequenza di misure senza essere vincolata ad accedere ai singoli registri di misura esattamente alla frequenza di campionamento. L'unico vincolo è che il buffer ha una capacità limitata pari a 1024Byte, superata la quale le misure meno recenti sono sovrascritte.

Register (Hex)	Register (Decimal)	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
23	35	TEMP_FIFO_EN	XG_FIFO_EN	YG_FIFO_EN	ZG_FIFO_EN	ACCEL_FIFO_EN	SLV2_FIFO_EN	SLV1_FIFO_EN	SLV0_FIFO_EN

Figura 24: Registro FIFO Enable (FIFO_EN)

- INT Pin / Bypass Enable Configuration (INT_PIN_CFG), questo registro permette di configurare il segnale di interrupt inviato dal sensore in modo che possa essere correttamente ricevuto e gestito dalla scheda. Di default i singoli bit relativi ai vari parametri del registro sono settati tutti a zero.

Register (Hex)	Register (Decimal)	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
37	55	INT_LEVEL	INT_OPEN	LATCH_INT_EN	INT_RD_CLEAR	FSYNC_INT_LEVEL	FSYNC_INT_EN	I2C_BYPASS_EN	-

Figura 25: Registro INT Pin / Bypass Enable Configuration (INT_PIN_CFG)

- Interrupt Enable (INT_ENABLE), tale registro consente di abilitare le sorgenti di interrupt disponibili sul sensore, quali l'overflow del buffer FIFO e la disponibilità di una nuova misura. Nello specifico, settando a uno soltanto il bit FIFO_OFLOW_EN, il sensore genera un segnale di interrupt sul pin INT ogni volta che il buffer FIFO va in overflow.

Register (Hex)	Register (Decimal)	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
38	56		-		FIFO_OFLOW_EN	I2C_MST_INT_EN	-	-	DATA_RDY_EN

Figura 26: Registro Interrupt Enable (INT_ENABLE)

- Accelerometer Measurements (ACCEL_XOUT_H, ACCEL_XOUT_L, ACCEL_YOUT_H, ACCEL_YOUT_L, ACCEL_ZOUT_H e ACCEL_ZOUT_L), questi registri contengono le misure di accelerazione lungo gli assi X, Y e Z aggiornate alla frequenza di campionamento. La misura di accelerazione su ogni asse è data dalla composizione di due registri a 8 bit adiacenti; ad esempio, come si può notare in Figura 27, per l'accelerazione lungo l'asse X occorre leggere il contenuto dei registri 59 e 60 come un unico intero a 16 bit in complemento a due.

Register (Hex)	Register (Decimal)	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
3B	59				ACCEL_XOUT[15:8]				
3C	60				ACCEL_XOUT[7:0]				
3D	61				ACCEL_YOUT[15:8]				
3E	62				ACCEL_YOUT[7:0]				
3F	63				ACCEL_ZOUT[15:8]				
40	64				ACCEL_ZOUT[7:0]				

Figura 27: Registri Accelerometer Measurements

Per poter ottenere la misura di accelerazione vera e propria occorre dividere la misura grezza contenuta nel registro per il corretto valore di sensibilità in base al range di misura selezionato. In Figura 28 sono riportati i valori di sensibilità al variare del range. Si fa notare che in questo modo si ottiene una misura di accelerazione espressa in g (accelerazione gravitazionale).

AFS_SEL	Full Scale Range	LSB Sensitivity
0	$\pm 2g$	16384 LSB/g
1	$\pm 4g$	8192 LSB/g
2	$\pm 8g$	4096 LSB/g
3	$\pm 16g$	2048 LSB/g

Figura 28: Sensibilità del sensore

- User Control (USER_CTRL), per il progetto questo registro è stato usato per abilitare e disabilitare il buffer FIFO, con il parametro FIFO_EN, e per resettarlo, con il parametro FIFO_RESET.

Register (Hex)	Register (Decimal)	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
6A	106	-	FIFO_EN	I2C_MST_EN	I2C_IF_DIS	-	FIFO_RESET	I2C_MST_RESET	SIG_COND_RESET

Figura 29: Registro User Control (USER_CTRL)

- Power Management 1 (PWR_MGMT_1), questo registro è di default inizializzato con i bit tutti a zero, il che corrisponde al funzionamento normale del sensore senza attivazione delle funzioni di risparmio energetico; per maggiore sicurezza si consiglia comunque di reinizializzare a zero i bit DEVICE_RESET, SLEEP e CYCLE prima di iniziare ad utilizzare il sensore.

Register (Hex)	Register (Decimal)	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
6B	107	DEVICE_RESET	SLEEP	CYCLE	-	TEMP_DIS	CLKSEL[2:0]		

Figura 30: Registro Power Management 1 (PWR_MGMT_1)

- FIFO Count Registers (FIFO_COUNT_H e FIFO_COUNT_L), questi registri sono di sola lettura e contengono il numero di byte attualmente istanziati nel buffer FIFO non ancora letti dal registro FIFO Read Write successivamente descritto. Si fa notare che per leggere correttamente la dimensione corrente del buffer FIFO occorre accedere prima al registro FIFO_COUNT_H e solo successivamente al registro FIFO_COUNT_L e poi combinare i due registri in modo da ottenere i 16 bit contenenti il valore della dimensione.

Register (Hex)	Register (Decimal)	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
72	114	FIFO_COUNT[15:8]							
73	115	FIFO_COUNT[7:0]							

Figura 31: Registri FIFO Count Registers (FIFO_COUNT_H e FIFO_COUNT_L)

- FIFO Read Write (FIFO_R_W), questo registro è usato per leggere i dati dal buffer FIFO. Anche se il registro è a 8 bit il modo corretto di accedere ai dati del buffer FIFO è quello di leggere dal registro tramite protocollo I2C un numero di byte pari al valore contenuto nei registri FIFO COUT. Questi byte vanno poi correttamente interpretati in base alle misure che si è scelto di memorizzare sulla FIFO con il registro FIFO Enable.

Register (Hex)	Register (Decimal)	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
74	116	FIFO_DATA[7:0]							

Figura 32: Registro FIFO Read Write (FIFO_R_W)

4.2.1. Protocollo di comunicazione I2C

Come accennato precedentemente, il protocollo di comunicazione utilizzato per l’interfacciamento scheda-sensore è l’I2C, in quanto esso è l’unico supportato dal sensore utilizzato. Di seguito è riportata una breve descrizione del suo funzionamento.

L’I2C (Inter-Integrated Circuit) è un protocollo di comunicazione che permette la comunicazione seriale sincrona tra due o più dispositivi I2C utilizzando un bus a due fili. Tale protocollo prevede infatti una linea SDA (Serial DAta), sulla quale i dati sono inviati serialmente, e una linea SCL (Serial CLock), per il segnale di clock.

In un bus I2C ci sono due tipi di nodi:

- nodo master: dispositivo che controlla il bus, emette il segnale di clock e genera i segnali di START e STOP;
- nodo slave: dispositivo che si sincronizza sul segnale di clock senza poterlo controllare.

Nel bus deve esserci un unico dispositivo che svolge il ruolo di master; possono invece esserci più dispositivi che svolgono contemporaneamente la funzione di slave, come mostrato in Figura 33. Si richiede però che essi abbiano indirizzi I2C univoci.

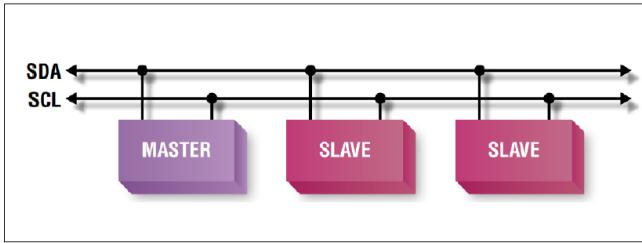


Figura 33: Bus I2C con un master e due slave

Il dispositivo master è quindi il dispositivo che controlla il bus in un certo istante, mentre i dispositivi slave restano in ascolto sul bus ricevendo dati dal master o inviandone qualora questo ne faccia loro richiesta. Per il progetto preso in esame, il ruolo di master è svolto dalla scheda STM32F429I-DISC1, mentre il ruolo di slave dal sensore MPU-6050.

Il protocollo I2C prevede che i dati siano trasmessi a pacchetti di 8 bit seguiti da un bit di Acknowledge (ACK). Il byte di dati è trasferito sulla linea SDA dal Most Significant Bit (MSB) al Less Significant Bit (LSB) ad opera del dispositivo che sta trasmettendo (master o slave), mentre l'ACK è generato dal ricevitore (slave o master) solo se questo riconosce il dato trasmesso (l'assenza del bit di ACK è interpretata come Not Acknowledge NACK).

Ogni sequenza di dati trasmessa è preceduta da una condizione di START e terminata da una condizione di STOP, entrambe generate dal master. Il primo byte di dati è sempre trasmesso dal master e contiene l'indirizzo a 7 bit dello slave seguito da un bit che indica il verso della trasmissione (0: master trasmette e slave riceve; 1: slave trasmette e master riceve).

Concretamente queste operazioni sono eseguite come segue:

- Dati: i singoli bit del dato trasmesso vengono trasferiti sulla linea SDA quando il clock è basso e devono essere stabili quando l'SCL è alto (vengono infatti letti dal ricevitore in queste condizioni);
- START e STOP: la linea SDA è commutata dal master mentre il clock è alto (si distingue per questo dai bit del dato), in particolare la transizione alto-basso corrisponde allo START (in assenza di trasmissione la linea SDA è infatti alta), la transizione basso-alto corrisponde allo STOP (dopo la trasmissione l'SDA deve essere riportato alto per ripristinare le condizioni iniziali);
- ACK e NACK: il ricevitore prende il controllo della linea SDA, lasciata alta dal trasmettitore, e la forza bassa; l'assenza di questa forzatura è interpretata come NACK ed è generalmente sintomo di un errore di trasmissione, viene però anche utilizzata in modalità lettura dal master per fermare la trasmissione dei dati da parte dello slave.

Una sequenza elementare di lettura o scrittura di dati tra master e slave segue quindi il seguente ordine (Figura 34):

- invio del bit di START da parte del master;
- invio dell'indirizzo dello slave (ADDRESS) ad opera del master;
- invio del bit di READ o WRITE, rispettivamente 1 o 0, sempre ad opera del master;
- attesa/invio del bit di Acknowledge (ACK), il quale viene emesso in risposta alla ricezione di un'informazione completa;
- invio/ricezione del byte dei dati (DATA);
- attesa/invio del bit di Acknowledge (ACK), questo step e il precedente possono essere ripetuti per leggere o scrivere più byte;
- invio del bit di STOP da parte del master.

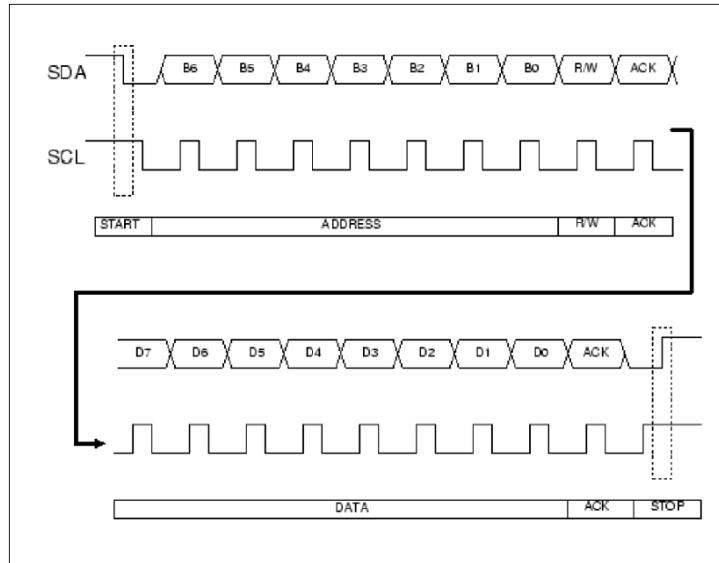


Figura 34: Tipica sequenza di trasferimento dati con protocollo I2C

Questo protocollo è facilmente utilizzabile sulle schede STM32. La STMicroelectronics mette infatti a disposizione il tool STM32CubeMX per abilitare e inizializzare l'I2C e una libreria, già inclusa in ogni progetto creato da STM32CubeIDE, che fornisce le funzioni per utilizzarlo nel progetto. Questi aspetti vengono approfonditi nel capitolo successivo; in particolare nel Paragrafo 5.1 viene descritto il funzionamento del tool STM32CubeMX mentre nel Paragrafo 5.2 sono analizzate le funzioni utilizzate per gestire l'I2C lato scheda.

5. Codice dell'applicazione HAR

In questo capitolo viene approfondito come l'ApplicationTemplate descritto nel Capitolo 2 è stato modificato per ottenere l'applicazione HAR sviluppata nel presente progetto. In particolare, sono commentati l'utilizzo del tool STM32CubeMX, le funzioni per l'I2C e per il sensore, la gestione dell'interrupt per l'overflow del buffer FIFO e le due funzioni per l'acquisizione e l'elaborazione dei dati per la rete neurale, rispettivamente in input e in output. Il codice completo dell'applicazione è reperibile nel Repository GitHub riportato in [8] e in Figura 35 ne è riportato lo schema.

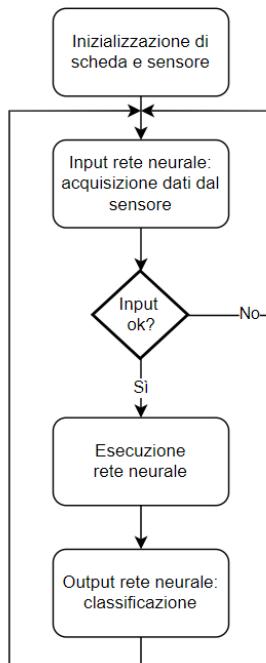


Figura 35: Diagramma di flusso dell'applicazione HAR

Come si può notare dallo schema il codice può essere suddiviso in una parte di inizializzazione della scheda STM32F429I-DISC1 e del sensore MPU-6050 e una parte che rappresenta il ciclo infinito di classificazione dell'attività attraverso la rete neurale. Tale ciclo può essere a sua volta suddiviso in:

- Input della rete neurale, ovvero acquisizione dei dati dal sensore accelerometrico;
- Esecuzione della rete neurale;
- Output della rete neurale, ovvero il risultato della classificazione dell'attività umana.

Si fa notare che, dopo la fase di acquisizione dei dati di input, vi è un controllo sulla loro validità; in particolare occorre controllare se durante l'acquisizione si sia verificato l'overflow del buffer FIFO. In questo caso i dati acquisiti non sono validi e non possono quindi essere forniti alla rete neurale ma vanno acquisite nuove misure accelerometriche prima di procedere con la classificazione.

Il ciclo di classificazione appena descritto è stato implementato modificando leggermente la funzione MX_X_CUBE_AI_Process () generata automaticamente dal tool X-CUBE-AI e riportata in Figura 11. La porzione di codice modificata è evidenziata in Figura 36.

```

381     /* 2 - main loop */
382
383     while(1) {
384
385         /* 1 - acquire and pre-process input data */
386         res = acquire_and_process_data(in_data);
387
388         /* 2a - FIFO overflow management */
389         if (res == -1) {
390             printf("* * * * * * * * * * * * * * * * * * * * * * * *\r\n");
391             printf("\t\t\t FIFO OVERFLOW!\r\n");
392             printf("* * * * * * * * * * * * * * * * * * * * * * * *\r\n\r\n");
393         } else {
394             /* 2b - process the data - call inference engine */
395             ai_run(in_data, out_data);
396             /* 3- post-process the predictions */
397             post_process(out_data);
398         }
399     };
400 }

```

Figura 36: Modifica funzione MX_X_CUBE_AI_Process ()

5.1. STM32CubeMX

Il primo step effettuato per lo sviluppo dell'applicazione HAR è quello di utilizzare il tool STM32CubeMX, per generare automaticamente il codice per l'inizializzazione e la configurazione della scheda STM32F429I-DISC1, e la sua estensione X-CUBE-AI, per implementare la rete neurale sulla scheda e generare l'ApplicationTemplate.

I generici passi da seguire per utilizzare l'ApplicationTemplate sono descritti nella guida riportata nel Paragrafo 2.2; si riportano quindi soltanto le modifiche effettuate per il presente progetto:

- selezionare come scheda la STM32F429I-DISC1, come già mostrato in Figura 4;
- importare il modello della rete neurale per l'HAR descritta al Capitolo 3, caricando il file model.h5 scaricabile dal repository GitHub riportato in [4];
- selezionare come fattore di compressione il valore 4 sulla base dei risultati descritti nel Paragrafo 3.2.

Inoltre, si fa notare che prima di salvare il progetto e generare il codice occorre modificare ulteriormente il file “NomeProgetto.ioc” per poter configurare correttamente la scheda STM32F429I-DISC1 per l'applicazione HAR sviluppata nel progetto.

La prima modifica da effettuare è relativa alla scelta della frequenza di clock del microcontrollore; per il presente progetto si è scelto di impostare tale frequenza al valore massimo pari a 180MHz. Per cambiare la frequenza di clock occorre modificare il campo “HCLK (MHz)” dalla sezione Clock Configuration, come mostrato in Figura 37.

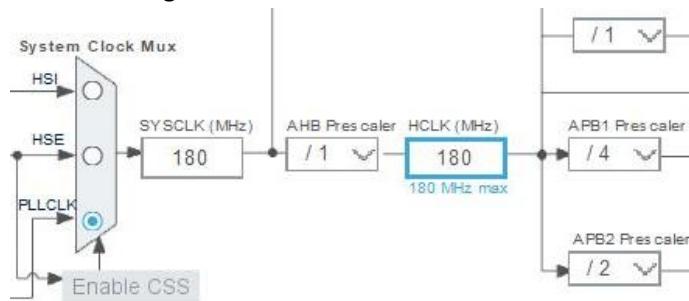


Figura 37: Modifica frequenza di clock

La seconda modifica apportata al file “NomeProgetto.ioc” riguarda invece l'abilitazione delle seguenti funzionalità della STM32F429I-DISC1 utilizzate per l'applicazione del progetto:

- TIM1, timer per poter gestire dei ritardi nell'ordine dei microsecondi che sono necessari per la procedura di recovery dell'I2C successivamente descritta. Seguendo i passaggi riportati in [9], per il progetto si è selezionato il clock interno come sorgente di clock e si è settato come valore di Prescaler “180-1” come mostrato in Figura 38. Il valore del Prescaler è scelto in modo che il

contatore del timer sia incrementato ogni microsecondo; essendo la frequenza di clock di 180MHz, il Prescaler va settato a 180-1, questo perché viene aggiunto automaticamente 1 ad ogni valore impostato.

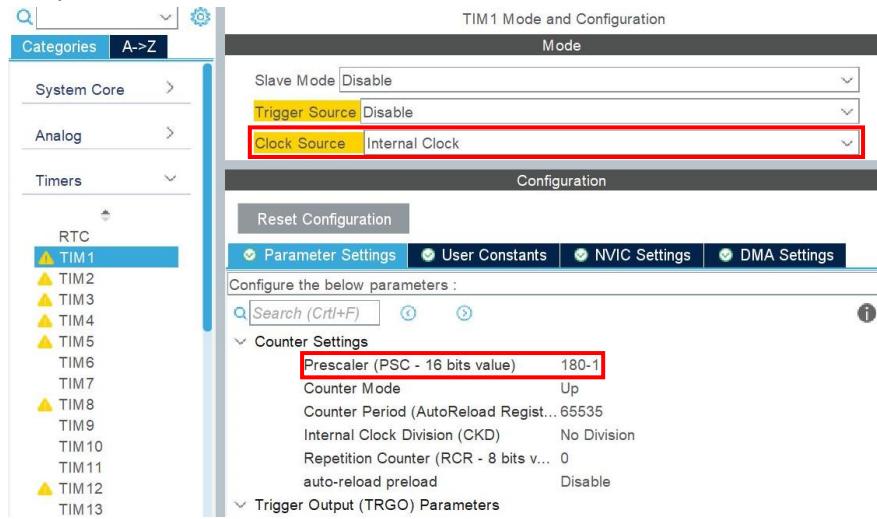


Figura 38: Abilitazione TIM1

- I2C3, interfaccia di comunicazione I2C per la comunicazione scheda-sensore. Per abilitare questo protocollo di comunicazione occorre settare il campo "I2C" al valore "I2C", come mostrato in Figura 39. Si fa notare che di default l'I2C è abilitato in modalità "Standard Mode" con una frequenza di clock pari a 100KHz.

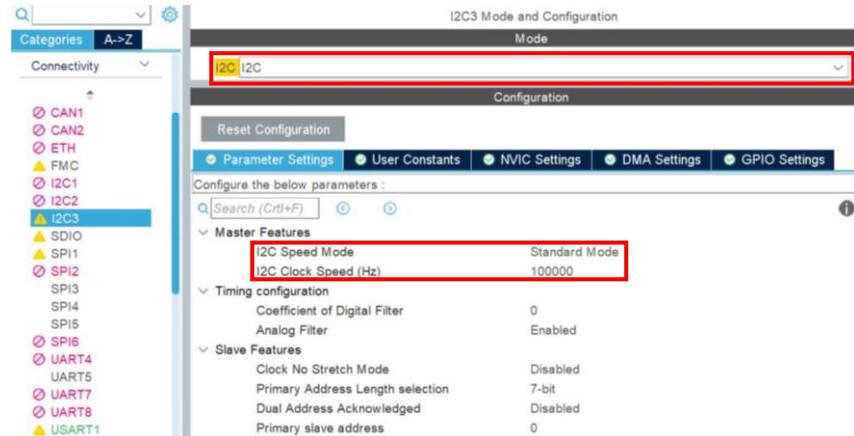


Figura 39: Abilitazione I2C3

In Figura 40 sono mostrati i pin che vengono automaticamente abilitati per la comunicazione I2C; in particolare il pin PA8 per il clock (SCL) e il pin PC9 per i dati (SDA).

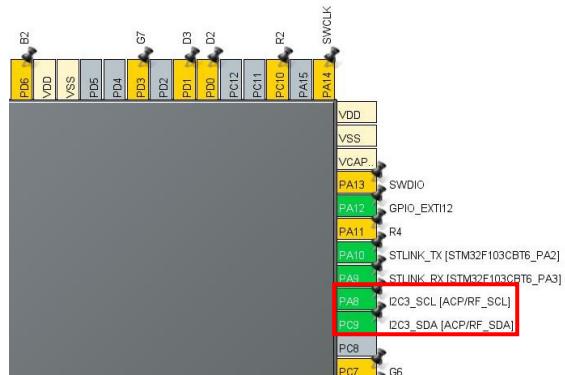


Figura 40: Pin dell'I2C

- USART1, interfaccia di comunicazione USART utilizzata per visualizzare i risultati dell'HAR sullo schermo del PC. Per abilitare questa interfaccia è sufficiente settare il campo “Mode” al valore “Asynchronous”, come mostrato in Figura 41.

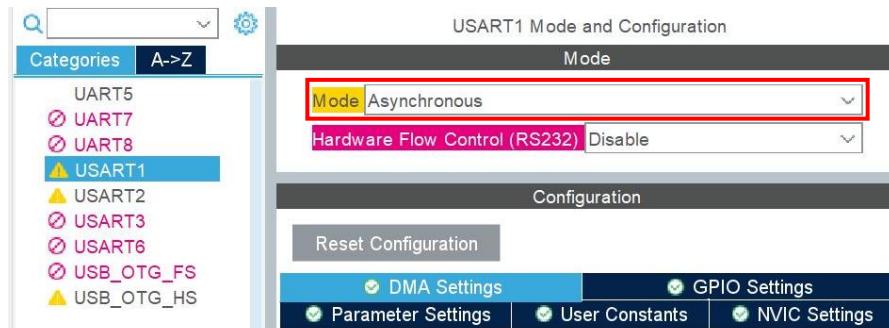


Figura 41: Abilitazione USART1

Il risultato dell’abilitazione delle funzionalità appena descritte è riportato in Figura 42.



Figura 42: Funzionalità abilitate sulla STM32F429I-DISC1

L’ultima modifica apportata al file “NomeProgetto.ioc” è necessaria per permettere alla scheda di ricevere i segnali di interrupt generati dal sensore MPU-6050. In particolare, tra i vari pin selezionabili per la gestione dell’interrupt, si è scelto di utilizzare il pin PA12 che è stato quindi impostato a GPIO_EXTI12, come mostrato in Figura 43.

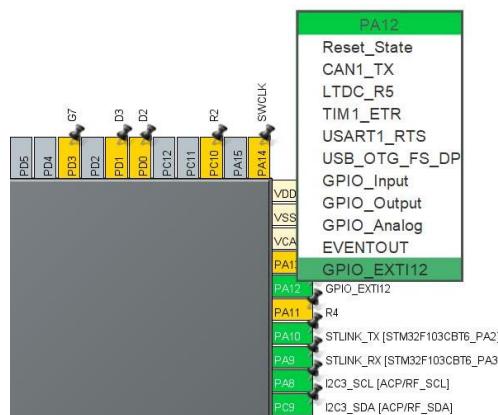




Figura 44: Abilitazione linea di interrupt

Dopo aver effettuato le modifiche precedentemente descritte occorre salvare il file "NomeProgetto.ioc" e far generare automaticamente il codice al tool STM32CubeMX.

Si fa notare che l'ApplicationTemplate non è predisposto per visualizzare messaggi sullo schermo del PC a differenza dell'applicazione Validation e della SystemPerformance. Queste ultime permettono di re-indirizzare i messaggi di output dell'applicazione sulla porta COM grazie ai seguenti file generati automaticamente da STM32CubeMX:

- NOME_PROGETTO\X-CUBE-AI\App\aiTestUtility.c
- NOME_PROGETTO\X-CUBE-AI\App\aiTestUtility.h
- NOME_PROGETTO\X-CUBE-AI\Target\bsp_ai.h

Il modo più semplice per estendere questa funzionalità anche all'applicazione del progetto è quello di copiare i file sopracitati nella cartella NOME_PROGETTO\X-CUBE-AI\App. Così facendo è possibile leggere l'output dell'applicazione HAR aprendo la porta COM; nel presente progetto è stato utilizzato il software PuTTY. Si fa notare che i file vanno copiati dopo aver generato il codice in quanto vengono eliminati automaticamente dall'IDE al momento della generazione automatica del codice.

5.2. Funzioni per la lettura e la scrittura dei registri con l'I2C

Come già accennato precedentemente, la STMicroelectronics mette a disposizione una libreria per la gestione dell'I2C. In questo paragrafo vengono descritte soltanto le due funzioni utilizzate nel progetto per leggere e scrivere i registri del sensore tramite l'I2C:

- **HAL_StatusTypeDef HAL_I2C_Mem_Read(I2C_HandleTypeDef *hi2c, uint16_t DevAddress, uint16_t MemAddress, uint16_t MemAddSize, uint8_t *pData, uint16_t Size, uint32_t Timeout)** è la funzione utilizzata per leggere un registro del sensore ed ha i seguenti argomenti:
 - I2C_HandleTypeDef *hi2c, puntatore alla struttura che contiene le informazioni di configurazione dell'interfaccia I2C3, abilitata come mostrato in Figura 39;
 - uint16_t DevAddress, indirizzo I2C del sensore;
 - uint16_t MemAddress, indirizzo del registro del sensore che si vuole leggere;
 - uint16_t MemAddSize, dimensione in Byte del registro del sensore che si vuole leggere;
 - uint8_t *pData, puntatore alla variabile su cui memorizzare i dati letti dal registro;
 - uint16_t Size, dimensione dei dati da leggere in Byte;
 - uint32_t Timeout, tempo massimo per il completamento dell'operazione di lettura.
- **HAL_StatusTypeDef HAL_I2C_Mem_Write(I2C_HandleTypeDef *hi2c, uint16_t DevAddress, uint16_t MemAddress, uint16_t MemAddSize, uint8_t *pData, uint16_t Size, uint32_t Timeout)** è invece la funzione utilizzata per scrivere su di un registro del sensore ed ha i seguenti argomenti:
 - I2C_HandleTypeDef *hi2c, puntatore alla struttura che contiene le informazioni di configurazione dell'interfaccia I2C3, abilitata come mostrato in Figura 39;
 - uint16_t DevAddress, indirizzo I2C del sensore;
 - uint16_t MemAddress, indirizzo del registro del sensore su cui si vuole scrivere;

- `uint16_t MemAddSize`, dimensione in Byte del registro del sensore su cui si vuole scrivere;
- `uint8_t *pData`, puntatore alla variabile contenente i dati da scrivere sul registro;
- `uint16_t Size`, dimensione dei dati da scrivere in Byte;
- `uint32_t Timeout`, tempo massimo per il completamento dell'operazione di scrittura.

Entrambe le funzioni restituiscono l'esito dell'operazione espresso tramite la variabile enum `HAL_StatusTypeDef` che può assumere i seguenti valori:

- `HAL_OK`, l'operazione è andata a buon fine;
- `HAL_ERROR`, l'operazione non è andata a buon fine;
- `HAL_BUSY`, l'operazione non può essere completata in quanto il master (la scheda) e lo slave (il sensore) sono in mutua attesa;
- `HAL_TIMEOUT`, l'operazione non si conclude entro il tempo impostato con la variabile `Timeout`.

5.3. Funzioni per il sensore

In questo paragrafo sono descritte le funzioni implementate per la gestione del sensore MPU-6050 utilizzato nel seguente progetto per acquisire le misure di accelerazione da fornire in ingresso alla rete neurale per l'HAR. Il particolare il sensore viene impiegato per ottenere un frame di 90 misure accelerometriche lungo gli assi X, Y e Z acquisite con una frequenza di campionamento di 20Hz, in modo da omologarsi alla frequenza di campionamento utilizzata per il training set, e in un range di $\pm 4g$. Il tempo richiesto per l'acquisizione di un frame da fornire in ingresso alla rete neurale è quindi pari a 4,5s; questo significa che per ottenere una nuova classificazione si deve attendere quel lasso di tempo più il tempo di esecuzione della rete, che, come detto, è pari a circa 69ms. Per poter velocizzare la classificazione e per omologare l'input al training set, si è scelto di introdurre un overlapping del 50%, di modo che ogni frame da dare in ingresso alla rete neurale sia costituito dalle ultime 45 misure accelerometriche del frame precedente e da 45 nuove acquisizioni, come mostrato in Figura 45.



Figura 45: Overlapping del 50%

Si è scelto inoltre di sfruttare il buffer FIFO che il sensore mette a disposizione per mantenere temporaneamente in memoria tali misure. In questo modo la scheda può leggere direttamente una sequenza di acquisizioni senza essere vincolata ad accedere ai singoli registri di misura esattamente alla frequenza di campionamento. Si fa notare che così facendo, mentre il sensore memorizza misure accelerometriche sul buffer FIFO, la scheda può eseguire la rete neurale senza doversi preoccupare della perdita di acquisizioni. Come già detto, l'unico vincolo è che il buffer ha una capacità limitata pari a 1024Byte, superata la quale le misure meno recenti sono sovrascritte. Per gestire questa condizione si è scelto di abilitare l'overflow del buffer FIFO come sorgente di interrupt. Questa problematica, nel normale funzionamento, non si presenta nel progetto in quanto l'esecuzione della rete neurale ha un tempo molto minore rispetto al tempo di acquisizione delle nuove 45 misure accelerometriche: la rete ha infatti un tempo medio di esecuzione pari a 69ms, mentre il tempo per acquisire 45 nuove misure accelerometriche alla frequenza di 20Hz è pari a circa 2.25s.

Dopo questa breve introduzione vengono descritte in dettaglio le funzioni per gestire il sensore implementate nei file “GY_521_sensor.c” e “GY_521_sensor.h” presenti nella cartella NOME_PROGETTO\X-CUBE-AI\App. In Figura 46 sono riportati i prototipi di tali funzioni.

```

62 //Sensor's initialization
63 void MPU6050_Init (void);
64
65 //50% overlapping management (translation of the frame's second half in the first half)
66 //and convert raw data into accelerometric measurements
67 void MPU6050_Conv_Order_Frame (void);
68
69 //Convert raw data into accelerometric measurements
70 void MPU6050_Conv_Frame (void);
71
72 //NN's inputs print
73 void MPU6050_Print_Frame_Part (void);
74
75 //Read 45 x 3 acquisitions (X,Y,Z) from FIFO buffer
76 void MPU6050_Read_FIFO_45(uint8_t);
77
78 //I2C recovery
79 void Recovery_i2c(void);
80
81 //Delay in microseconds for I2C recovery
82 void delay_us(uint16_t);
83
84 //Reset and reable FIFO buffer
85 void Reset_Reable_FIFO(void);
86
87 //Read FIFO Count register
88 uint16_t Read_FIFO_Count(void);

```

Figura 46: Prototipi delle funzioni del sensore

Sulla base di quanto detto nel Paragrafo 4.2, di seguito una breve descrizione delle singole funzioni:

- **MPU6050_Init ()**, la funzione permette di inizializzare il sensore configurandone i registri, scrivendo su di essi con la funzione HAL_I2C_Mem_Write () precedentemente descritta.
Il primo registro settato è il Power Management 1, che viene inizializzato con i bit tutti a 0, il che corrisponde al funzionamento normale del sensore senza attivazione delle funzioni di risparmio energetico.
Successivamente viene modificato il registro Accelerometer Configuration assegnando al campo AFS_SEL il valore 1 che corrisponde a settare il range di misura dell'accelerometro $\pm 4g$. Si è scelto tale range sulla base delle accelerazioni presenti nel dataset utilizzato per l'addestramento della rete neurale e di quelle ottenute nelle prime fasi di test del sensore.
Il prossimo registro settato è il Configuration Register che viene utilizzato per abilitare il filtro passa basso da applicare alle uscite del sensore. In particolare, viene assegnato il valore 1 al parametro DLPF_CFG per impostare il filtraggio minimo, larghezza di banda 184Hz e ritardo 2ms, che è necessario introdurre per abbassare la frequenza del giroscopio a 1KHz per poter ottenere con un opportuno fattore di scala una frequenza di campionamento di 20Hz.
La frequenza di campionamento pari a 20Hz viene impostata settando il fattore di scala, ovvero il registro Sample Rate Divider, al valore 49 (0x31 in esadecimale).
Il registro User Control è necessario per abilitare il buffer FIFO settando a 1 il campo FIFO_EN. Successivamente viene modificato il registro FIFO ENABLE in modo da abilitare la scrittura delle misure accelerometriche sul buffer FIFO settando a 1 il bit ACCEL_FIFO_EN.
Si è inizializzato a 0 il registro INT Pin / Bypass Enable Configuration in modo da configurare al valore di default il segnale di interrupt inviato dal sensore alla scheda.
Infine, viene modificato il registro Interrupt Enable per abilitare l'overflow del buffer FIFO come sorgente di interrupt, settando a 1 il bit FIFO_OFLOW_EN.
Si fa notare dopo ogni operazione di scrittura sui registri tramite funzione HAL_I2C_Mem_Write () viene eseguito un controllo per verificare se l'operazione è andata a buon fine o se si sono verificati errori; in quest'ultimo caso ci si limita a ripetere l'operazione in presenza di un errore HAL_ERROR o HAL_TIMEOUT, mentre se viene restituito l'errore HAL_BUSY prima di ripetere l'operazione viene eseguita la funzione Recovery_i2c (), successivamente descritta.

```

33 //Sensor's initialization
34 void MPU6050_Init(void) {
35
36     uint8_t Data;
37     HAL_StatusTypeDef ret;
38
39     //I2C Recovery
40     Recovery_i2c();
41
42     //Power Management 1 register
43     //sensor's normal operation
44     Data = 0;
45     ret = HAL_I2C_Mem_Write(&hi2c3, MPU6050_ADDR, PWR_MGMT_1_REG, 1, &Data, 1, 1000);
46
47     //Check for I2C error
48     while (ret != HAL_OK) {
49         if(ret == HAL_BUSY){
50             //I2C recovery
51             Recovery_i2c();
52         }
53         ret = HAL_I2C_Mem_Write(&hi2c3, MPU6050_ADDR, PWR_MGMT_1_REG, 1, &Data, 1, 1000);
54     }
55
56     //Accelerometer Configuration register
57     //set accelerometer's range to ± 4g
58     Data = 0x08;
59     ret = HAL_I2C_Mem_Write(&hi2c3, MPU6050_ADDR, ACCEL_CONFIG_REG, 1, &Data, 1, 1000);
60
61     //Check for I2C error
62     while (ret != HAL_OK) {
63         if(ret == HAL_BUSY){
64             //I2C recovery
65             Recovery_i2c();
66         }
67         ret = HAL_I2C_Mem_Write(&hi2c3, MPU6050_ADDR, ACCEL_CONFIG_REG, 1, &Data, 1, 1000);
68     }
69
70     //Configuration register
71     //enable Digital Low Pass Filter(DLPF) with highest bandwidth to have gyroscope frequency at 1kHz
72     Data = 0x01;
73     ret = HAL_I2C_Mem_Write(&hi2c3, MPU6050_ADDR, CONFIG_REG, 1, &Data, 1, 1000);
74
75     //Check for I2C error
76     while (ret != HAL_OK) {
77         if(ret == HAL_BUSY){
78             //I2C recovery
79             Recovery_i2c();
80         }
81         ret = HAL_I2C_Mem_Write(&hi2c3, MPU6050_ADDR, CONFIG_REG, 1, &Data, 1, 1000);
82     }
83
84     //Sample Rate Divider register
85     //set sample rate to 20 Hz
86     Data = 0x31;
87     ret = HAL_I2C_Mem_Write(&hi2c3, MPU6050_ADDR, SMPLRT_DIV_REG, 1, &Data, 1, 1000);
88
89     //Check for I2C error
90     while (ret != HAL_OK) {
91         if(ret == HAL_BUSY){
92             //I2C recovery
93             Recovery_i2c();
94         }
95         ret = HAL_I2C_Mem_Write(&hi2c3, MPU6050_ADDR, SMPLRT_DIV_REG, 1, &Data, 1, 1000);
96     }
97
98     //User Control register
99     //enable FIFO buffer
100    Data = 0x40;
101    ret = HAL_I2C_Mem_Write(&hi2c3, MPU6050_ADDR, USER_CTRL_REG, 1, &Data, 1, 1000);
102
103    //Check for I2C error
104    while (ret != HAL_OK) {
105        if(ret == HAL_BUSY){
106            //I2C recovery
107            Recovery_i2c();
108        }
109        ret = HAL_I2C_Mem_Write(&hi2c3, MPU6050_ADDR, USER_CTRL_REG, 1, &Data, 1, 1000);
110    }
111
112    //FIFO Enable register
113    //select accelerometer's data for FIFO buffer
114    Data = 0x08;
115    ret = HAL_I2C_Mem_Write(&hi2c3, MPU6050_ADDR, FIFO_EN_REG, 1, &Data, 1, 1000);
116
117    //Check for I2C error
118    while (ret != HAL_OK) {
119        if(ret == HAL_BUSY){
120            //I2C recovery
121            Recovery_i2c();
122        }

```

```

123     ret = HAL_I2C_Mem_Write(&hi2c3, MPU6050_ADDR, FIFO_EN_REG, 1, &Data, 1, 1000);
124 }
125
126 //INT Pin / Bypass Enable Configuration register
127 //configure interrupt signal (default)
128 Data = 0x00;
129 ret = HAL_I2C_Mem_Write(&hi2c3, MPU6050_ADDR, INT_PIN_CFG_REG, 1, &Data, 1, 1000);
130
131 //Check for I2C error
132 while (ret != HAL_OK) {
133     if(ret == HAL_BUSY){
134         //I2C recovery
135         Recovery_i2c();
136     }
137     ret = HAL_I2C_Mem_Write(&hi2c3, MPU6050_ADDR, INT_PIN_CFG_REG, 1, &Data, 1, 1000);
138 }
139
140 //Interrupt Enable register
141 //enable FIFO overflow interrupt
142 Data = 0x10;
143 ret = HAL_I2C_Mem_Write(&hi2c3, MPU6050_ADDR, INT_ENABLE_REG, 1, &Data, 1, 1000);
144
145 //Check for I2C error
146 while (ret != HAL_OK) {
147     if(ret == HAL_BUSY){
148         //I2C recovery
149         Recovery_i2c();
150     }
151     ret = HAL_I2C_Mem_Write(&hi2c3, MPU6050_ADDR, INT_ENABLE_REG, 1, &Data, 1, 1000);
152 }
153 }
```

Figura 47: Funzione `MPU6050_Init()`

- **MPU6050_Conv_Order_Frame()**, la seguente funzione esegue l'overlapping del 50% come descritto precedentemente e converte le 45 nuove misure accelerometriche grezze lette dal buffer FIFO in dati accelerometrici con unità di misura $\frac{m}{s^2}$ con la seguente formula:

$$\text{Queue_A\#} = \frac{\text{Queue_A\#_Raw}}{\text{LSB_Sensitivity}} g, \text{ dove:}$$

- # sta ad indicare gli assi X, Y e Z;
- Queue_A# è la misura di accelerazione espressa in $\frac{m}{s^2}$;
- Queue_A#_Raw è la misura di accelerazione grezza letta dal buffer FIFO;
- LSB_Sensitivity è la sensibilità del sensore che per il progetto, data la scelta di avere un range pari a $\pm 4g$, è pari a 8192LSB/g (Figura 28);
- g è l'accelerazione gravitazionale.

Si fa notare che Queue_Ax, Queue_Ay e Queue_Az sono dei vettori di dimensione 90 utilizzati per memorizzare il frame da fornire alla rete neurale e che Queue_Ax_Raw, Queue_Ay_Raw e Queue_Az_Raw sono dei vettori di dimensione 90 su cui vengono memorizzati i dati grezzi letti dal buffer FIFO.

```

182 //50% overlapping management (translation of the frame's second half in the first half)
183 //and convert raw data into accelerometric measurements
184 void MPU6050_Conv_Order_Frame(void) {
185
186     //Translation of the frame's second half in the first half in Queue_A* (* = X,Y,Z)
187     for (int8_t j = 0; j < 45; j++) {
188         Queue_AX[j] = Queue_AX[j + 45];
189         Queue_AY[j] = Queue_AY[j + 45];
190         Queue_AZ[j] = Queue_AZ[j + 45];
191     }
192
193     //Convert raw data in Queue_A*_Raw (* = X,Y,Z) into accelerometric measurements Queue_A* (* = X,Y,Z)
194     int8_t i = 0;
195     for (int8_t j = 45; j < dim_frame; j++) {
196         Queue_AX[j] = (Queue_AX_Raw[i] / LSB_Sensitivity) * g;
197         Queue_AY[j] = (Queue_AY_Raw[i] / LSB_Sensitivity) * g;
198         Queue_AZ[j] = (Queue_AZ_Raw[i] / LSB_Sensitivity) * g;
199         i++;
200     }
201 }
```

Figura 48: Funzione `MPU6050_Conv_Order_Frame()`

- **MPU6050_Conv_Frame()**, la funzione converte le prime 90 misure accelerometriche grezze lette dal buffer FIFO in dati accelerometrici con unità di misura $\frac{m}{s^2}$ con la formula descritta precedentemente.

```

203 //Convert raw data into accelerometric measurements
204 void MPU6050_Conv_Frame(void) {
205
206     //Convert raw data in Queue_A*_Raw (* = X,Y,Z) into accelerometric measurements Queue_A* (* = X,Y,Z)
207     for (int8_t j = 0; j < dim_frame; j++) {
208         Queue_Ax[j] = (Queue_Ax_Raw[j] / LSB_Sensitivity) * g;
209         Queue_Ay[j] = (Queue_Ay_Raw[j] / LSB_Sensitivity) * g;
210         Queue_Az[j] = (Queue_Az_Raw[j] / LSB_Sensitivity) * g;
211     }
212 }
```

Figura 49: Funzione *MPU6050_Conv_Frame ()*

- **MPU6050_Print_Frame_Part ()**, la funzione esegue la stampa a schermo del frame di 90x3 misure di accelerazione che è dato in ingresso alla rete neurale per la classificazione. In particolare, ci si limita a mostrare solo le misure con indice 0, 44, 45, 89 in modo da verificare la correttezza sia delle misure del sensore sia dell'overlapping.

```

214 //NN's inputs print
215 void MPU6050_Print_Frame_Part(void) {
216
217     //Print of some NN input frame's sample (index 0,44,45,89)
218     //to check 50% overlapping
219     printf("*****\n");
220     printf("Sample 0:\tAx= %.2f\tAy= %.2f\tAz= %.2f\t[m/s^2]\r\n", Queue_Ax[0],
221             Queue_Ay[0], Queue_Az[0]);
222     printf("Sample 44:\tAx= %.2f\tAy= %.2f\tAz= %.2f\t[m/s^2]\r\n",
223             Queue_Ax[44], Queue_Ay[44], Queue_Az[44]);
224     printf("Sample 45:\tAx= %.2f\tAy= %.2f\tAz= %.2f\t[m/s^2]\r\n",
225             Queue_Ax[45], Queue_Ay[45], Queue_Az[45]);
226     printf("Sample 89:\tAx= %.2f\tAy= %.2f\tAz= %.2f\t[m/s^2]\r\n",
227             Queue_Ax[89], Queue_Ay[89], Queue_Az[89]);
228 }
```

Figura 50: Funzione *MPU6050_Print_Frame_Part ()*

- **MPU6050_Read_FIFO_45 ()**, la seguente funzione legge 45x3 nuove misure di accelerazione dal buffer FIFO e le memorizza nei vettori Queue_Ax_Raw, Queue_Ay_Raw e Queue_Az_Raw a partire dall'indice specificato come argomento. In particolare, si fa notare che la lettura del buffer FIFO può essere effettuata con un'unica operazione di lettura dal registro FIFO Read Write e che ad essa segue il solito controllo di verifica errori dell'I2C descritto precedentemente.

```

155 //Read 45 x 3 acquisitions (X,Y,Z) from FIFO buffer (135 x 2 byte)
156 //input i : index of Queue_A*_Raw (* = X,Y,Z) where FIFO buffer's samples are stored
157 void MPU6050_Read_FIFO_45(uint8_t i) {
158     uint8_t Rec_Data[270];
159
160     //Read 45 x 3 acquisitions from FIFO buffer
161     HAL_StatusTypeDef ret = HAL_I2C_Mem_Read(&i2c3, MPU6050_ADDR, FIFO_R_W_REG, 1, Rec_Data, 270, 1000);
162
163     //Check for I2C error
164     while (ret != HAL_OK) {
165         if(ret == HAL_BUSY){
166             //I2C recovery
167             Recovery_i2c();
168         }
169         ret = HAL_I2C_Mem_Read(&i2c3, MPU6050_ADDR, FIFO_R_W_REG, 1, Rec_Data, 270, 1000);
170     }
171
172     //Store the 45x3 acquisitions in Queue_A*_Raw (* = X,Y,Z)
173     for (uint16_t j = 0; j < 270; j += 6) {
174         Queue_Ax_Raw[i] = (int16_t) Rec_Data[j] << 8 | (int16_t) Rec_Data[j + 1];
175         Queue_Ay_Raw[i] = (int16_t) Rec_Data[j + 2] << 8 | (int16_t) Rec_Data[j + 3];
176         Queue_Az_Raw[i] = (int16_t) Rec_Data[j + 4] << 8 | (int16_t) Rec_Data[j + 5];
177         i++;
178     }
179 }
```

Figura 51: Funzione *MPU6050_Read_FIFO_45 ()*

- **Recovery_i2c ()**, la funzione implementa la procedura di ripristino dell'I2C da svolgere quando si verifica un errore di tipo HAL_BUSY. Si ricorda che questo errore avviene quando il master e lo slave rimangono bloccati in una condizione di mutua attesa. Per poter uscire da tale condizione è necessario assumere manualmente il controllo della linea SCL ed inviare 10 impulsi di clock su questa linea alla frequenza di clock dell'I2C. Così facendo lo slave rilascia la linea e il master può riprenderne il controllo. Per fare ciò occorre disabilitare l'interfaccia I2C3, modificare la funzione del pin PA8 da SCL a GPIO (General Purpose Input Output), inviare 10 impulsi di clock ed infine riabilitare l'interfaccia I2C3. Per inviare gli impulsi di clock alla frequenza corretta è sufficiente alternare un livello logico alto ed uno basso con un opportuno

ritardo, come riportato in [10]. La funzione che genera il ritardo, `delay_us()`, è descritta nel punto successivo.

```

230 //I2C recovery from HAL_BUSY error
231 void Recovery_i2c(void) {
232
233     //I2C de-initialization
234     HAL_I2C_DeInit(&hi2c3);
235
236     //Change the function of SCL pin (PA8) into GPIO pin
237     GPIO_InitTypeDef GPIO_InitStruct = { 0 };
238
239     HAL_GPIO_WritePin(GPIOA, GPIO_PIN_8, GPIO_PIN_SET);
240
241     GPIO_InitStruct.Pin = GPIO_PIN_8;
242     GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
243     GPIO_InitStruct.Pull = GPIO_NOPULL;
244     GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
245     HAL_GPIO_Init(GPIOA, &GPIO_InitStruct);
246
247     //Send 10 clock pulses on SCL pin at 100kHz (I2C Standard Mode)
248     for (int i = 0; i < 10; i++) {
249         delay_us(10);
250         HAL_GPIO_WritePin(GPIOA, GPIO_PIN_8, GPIO_PIN_RESET);
251         delay_us(10);
252         HAL_GPIO_WritePin(GPIOA, GPIO_PIN_8, GPIO_PIN_SET);
253     }
254
255     //I2C de-initialization
256     HAL_I2C_DeInit(&hi2c3);
257
258     //I2C re-initialization
259     HAL_I2C_Init(&hi2c3);
260 }
```

Figura 52: Funzione `Recovery_i2c()`

- **`delay_us()`**, funzione che utilizza il timer TIM1 per introdurre un ritardo di “us” microsecondi, dove “us” è l’argomento della funzione. Si ricorda che il timer è stato configurato affinché il suo contatore si incrementi ogni microsecondo.

```

262 //Delay in microseconds for I2C recovery
263 void delay_us(uint16_t us) {
264
265     //Reset timer counter
266     __HAL_TIM_SET_COUNTER(&htim1, 0);
267
268     //Wait for the timer counter to reach the input microseconds
269     while (__HAL_TIM_GET_COUNTER(&htim1) < us);
270 }
```

Figura 53: Funzione `delay_us()`

- **`Reset_Reable_FIFO()`**, la funzione permette di resettare il buffer FIFO utilizzando il registro User Control. Nello specifico è necessario settare a 1 il parametro FIFO_RESET per effettuare il reset vero e proprio e poi mettere a 1 il parametro FIFO_EN per riabilitare il buffer FIFO. Questa funzione viene utilizzata quando si verifica l’overflow del buffer FIFO per scartare le misure corrotte e ripetere l’acquisizione.

```

272 //Reset and reable FIFO buffer
273 void Reset_Reable_FIFO(){
274     HAL_StatusTypeDef ret;
275
276     //Reset FIFO
277     uint8_t Data = 0x04;
278     ret = HAL_I2C_Mem_Write(&hi2c3, MPU6050_ADDR, USER_CTRL_REG, 1, &Data, 1, 1000);
279
280     //Check for I2C error
281     while (ret != HAL_OK) {
282         if (ret == HAL_BUSY) {
283             //I2C recovery
284             Recovery_i2c();
285         }
286         ret = HAL_I2C_Mem_Write(&hi2c3, MPU6050_ADDR, USER_CTRL_REG, 1, &Data, 1, 1000);
287     }
288
289     //Reable FIFO
290     Data = 0x40;
291     ret = HAL_I2C_Mem_Write(&hi2c3, MPU6050_ADDR, USER_CTRL_REG, 1, &Data, 1, 1000);
292
293     //Check for I2C error
294     while (ret != HAL_OK) {
295         if (ret == HAL_BUSY){
296             //I2C recovery
297             Recovery_i2c();
298         }
299         ret = HAL_I2C_Mem_Write(&hi2c3, MPU6050_ADDR, USER_CTRL_REG, 1, &Data, 1, 1000);
300     }
301 }
```

Figura 54: `Reset_Reable_FIFO()`

- **Read_FIFO_Count()**, la funzione restituisce il numero di byte attualmente istanziati nel buffer FIFO non ancora letti dal registro FIFO Read Write, leggendolo dai registri FIFO_COUNT_H e FIFO_COUNT_L. Si ricorda che per leggere correttamente la dimensione corrente del buffer FIFO occorre accedere in lettura prima al registro FIFO_COUNT_H e solo successivamente al registro FIFO_COUNT_L e poi combinarli. Tale funzione viene utilizzata per verificare che vi siano almeno 45 nuove misure accelerometriche disponibili nel buffer FIFO.

```

303 //Read FIFO Count register
304 //Output : FIFO Count value
305 uint16_t Read_FIFO_Count(){
306
307     HAL_StatusTypeDef ret;
308     uint16_t fifo_count = 0;
309     uint8_t Rec_Data[2];
310
311     //Read FIFO_COUNT_H
312     ret = HAL_I2C_Mem_Read(&hi2c3, MPU6050_ADDR, FIFO_COUNT_H_REG, 1, Rec_Data, 1, 1000);
313
314     //Check for I2C error
315     while (ret != HAL_OK) {
316         if(ret == HAL_BUSY){
317             //I2C recovery
318             Recovery_i2c();
319         }
320         ret = HAL_I2C_Mem_Read(&hi2c3, MPU6050_ADDR, FIFO_COUNT_H_REG, 1, Rec_Data, 1, 1000);
321     }
322
323     //Read FIFO_COUNT_L
324     ret = HAL_I2C_Mem_Read(&hi2c3, MPU6050_ADDR, FIFO_COUNT_L_REG, 1, Rec_Data + 1, 1, 1000);
325
326     //Check for I2C error
327     while (ret != HAL_OK) {
328         if(ret == HAL_BUSY){
329             //I2C recovery
330             Recovery_i2c();
331         }
332         ret = HAL_I2C_Mem_Read(&hi2c3, MPU6050_ADDR, FIFO_COUNT_H_REG, 1, Rec_Data, 1, 1000);
333     }
334
335     //Get FIFO Count value
336     fifo_count = (uint16_t) (Rec_Data[0] << 8 | Rec_Data[1]);
337
338     return fifo_count;
339 }
```

Figura 55: Funzione Read_FIFO_Count ()

5.4. Gestione dell'interrupt per l'overflow del buffer FIFO

In questo paragrafo viene descritto come si è scelto di gestire l'overflow del buffer FIFO. Come già anticipato il buffer FIFO del sensore ha una capacità limitata di 1024Byte, superata la quale le misure meno recenti sono sovrascritte. Anche se questa problematica non si presenta nel normale funzionamento dell'applicazione HAR, si è scelto comunque di gestirla introducendo la variabile di flag globale flag_FIFO_overflow. Tale variabile è inizializzata a 0 nel file GY_521_sensor.c e viene settata a 1 quando la scheda riceve sul pin A12 il segnale di interrupt che il sensore genera quando si verifica l'overflow del buffer FIFO. In Figura 56 è riportato il codice dell'Interrupt Service Routine (ISR) che viene mandata in esecuzione nel momento in cui arriva l'interrupt.

```

568 /* USER CODE BEGIN 4 */
569 // EXTI Line12 External Interrupt ISR Handler CallBackFun
570 void HAL_GPIO_EXTI_Callback(uint16_t GPIO_Pin) {
571     // If the interrupt source is A12 Pin, set FIFO overflow's flag = 1
572     if (GPIO_Pin == GPIO_PIN_12)
573     {
574         flag_FIFO_overflow = 1;
575     }
576 }
577
578 /* USER CODE END 4 */
```

Figura 56: Interrupt Service Routine per la gestione dell'overflow del buffer FIFO

Si ricorda che, lato scheda, il pin A12 è stato configurato per la ricezione di segnali di interrupt, come mostrato nel Paragrafo 5.1, e che, lato sensore, è stata abilitata l'overflow del buffer FIFO come sorgente di interrupt, come descritto nel Paragrafo 5.3.

5.5. Funzione per l'acquisizione dei dati in input alla rete neurale

In questo paragrafo viene commentato come la funzione `acquire_and_process_data()`, introdotta nel Paragrafo 2.2, è stata modificata per l'applicazione HAR sviluppata per questo progetto. Si ricorda che tale funzione viene generata automaticamente nell'ApplicationTemplate ma essa è vuota e va implementata in base all'applicazione in modo che gestisca l'acquisizione e il pre-processamento dei dati di input da fornire alla rete neurale. In Figura 57 è riportato il diagramma di flusso della funzione implementata nel progetto.

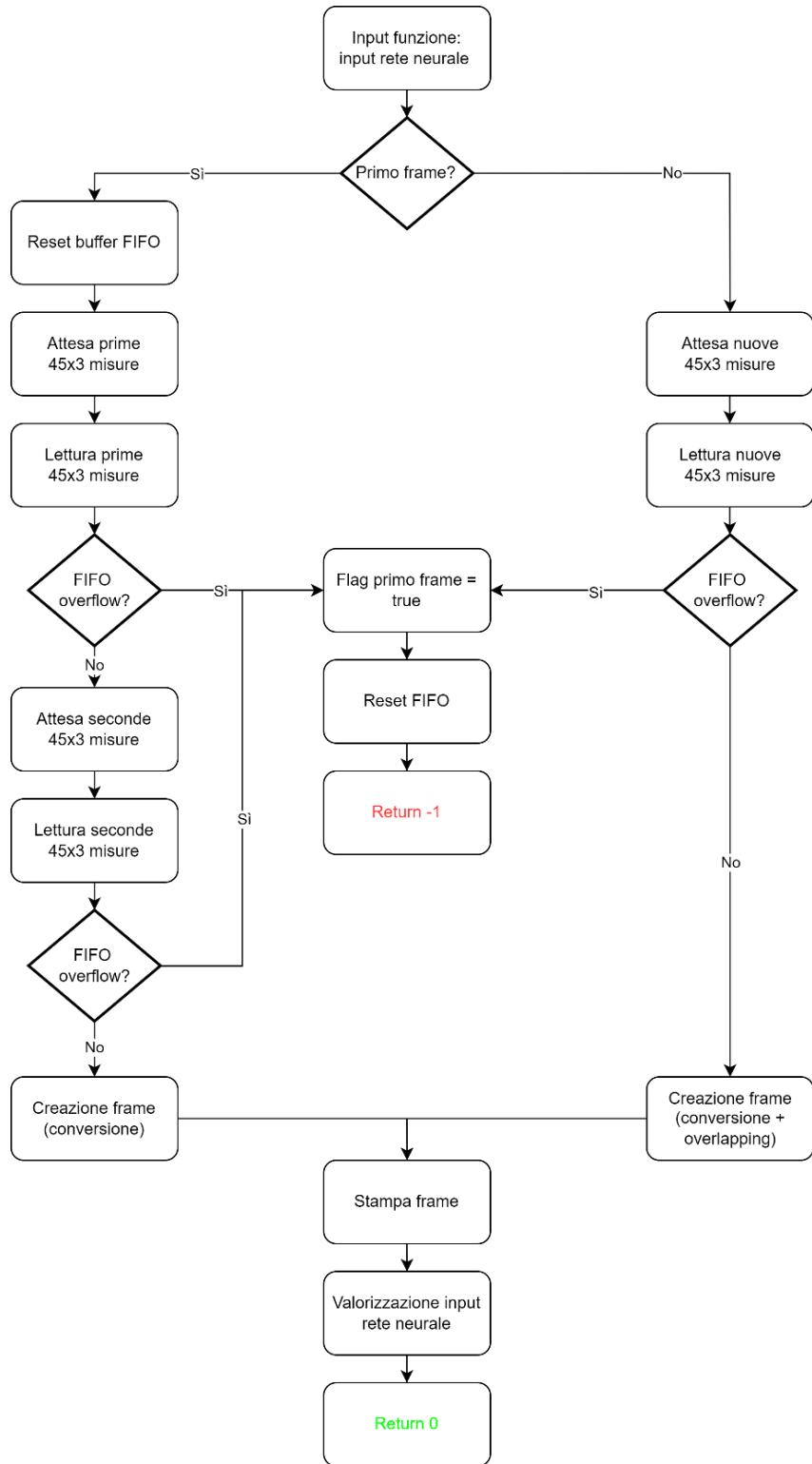


Figura 57: Diagramma di flusso della funzione `acquire_and_process_data()`

Come mostrato nel diagramma di flusso, la funzione prende in ingresso un vettore di byte di dimensione 90x3x4 (90x3 dimensione del frame e x4 perché ogni misura è espressa come float su 4 byte) che deve istanziare con le misure lette dal sensore accelerometrico per creare il frame in ingresso alla rete neurale.

Avendo scelto di gestire l'overlapping al 50%, nella creazione di tale frame occorre separare due casi, che sono distinti con l'utilizzo della variabile booleana flag_first_frame:

- First frame, in questo caso il frame da fornire alla rete neurale è costituito interamente da nuove misure; questo avviene sia alla prima classificazione effettuata dall'applicazione HAR sia a seguito dell'overflow del buffer FIFO e del suo conseguente ripristino durante l'esecuzione dell'applicazione. Di seguito sono riportati i passi da seguire per questo caso:
 - Reset buffer FIFO, questa operazione è svolta con la funzione Reset_Reable_FIFO () e serve per ripristinare il buffer FIFO.
 - Attesa prime 45x3 misure, l'attesa è gestita con un ciclo while all'interno del quale si utilizza la funzione Read_FIFO_Count () per verificare quando la dimensione corrente del buffer FIFO è maggiore o uguale a 270 byte, che corrisponde ad avere almeno 45x3 prime nuove misure (ogni misura occupa 2 byte, quindi 45x3x2=270).
 - Lettura prime 45x3 misure, la lettura è svolta con la funzione MPU6050_Read_FIFO_45 (0) che permette di ottenere le prime 45x3 nuove misure di accelerazione dal buffer FIFO e di memorizzarle nei vettori Queue_Ax_Raw, Queue_Ay_Raw e Queue_Az_Raw a partire dall'indice 0.
 - Check FIFO overflow, questo controllo viene eseguito per verificare la validità dei dati appena letti dal buffer FIFO. I dati letti potrebbero essere infatti corrotti se nel frattempo la scheda ha ricevuto il segnale di interrupt inviato dal sensore quando si verifica l'overflow del buffer FIFO. Questa condizione è gestita con la variabile booleana globale flag_FIFO_overflow che viene settata a true nell'ISR e testata in questo passo. Se non c'è stato l'overflow, quindi il controllo dà esito negativo, si prosegue con i passi successivi. In caso contrario, in presenza di overflow, occorre settare a 1 la variabile flag_first_frame, resettare il buffer FIFO con la funzione Reset_Reable_FIFO () e far terminare la funzione acquire_and_process_data () con codice di errore "-1". Si ricorda che in questo caso, come mostrato in Figura 36, non vengono eseguite le funzioni per l'esecuzione della rete neurale e per il post-processamento dei dati per la classificazione ma si ripete il ciclo rieseguendo la funzione acquire_and_process_data ().
 - Attesa seconde 45x3 misure, l'attesa è gestita esattamente come quella per le prime 45x3 misure.
 - Lettura seconde 45x3 misure, la lettura è analoga a quella delle prime 45x3 misure ma è svolta con la funzione MPU6050_Read_FIFO_45 (45) in modo da memorizzare le nuove 45x3 misure sulla seconda parte dei vettori Queue_Ax_Raw, Queue_Ay_Raw e Queue_Az_Raw, a partire quindi dall'indice 45.
 - Check FIFO overflow, uguale al passo omonimo precedentemente descritto.
 - Creazione frame (conversione), conversione in $\frac{m}{s^2}$ delle 90x3 misure accelerometriche grezze lette dal buffer FIFO con la funzione MPU6050_Conv_Frame ().
- Not-first frame, invece, in questo caso il frame da fornire alla rete neurale è costruito con la tecnica dell'overlapping al 50% già descritta precedentemente. Di seguito sono riportati i passi da seguire per questo caso:

- Attesa nuove 45x3 misure, l'attesa è gestita con un ciclo while all'interno del quale si utilizza la funzione Read_FIFO_Count () per verificare quando la dimensione corrente del buffer FIFO è maggiore o uguale a 270 byte, che corrisponde ad avere almeno 45x3 nuove misure (ogni misura occupa 2 byte, quindi 45x3x2=270).
- Lettura nuove 45x3 misure, la lettura è svolta con la funzione MPU6050_Read_FIFO_45 (0) che permette di ottenere le nuove 45x3 misure di accelerazione dal buffer FIFO e di memorizzarle nei vettori Queue_Ax_Raw, Queue_Ay_Raw e Queue_Az_Raw a partire dall'indice 0.
- Check FIFO overflow, uguale ai passi omonimi precedentemente descritti.
- Creazione frame (conversione + overlapping), conversione in $\frac{m}{s^2}$ delle nuove 45x3 misure accelerometriche grezze lette dal buffer FIFO e gestione dell'overlapping al 50% con la funzione MPU6050_Conv_Order_Frame ().

Per entrambe le casistiche, al termine dei passi precedente descritti, i vettori Queue_Ax, Queue_Ay e Queue_Az contengono il corretto frame di 90x3 misure accelerometriche da fornire in ingresso alla rete neurale. A questo punto si effettua la stampa di alcune parti del frame con la funzione MPU6050_Print_Frame_Part () e si valorizza con il frame il vettore di byte che la funzione acquire_and_process_data () prende in ingresso. In Figura 58 è riportato il codice della funzione acquire_and_process_data () appena descritta.

```

179 // Function to acquire and pre-process NN's input data
180 int acquire_and_process_data(void *data) {
181
182     ai_i8 *pointer = (ai_i8*) data;
183
184     uint16_t fifo_count;
185
186     // First frame
187     if (flag_first_frame == 1) {
188
189         flag_first_frame = 0;
190
191         // Reset and reable FIFO
192         Reset_Reable_FIFO();
193
194         // Waiting for 45 x 3 new acquisitions (135 x 2 byte) from FIFO buffer
195         fifo_count = 0;
196         while (fifo_count < 270) {
197             //Read FIFO Count
198             fifo_count = Read_FIFO_Count();
199         }
200
201         // Read first 45 x 3 acquisitions (X,Y,Z) from FIFO buffer
202         MPU6050_Read_FIFO_45(0);
203
204         // Check FIFO buffer Overflow Interrupt
205         if(flag_FIFO_overflow == 1){
206             flag_FIFO_overflow = 0;
207             flag_first_frame = 1;
208
209             // Reset and reable FIFO
210             Reset_Reable_FIFO();
211
212             return -1;
213         }
214
215         // Waiting for other 45 x 3 new acquisitions (135 x 2 byte) from FIFO buffer
216         fifo_count = 0;
217         while (fifo_count < 270) {
218             //Read FIFO Count
219             fifo_count = Read_FIFO_Count();
220         }
221
222         // Read second 45 x 3 acquisitions (X,Y,Z) from FIFO buffer
223         MPU6050_Read_FIFO_45(45);
224
225         // Check FIFO buffer Overflow Interrupt
226         if(flag_FIFO_overflow == 1){
227             flag_FIFO_overflow = 0;
228             flag_first_frame = 1;
229
230             // Reset and reable FIFO
231             Reset_Reable_FIFO();
232
233             return -1;
234         }
235
236         // Convert raw data into accelerometric measurements (m/s^2)
237         MPU6050_Conv_Frame();

```

```

238     } else {
239         // Not first frame, 50% frame overlapping
240
241         // Waiting for new 45 x 3 new acquisitions (135 x 2 byte) from FIFO buffer
242         fifo_count = 0;
243         while (fifo_count < 270) {
244             //Read FIFO Count
245             fifo_count = Read_FIFO_Count();
246         }
247
248         // Read new 45 x 3 acquisitions (X,Y,Z) from FIFO buffer
249         MPU6050_Read_FIFO_45(0);
250
251         // Check FIFO buffer Overflow Interrupt
252         if(flag_FIFO_overflow == 1){
253             flag_FIFO_overflow = 0;
254             flag_first_frame = 1;
255
256             // Reset and reable FIFO
257             Reset_Reable_FIFO();
258
259             return -1;
260         }
261
262         // 50% overlapping management (translation of the frame's second half in the first half)
263         // and convert raw data into accelerometric measurements
264         MPU6050_Conv_Order_Frame();
265
266     }
267
268     // NN's inputs print
269     MPU6050_Print_Frame_Part();
270
271     // NN's inputs
272     for (uint8_t j = 0; j < dim_frame; ++j) {
273         *(ai_float*) (pointer + j * 12) = Queue_Ax[j];
274         *(ai_float*) (pointer + j * 12 + 4) = Queue_Ay[j];
275         *(ai_float*) (pointer + j * 12 + 8) = Queue_Az[j];
276     }
277
278     return 0;
279 }

```

Figura 58: Codice della funzione acquire_and_process_data ()

5.6. Funzione per l'elaborazione dei dati in output dalla rete neurale

In questo paragrafo vengono commentate le modifiche apportate alla funzione post_process (), anch'essa introdotta nel Paragrafo 2.2. Si ricorda che anche tale funzione viene generata automaticamente nell'ApplicationTemplate ma va customizzata in base all'applicazione in modo che gestisca il post-processamento dei dati di output della rete neurale per determinare la classe di appartenenza dell'input. In particolare, la funzione prende in ingresso il vettore di byte di dimensione 6x4 (6 sono il numero di attività classificate e 4 perché le uscite della rete neurale sono espresse come float su 4 byte) che la funzione ai_run () ha precedentemente valorizzato con le uscite generate dall'esecuzione della rete neurale. La funzione si limita a determinare la classe di appartenenza dell'input della rete neurale cercando il massimo tra le uscite della rete; si ricorda infatti che le uscite della rete rappresentano le probabilità di appartenenza dell'ingresso ad un data classe. Si è scelto quindi di stampare a schermo l'esito della classificazione mostrando l'attività con probabilità massima e il valore di tale probabilità. In Figura 59 è riportato il codice della funzione post_process () appena descritta.

```

282 // Function to post-process NN's output data
283 int post_process(void *data) {
284     ai_i8 *pointer = (ai_i8*)data;
285
286     float max = 0;
287     ai_u8 activity;
288
289     // Research for max probability's activity
290     for (ai_size j = 0; j < 6; ++j) {
291         float value = *(ai_float*) (pointer + j * 4);
292         if (value >= max) {
293             max = value;
294             activity = j + 1;
295         }
296     }
297
298     // Print of max probability's activity
299     switch (activity) {
300     case 1:
301         printf("Activity: DOWNSTAIRS");
302         break;
303     case 2:
304         printf("Activity: JOGGING");
305         break;
306     case 3:
307         printf("Activity: SITTING");
308         break;
309     case 4:
310         printf("Activity: STANDING");
311         break;
312     case 5:
313         printf("Activity: UPSTAIRS");
314         break;
315     case 6:
316         printf("Activity: WALKING");
317         break;
318     }
319     printf("\t\t\t\t Probability: %.2f%% \r\n", max * 100);
320     printf("*****\r\n");
321
322     return 0;
323 }
```

Figura 59: Codice della funzione `post_process()`

6. Test dell'applicazione HAR

In questo capitolo è descritta la fase di test dell'applicazione HAR svolta al fine di valutare la capacità dell'applicazione di classificare correttamente l'attività dell'utente; in particolare vengono descritti i test effettuati, come sono stati realizzati e i risultati ottenuti.

6.1. Setup utilizzato per i test

In questo paragrafo è descritto il setup impiegato per effettuare tutti i test analizzati nel paragrafo successivo.

Il primo step è quello di effettuare i collegamenti fra la scheda STM32F429I-DISC1 e il sensore MPU-6050; in Figura 60 è riportato lo schema di cablaggio tra la scheda e il sensore utilizzato per il presente progetto.

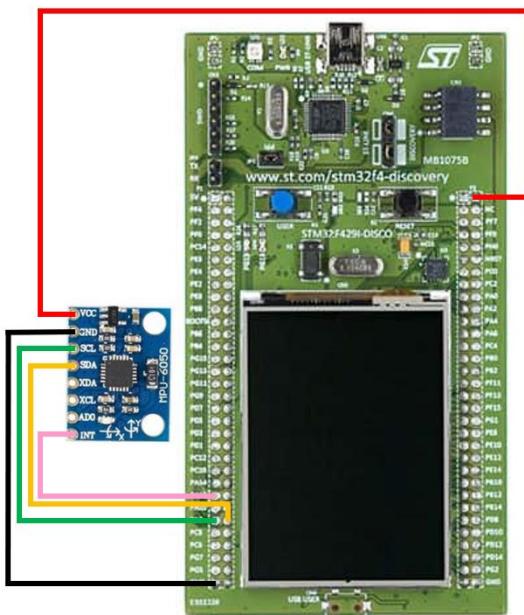


Figura 60: Schema di cablaggio scheda-sensore

In Tabella 3 è invece riportata la corrispondenza tra i pin del sensore e quelli della scheda.

Sensore MPU-6050	Scheda STM32F429I-DISC1
VCC	3V
GND	GND
SCL	PA8
SDA	PC9
INT	PA12

Tabella 3: Corrispondenza dei pin per cablaggio scheda-sensore

Il cablaggio fisico scheda-sensore utilizzato per i test è infine riportato in Figura 61. Si fa notare che si utilizzano solo 5 degli 8 pin disponibili per il sensore; in particolare quelli per l'alimentazione, VCC e GND, quelli per il protocollo di comunicazione I2C tra scheda e sensore, SCL e SDA, e quello per la gestione degli interrupt, INT.

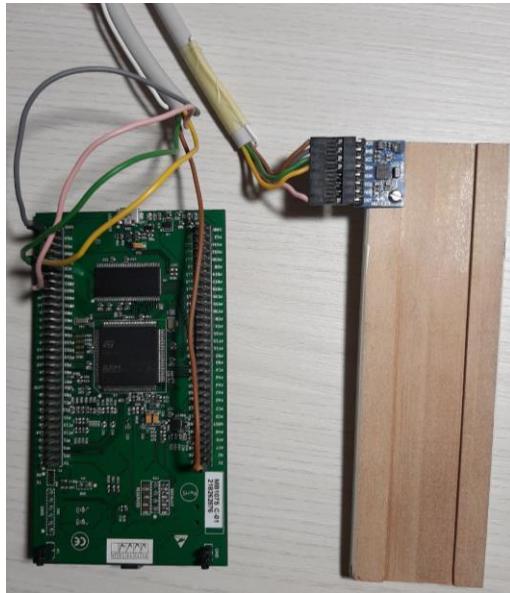


Figura 61: Cablaggio scheda-sensore

Si fa inoltre notare che, come visibile in Figura 61, nel setup per i test il sensore è fissato ad un tavoletta per cercare di ottenere misure accelerometriche analoghe a quelle che fornirebbe l'accelerometro di uno smartphone posto nella tasca dei pantaloni. Si ricorda infatti che la rete neurale utilizzata è stata addestrata con un training set costruito con i dati provenienti da un accelerometro di uno smartphone. Di conseguenza per omologarsi al training set il sensore è stato posizionato nella tasca destra con l'orientamento visibile in Figura 62. Si fa notare che questa scelta di posizionamento è stata fatta sulla base dei risultati dei primi test effettuati sulle attività di Sitting e Standing. Si è visto infatti che soltanto con questo orientamento del sensore la loro classificazione avveniva correttamente. Probabilmente questo limite è dovuto a come è stata addestrata la rete.



Figura 62: Collocazione del sensore in tasca

Lo step successivo da effettuare è quello di caricare il codice compilato dell'applicazione HAR sulla scheda STM32F429I-DISC1; per fare questo è sufficiente scaricare il codice dal repository GitHub riportato in [8], importare il progetto nell'IDE STM32CubeIDE, compilarlo e caricarlo sulla scheda STM32F429I-DISC1 collegata al PC via USB.

Si fa notare che il collegamento PC-scheda via USB va mantenuto anche in fase di test, in quanto il PC è sia fonte di alimentazione per la scheda sia utilizzato come schermo per visualizzare i risultati della classificazione in tempo reale mentre il test è in esecuzione.

L'ultimo aspetto da approfondire è l'utilizzo del software PuTTY, scaricabile da [11], per la lettura dell'output dell'applicazione HAR. Come già anticipato questo software permette di aprire la porta COM su cui è stato reindirizzato l'output dell'applicazione e di visualizzare sullo schermo del PC i dati che transitano sulla porta COM. Per visualizzare tali dati è sufficiente specificare la porta COM e la velocità corretta di trasmissione dei dati come mostrato in Figura 63.

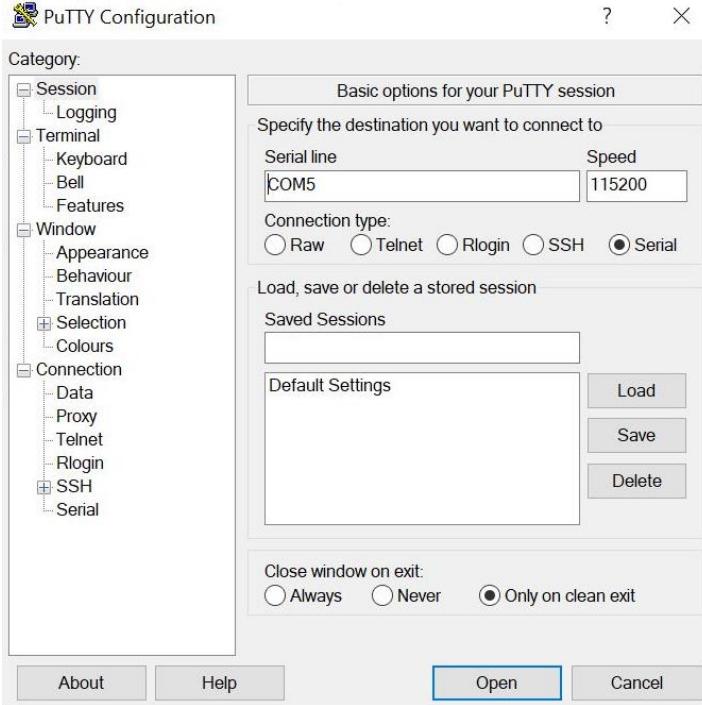


Figura 63: Configurazione di PuTTY per l'apertura della porta COM

Una volta premuto su “Open” si apre una schermata dove vengono visualizzati i dati che transitano sulla porta COM selezionata; un esempio di questi dati è mostrato in Figura 64.

```

COM5 - PuTTY
Activity: UPSTAIRS Probability: 88.84%
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * 
Sample 0: Ax= 1.35 Ay= 8.51 Az= 2.61 [m/s^2]
Sample 44: Ax= 2.14 Ay= 10.13 Az= -2.88 [m/s^2]
Sample 45: Ax= 0.85 Ay= 9.94 Az= 0.61 [m/s^2]
Sample 89: Ax= 1.15 Ay= 9.60 Az= 0.39 [m/s^2]
Activity: UPSTAIRS Probability: 97.84%
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * 
Sample 0: Ax= 0.85 Ay= 9.94 Az= 0.61 [m/s^2]
Sample 44: Ax= 1.15 Ay= 9.60 Az= 0.39 [m/s^2]
Sample 45: Ax= 1.16 Ay= 9.67 Az= 0.37 [m/s^2]
Sample 89: Ax= 1.12 Ay= 9.61 Az= 0.27 [m/s^2]
Activity: STANDING Probability: 99.76%
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * 
Sample 0: Ax= 1.16 Ay= 9.67 Az= 0.37 [m/s^2]
Sample 44: Ax= 1.12 Ay= 9.61 Az= 0.27 [m/s^2]
Sample 45: Ax= 1.04 Ay= 9.64 Az= 0.45 [m/s^2]
Sample 89: Ax= 1.47 Ay= 10.47 Az= 4.40 [m/s^2]
Activity: STANDING Probability: 91.78%
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * 
Sample 0: Ax= 1.04 Ay= 9.64 Az= 0.45 [m/s^2]
Sample 44: Ax= 1.47 Ay= 10.47 Az= 4.40 [m/s^2]
Sample 45: Ax= 0.49 Ay= 9.92 Az= -4.02 [m/s^2]
Sample 89: Ax= 5.97 Ay= 4.18 Az= -2.63 [m/s^2]
Activity: WALKING Probability: 99.73%
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * 
Sample 0: Ax= 0.49 Ay= 9.92 Az= -4.02 [m/s^2]
Sample 44: Ax= 5.97 Ay= 4.18 Az= -2.63 [m/s^2]
Sample 45: Ax= 4.65 Ay= 0.90 Az= 8.94 [m/s^2]
Sample 89: Ax= -0.06 Ay= 0.46 Az= 9.02 [m/s^2]
Activity: UPSTAIRS Probability: 59.16%
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * 

```

Figura 64: Esempio dell'output visualizzato da PuTTY

PuTTY permette inoltre di salvare su un file con estensione .txt i dati trasmessi sulla porta COM. Per abilitare questo log occorre andare sulla schermata Session->Logging, spuntare “All session output” e specificare il percorso e il nome del file del log, come mostrato in Figura 65. Infine, occorre ritornare su Session e salvare le impostazioni settate premendo prima su Default Settings e poi su Save.

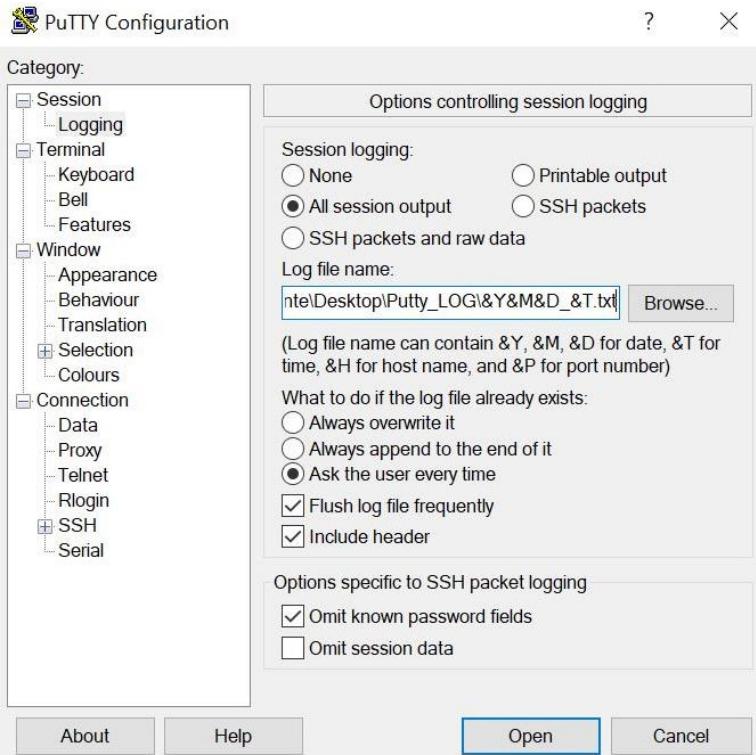


Figura 65: Configurazione di PuTTY per il log

Si ricorda che i dati che si è scelto di stampare e di loggare sono i seguenti:

- input della rete: vengono stampate solo le misure accelerometriche con indice 0, 44, 45 e 89 del frame fornito in ingresso alla rete neurale per la classificazione;
- esito della classificazione: viene mostrata l'attività con probabilità massima e il valore di tale probabilità.

6.2. Test effettuati e risultati

In questo paragrafo sono riportati i test effettuati con il setup sopracitato per valutare la capacità dell'applicazione di classificare correttamente l'attività corrente dell'utente. In particolare, si sono testate tutte le 6 classi di attività (Sitting, Standing, Walking, Jogging, Upstairs e Downstairs) cercando di comprendere i limiti e le problematiche dell'applicazione sviluppata. Ogni test è stato eseguito da due utenti diversi per valutare se le caratteristiche dell'utente influenzano la classificazione; si anticipa che non sono state notate differenze significative e quindi si riportano i risultati dei test senza specificare l'utente.

Di seguito sono riportati i test effettuati e i rispettivi risultati ottenuti:

- **Sitting**, per testare questa attività si è rimasti seduti su una sedia sia stando immobili sia muovendosi leggermente per simulare nel modo più realistico possibile l'attività. I test mostrano che l'applicazione classifica correttamente il Sitting in tutte le condizioni e con una probabilità molto alta, come mostrato in Figura 66.

```

* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
Sample 0:      Ax= 5.97          Ay= 5.39          Az= 5.18          [m/s^2]
Sample 44:     Ax= 6.02          Ay= 5.34          Az= 5.18          [m/s^2]
Sample 45:     Ax= 6.04          Ay= 5.40          Az= 5.18          [m/s^2]
Sample 89:     Ax= 5.94          Ay= 5.41          Az= 5.24          [m/s^2]
Activity: SITTING                               Probability: 99.86%
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *

* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
Sample 0:      Ax= 6.04          Ay= 5.40          Az= 5.18          [m/s^2]
Sample 44:     Ax= 5.94          Ay= 5.41          Az= 5.24          [m/s^2]
Sample 45:     Ax= 5.96          Ay= 5.46          Az= 5.22          [m/s^2]
Sample 89:     Ax= 6.01          Ay= 5.40          Az= 5.17          [m/s^2]
Activity: SITTING                               Probability: 99.87%
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *

* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
Sample 0:      Ax= 5.96          Ay= 5.46          Az= 5.22          [m/s^2]
Sample 44:     Ax= 6.01          Ay= 5.40          Az= 5.17          [m/s^2]
Sample 45:     Ax= 5.98          Ay= 5.41          Az= 5.16          [m/s^2]
Sample 89:     Ax= 5.97          Ay= 5.38          Az= 5.12          [m/s^2]
Activity: SITTING                               Probability: 99.87%
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *

```

Figura 66: Test Sitting –fermo

Si fa notare però che muovendosi la probabilità si abbassa leggermente ma rimane comunque alta, come mostrato in Figura 67.

```

* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
Sample 0:      Ax= 6.71          Ay= 4.78          Az= 4.98          [m/s^2]
Sample 44:     Ax= 7.28          Ay= 4.21          Az= 4.58          [m/s^2]
Sample 45:     Ax= 7.46          Ay= 4.52          Az= 4.46          [m/s^2]
Sample 89:     Ax= 7.17          Ay= 5.11          Az= 4.78          [m/s^2]
Activity: SITTING                               Probability: 97.60%
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *

* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
Sample 0:      Ax= 7.46          Ay= 4.52          Az= 4.46          [m/s^2]
Sample 44:     Ax= 7.17          Ay= 5.11          Az= 4.78          [m/s^2]
Sample 45:     Ax= 6.42          Ay= 4.54          Az= 4.78          [m/s^2]
Sample 89:     Ax= 7.19          Ay= 4.01          Az= 5.14          [m/s^2]
Activity: SITTING                               Probability: 97.11%
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *

* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
Sample 0:      Ax= 6.42          Ay= 4.54          Az= 4.78          [m/s^2]
Sample 44:     Ax= 7.19          Ay= 4.01          Az= 5.14          [m/s^2]
Sample 45:     Ax= 7.11          Ay= 4.23          Az= 5.07          [m/s^2]
Sample 89:     Ax= 6.86          Ay= 5.57          Az= 4.99          [m/s^2]
Activity: SITTING                               Probability: 97.20%
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *

```

Figura 67: Test Sitting - leggero movimento

- **Standing**, questa attività è stata testata rimanendo in piedi sia immobili sia muovendosi leggermente, come per il Sitting. Il risultato dello Standing stando fermo è riportato in Figura 68 e mostra che l’attività è classificata correttamente con una probabilità molto alta.

```

* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
Sample 0:      Ax= 1.47          Ay= 9.64          Az= -0.13          [m/s^2]
Sample 44:     Ax= 1.51          Ay= 9.60          Az= -0.13          [m/s^2]
Sample 45:     Ax= 1.50          Ay= 9.62          Az= -0.19          [m/s^2]
Sample 89:     Ax= 1.55          Ay= 9.64          Az=  0.03          [m/s^2]
Activity: STANDING                         Probability: 99.86%
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *

* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
Sample 0:      Ax= 1.50          Ay= 9.62          Az= -0.19          [m/s^2]
Sample 44:     Ax= 1.55          Ay= 9.64          Az=  0.03          [m/s^2]
Sample 45:     Ax= 1.54          Ay= 9.60          Az= -0.07          [m/s^2]
Sample 89:     Ax= 1.56          Ay= 9.66          Az= -0.15          [m/s^2]
Activity: STANDING                         Probability: 99.85%
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *

* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
Sample 0:      Ax= 1.54          Ay= 9.60          Az= -0.07          [m/s^2]
Sample 44:     Ax= 1.56          Ay= 9.66          Az= -0.15          [m/s^2]
Sample 45:     Ax= 1.52          Ay= 9.65          Az= -0.08          [m/s^2]
Sample 89:     Ax= 1.60          Ay= 9.57          Az= -0.12          [m/s^2]
Activity: STANDING                         Probability: 99.83%
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *

```

Figura 68: Test Standing – fermo

In Figura 69 sono invece riportati i risultati dello Standing muovendosi leggermente; si fa notare che in generale le probabilità si abbassano e che in alcune classificazioni l'attività rilevata è erroneamente quella di Sitting.

```

* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
Sample 0:      Ax= 2.07          Ay= 9.57          Az= -0.43          [m/s^2]
Sample 44:     Ax= 2.06          Ay= 9.39          Az=  1.59          [m/s^2]
Sample 45:     Ax= 2.12          Ay= 9.33          Az=  1.64          [m/s^2]
Sample 89:     Ax= 1.61          Ay= 9.40          Az=  1.52          [m/s^2]
Activity: STANDING                         Probability: 79.57%
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *

* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
Sample 0:      Ax= 2.12          Ay= 9.33          Az=  1.64          [m/s^2]
Sample 44:     Ax= 1.61          Ay= 9.40          Az=  1.52          [m/s^2]
Sample 45:     Ax= 1.73          Ay= 9.13          Az=  1.54          [m/s^2]
Sample 89:     Ax= 2.28          Ay= 9.52          Az=  1.26          [m/s^2]
Activity: SITTING                          Probability: 65.07%
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *

* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
Sample 0:      Ax= 1.73          Ay= 9.13          Az=  1.54          [m/s^2]
Sample 44:     Ax= 2.28          Ay= 9.52          Az=  1.26          [m/s^2]
Sample 45:     Ax= 2.23          Ay= 9.34          Az=  1.33          [m/s^2]
Sample 89:     Ax= 2.53          Ay= 9.42          Az=  0.43          [m/s^2]
Activity: SITTING                          Probability: 78.16%
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *

```

Figura 69: Test Standing - leggero movimento

- **Walking**, questa attività è stata testata a diverse velocità di andatura, questo perché si è notato che l'andatura influenza notevolmente l'esito della classificazione. In particolare, i risultati mostrano che con un'andatura veloce la classificazione è abbastanza corretta (Figura 70); il più delle volte l'attività viene infatti classificata correttamente come Walking anche se alcuni frame sono classificati con Upstairs, questo perché le due attività generano accelerazioni simili.

```

* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
Sample 0:      Ax= 8.12      Ay= 14.31      Az= 4.78      [m/s^2]
Sample 44:     Ax= 4.62      Ay= 7.56       Az= 0.89      [m/s^2]
Sample 45:     Ax= 5.01      Ay= 8.62       Az= -0.09     [m/s^2]
Sample 89:     Ax= 3.19      Ay= 9.91       Az= 0.53      [m/s^2]
Activity: WALKING                               Probability: 99.87%
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *

* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
Sample 0:      Ax= 5.01      Ay= 8.62       Az= -0.09     [m/s^2]
Sample 44:     Ax= 3.19      Ay= 9.91       Az= 0.53      [m/s^2]
Sample 45:     Ax= 5.47      Ay= 11.82      Az= -0.83     [m/s^2]
Sample 89:     Ax= 1.80      Ay= 14.44      Az= -0.01     [m/s^2]
Activity: WALKING                               Probability: 95.25%
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *

* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
Sample 0:      Ax= 5.47      Ay= 11.82      Az= -0.83     [m/s^2]
Sample 44:     Ax= 1.80      Ay= 14.44      Az= -0.01     [m/s^2]
Sample 45:     Ax= 4.35      Ay= 9.08       Az= 1.57      [m/s^2]
Sample 89:     Ax= 2.18      Ay= 3.12       Az= 1.80      [m/s^2]
Activity: UPSTAIRS                             Probability: 90.49%
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *

```

Figura 70: Test Walking - medio veloce

In Figura 71 sono invece riportati i risultati ottenuti con un'andatura lenta; come si può notare in questo caso la rete non classifica correttamente. Questo probabilmente è dovuto al training set con cui è stata addestrata la rete neurale.

```

* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
Sample 0:      Ax= 2.32      Ay= 9.50       Az= -2.07     [m/s^2]
Sample 44:     Ax= 4.72      Ay= 6.95       Az= 0.36      [m/s^2]
Sample 45:     Ax= 4.97      Ay= 7.03       Az= 1.13      [m/s^2]
Sample 89:     Ax= 4.59      Ay= 7.30       Az= -1.37     [m/s^2]
Activity: SITTING                            Probability: 90.74%
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *

* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
Sample 0:      Ax= 4.97      Ay= 7.03       Az= 1.13      [m/s^2]
Sample 44:     Ax= 4.59      Ay= 7.30       Az= -1.37     [m/s^2]
Sample 45:     Ax= 3.06      Ay= 5.65       Az= -0.36     [m/s^2]
Sample 89:     Ax= 3.26      Ay= 5.81       Az= -0.02     [m/s^2]
Activity: UPSTAIRS                           Probability: 88.53%
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *

* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
Sample 0:      Ax= 3.06      Ay= 5.65       Az= -0.36     [m/s^2]
Sample 44:     Ax= 3.26      Ay= 5.81       Az= -0.02     [m/s^2]
Sample 45:     Ax= 2.22      Ay= 4.44       Az= 0.32      [m/s^2]
Sample 89:     Ax= 2.36      Ay= 6.77       Az= -0.40     [m/s^2]
Activity: UPSTAIRS                           Probability: 83.30%
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *

```

Figura 71: Test Walking - lento

- **Jogging**, per testare questa attività gli utenti hanno corso con diverse andature. I test mostrano che l'applicazione classifica sempre correttamente il Jogging con una probabilità molto alta, come mostrato in Figura 72.

```

* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
Sample 0:      Ax= 1.10          Ay= 9.63          Az= -0.34          [m/s^2]
Sample 44:     Ax= 14.83         Ay= 17.06         Az= 5.57          [m/s^2]
Sample 45:     Ax= -0.25         Ay= 17.17         Az= -3.23          [m/s^2]
Sample 89:     Ax= 8.99          Ay= 7.74          Az= 4.35          [m/s^2]
Activity: JOGGING                         Probability: 100.00%
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *

* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
Sample 0:      Ax= -0.25         Ay= 17.17         Az= -3.23          [m/s^2]
Sample 44:     Ax= 8.99          Ay= 7.74          Az= 4.35          [m/s^2]
Sample 45:     Ax= 2.48          Ay= 19.39         Az= 10.44          [m/s^2]
Sample 89:     Ax= -11.24        Ay= 3.26          Az= -2.96          [m/s^2]
Activity: JOGGING                         Probability: 99.66%
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *

* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
Sample 0:      Ax= 2.48          Ay= 19.39         Az= 10.44          [m/s^2]
Sample 44:     Ax= -11.24        Ay= 3.26          Az= -2.96          [m/s^2]
Sample 45:     Ax= 0.37          Ay= 23.09         Az= -2.66          [m/s^2]
Sample 89:     Ax= 1.19          Ay= 4.65          Az= -5.94          [m/s^2]
Activity: JOGGING                         Probability: 99.98%
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *

```

Figura 72: Test Jogging

- **Upstairs**, per testare questa attività gli utenti hanno salito più rampe di scale intervallate da un pianerottolo. Come riportato in Figura 73, i test mostrano che l'applicazione classifica correttamente l'Upstairs con una probabilità alta. Tuttavia, vi sono delle volte in cui l'applicazione classifica altre attività, come lo Standing, questo è probabilmente dovuto ai pianerottoli posti tra le rampe di scale.

```

* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
Sample 0:      Ax= 0.06          Ay= 9.68          Az= -3.76          [m/s^2]
Sample 44:     Ax= 6.05          Ay= 11.93         Az= 1.32          [m/s^2]
Sample 45:     Ax= 6.48          Ay= 12.16         Az= 1.26          [m/s^2]
Sample 89:     Ax= 1.83          Ay= 9.76          Az= -1.32          [m/s^2]
Activity: STANDING                         Probability: 55.86%
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *

* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
Sample 0:      Ax= 6.48          Ay= 12.16         Az= 1.26          [m/s^2]
Sample 44:     Ax= 1.83          Ay= 9.76          Az= -1.32          [m/s^2]
Sample 45:     Ax= 0.86          Ay= 9.32          Az= -2.07          [m/s^2]
Sample 89:     Ax= 6.22          Ay= 10.73         Az= -0.34          [m/s^2]
Activity: UPSTAIRS                         Probability: 81.42%
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *

* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
Sample 0:      Ax= 0.86          Ay= 9.32          Az= -2.07          [m/s^2]
Sample 44:     Ax= 6.22          Ay= 10.73         Az= -0.34          [m/s^2]
Sample 45:     Ax= 4.23          Ay= 12.28         Az= -3.43          [m/s^2]
Sample 89:     Ax= 0.50          Ay= 11.20         Az= -3.40          [m/s^2]
Activity: UPSTAIRS                         Probability: 91.84%
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *

```

Figura 73: Test Upstairs

- **Downstairs**, per testare questa attività gli utenti hanno sceso più rampe di scale intervallate da un pianerottolo. I risultati dei test, come riportato in Figura 74, mostrano che l'applicazione riconosce correttamente che l'utente sta facendo le scale ma non riesce sempre a classificare correttamente il Downstairs, alternandolo all'Upstairs.

```

* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
Sample 0:      Ax= 4.05          Ay= 9.11          Az= -1.09          [m/s^2]
Sample 44:     Ax= 2.17          Ay= 7.58          Az= 1.39          [m/s^2]
Sample 45:     Ax= 1.94          Ay= 11.29         Az= -0.84          [m/s^2]
Sample 89:     Ax= 6.30          Ay= 8.93          Az= 0.17          [m/s^2]
Activity: UPSTAIRS                               Probability: 90.31%
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *

* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
Sample 0:      Ax= 1.94          Ay= 11.29         Az= -0.84          [m/s^2]
Sample 44:     Ax= 6.30          Ay= 8.93          Az= 0.17          [m/s^2]
Sample 45:     Ax= 2.90          Ay= 9.73          Az= -0.93          [m/s^2]
Sample 89:     Ax= 4.43          Ay= 3.56          Az= 2.32          [m/s^2]
Activity: UPSTAIRS                               Probability: 84.28%
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *

* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
Sample 0:      Ax= 2.90          Ay= 9.73          Az= -0.93          [m/s^2]
Sample 44:     Ax= 4.43          Ay= 3.56          Az= 2.32          [m/s^2]
Sample 45:     Ax= 3.27          Ay= 3.08          Az= 2.84          [m/s^2]
Sample 89:     Ax= 5.65          Ay= 6.51          Az= 0.47          [m/s^2]
Activity: DOWNSTAIRS                               Probability: 58.39%
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *

* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
Sample 0:      Ax= 3.27          Ay= 3.08          Az= 2.84          [m/s^2]
Sample 44:     Ax= 5.65          Ay= 6.51          Az= 0.47          [m/s^2]
Sample 45:     Ax= 4.73          Ay= 5.68          Az= -0.15          [m/s^2]
Sample 89:     Ax= 5.65          Ay= 8.43          Az= -0.03          [m/s^2]
Activity: DOWNSTAIRS                               Probability: 73.56%
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *

```

Figura 74: Test Downstairs

7. Conclusioni e sviluppi futuri

I risultati ottenuti in fase di test mostrano che nel complesso l'applicazione HAR sviluppata nel presente progetto riesce a classificare correttamente l'attività dell'utente mandando in esecuzione una rete neurale sulla scheda STM32F429I-DISC1 con il sensore MPU-6050. Questi risultati sono stati ottenuti grazie al fatto che si è tentato il più possibile di replicare in fase di test le condizioni sotto le quali sono stati raccolti i dati utilizzati per addestrare la rete neurale. Si ricorda infatti che i dati del training set provengono dall'accelerometro di uno smartphone mentre i dati da noi utilizzati sono forniti da un sensore esterno.

Alla luce di questi risultati, si può pensare di migliorare tale applicazione come segue:

- sviluppando un prototipo facilmente indossabile dall'utente che non richieda l'utilizzo del PC per l'alimentazione e per visualizzare i risultati a schermo, utilizzando quindi una batteria esterna e lo schermo della scheda STM32F429I-DISC1;
- riaddestrando la rete neurale su un nuovo training set da costruire ad hoc per il nuovo prototipo.

Ci si aspetta che con questi miglioramenti e rieseguendo i test l'applicazione riesca a classificare in modo più accurato le attività dell'utente.

Bibliografia

- [1] STMicroelectronics, [STM32CubeIDE](#)
- [2] M. C. Giannini e N. Elisei, Analisi e valutazione della perdita di accuratezza di una rete neurale su scheda STM, 2021
- [3] STMicroelectronics, [Getting started with X-CUBE-AI Expansion Package for AI](#)
- [4] [GitHub - Human Activity Recognition using CNN in Keras](#)
- [5] STMicroelectronics, [STM32F429I-DISC1's documentation](#)
- [6] Inc. IvenSense, [Datasheet MPU-6050](#)
- [7] Inc. IvenSense, [Register Map MPU-6050](#)
- [8] [Repository GitHub dell'applicazione HAR sviluppata nel progetto](#)
- [9] [Microsecond/Nanoseconds delay in STM32](#)
- [10] [I2C lock-up: prevention and recovery](#)
- [11] [PuTTY](#)