Sommario

Il rapido progredire della tecnologia ha portato l'industria ad un'evoluzione altrettanto veloce, tanto che oggi non è scorretto parlare di un'industria 4.0. Grazie al processo di digitalizzazione e allo sviluppo di unità di calcolo sempre più efficienti è stato possibile entrare in un nuovo paradigma manutentivo che ha permesso di rinnovare vecchie tecniche e strategie di manutenzione. Alla manutenzione preventiva viene così a sostituirsi quella predittiva. Questa si basa sull'individuare sul nascere eventuali condizioni di funzionamento non ordinarie in cui si trovano a lavorare i macchinari localizzati all'interno dell'ambiente industriale, con grande attenzione verso i motori elettrici. In questo contesto assume grande rilevanza la ricerca che negli ultimi decenni ha portato a sviluppare algoritmi di apprendimento automatico. Il machine learning e il deep learning hanno aperto le porte alla possibilità di realizzare architetture a microcontrollore che non solo sono in grado di raccogliere dati sul funzionamento dei macchinari, ma anche di catalogarli e classificarli, così da poter distinguere se il loro stato di operatività è ordinario o anomalo.

Indice

Elenco delle figure

Chapter 1

Introduzione

L'industria 4.0 si fonda su due elementi cardine: la digitalizzazione e l'interconnessione dei diversi dispositivi presenti all'interno dell'ambiente industriale. L'obbiettivo che questa si prefigge, quindi, è quello di costruire ambienti industriali all'interno dei quali poter controllare ed ottimizzare ogni aspetto del processo produttivo, e dove avere dispositivi capaci di comunicare e scambiare informazioni tra loro senza la necessità di un intervento da parte di operatori umani. Nel contesto appena descritto, un ruolo di primaria importanza è ricoperto dalla manutenzione predittiva. Questa si inserisce all'interno di un paradigma manutentivo essenzialmente diverso da quello della manutenzione preventiva, che si basa sull'esecuzione di attività di controllo e di manutenzione periodiche programmate al fine di evitare guasti futuri. La manutenzione predittiva, infatti, si fonda sull'idea di riconoscere sul nascere eventuali problemi legati al funzionamento delle diverse tipologie di macchinari che si trovano comunemente all'interno dell'ambiente di lavoro industriale. La possibilità di individuare tempestivamente condizioni di malfunzionamento nei macchinari industriali consente di intervenire in modo rapido e mirato, così da evitare che questi vengano danneggiati o, nel caso peggiore, che incorrano in un malfunzionamento catastrofico. Si parla quindi di una strategia che va ad abbattere notevolmente i costi legati ad operazioni di controllo periodiche. Ciò è reso possibile dall'uso combinato di sistemi a microcontrollore ed algoritmi sempre più efficienti di intelligenza artificiale, attraverso i quali è possibile creare sensori "intelligenti" che siano in grado di riconoscere quando un macchinario o un componente si trova a lavorare in condizioni di funzionamento anomalo, così da poter agire di conseguenza nel minor tempo possibile ed evitare possibili guasti.

1.1 Il monitoraggio dei motori elettrici

Il focus principale di questa tesi è la manutenzione predittiva dei motori elettrici, e in particolare modo dei motori ad induzione, più comunemente conosciuti come motori asincroni. I motori ad induzione rappresentano infatti uno dei punti focali dell'industria moderna, dal momento che circa il 90% dei processi industriali dipende da essi. Risulta quindi di fondamentale importanza monitorare le condizioni di salute dei motori. L'insorgere di condizioni di malfunzionamento, derivanti da problemi di natura meccanica e/o elettrica, porterebbe il motore a lavorare in regime di funzionamento anomalo, causando gravi perdite in termini di rendimento e rischiando di danneggiare il rotore o lo statore. Inoltre, nel caso peggiore, il motore potrebbe incorrere in un malfunzionamento catastrofico (Figura 1.1).

Bisogna quindi individuare le principali categorie di guasti che possono verificarsi all'interno di un motore chiamato ad operare per lunghi periodi di tempo. Tra queste, quelle di maggiore interesse sono:

- Danneggiamento dei cuscinetti: possono essere causati da vari fattori, come mancanza di lubrificazione o stress meccanici sulle sfere dei cuscinetti con conseguente danneggiamento degli elementi rotanti.
- Danneggiamento della pista interna ed esterna
- Guasto dello statore o del rotore

Questi guasti porterebbero ad una serie di effetti negativi, quali ad esempio una maggiore quantità di calore generato, un maggior consumo di energia o un minor valore di coppia d'uscita. Risulta quindi essenziale essere in grado di monitorare le condizioni di salute di un motore, così da essere in grado di intervenire in maniera tempestiva qualora esso si trovasse a lavorare in condizioni non ordinarie e contenere la natura del danno. Definito quindi il target principale del lavoro di tesi, bisogna capire di quali strumenti abbiamo bisogno per effettuare la manutenzione predittiva dei motori elettrici.

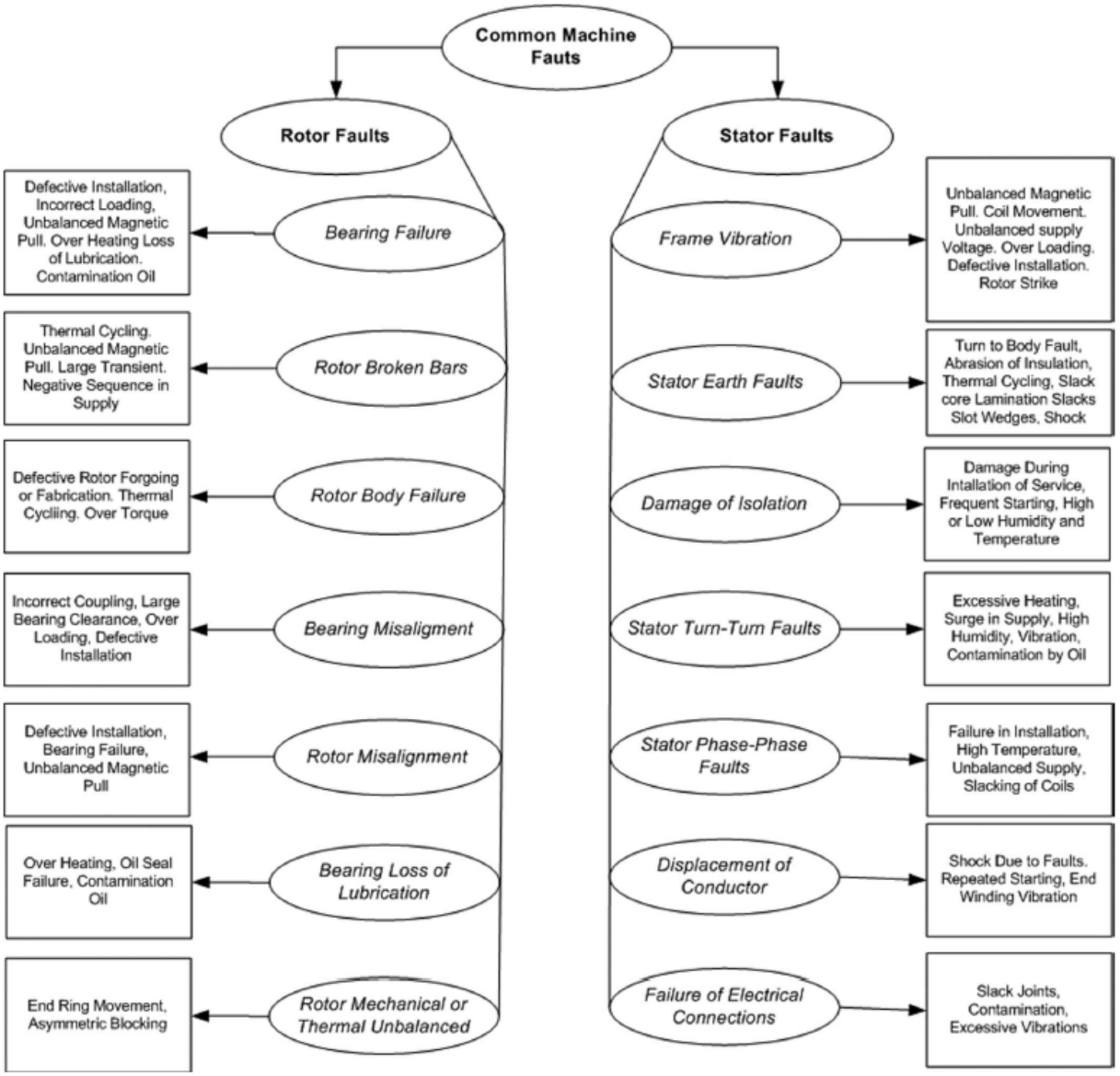


Figure 1.1: Tipologie di guasti nei motori a induzione

1.2 Analisi delle vibrazioni

Sono diverse le tecniche che possono essere utilizzate per il monitoraggio delle condizioni di salute e di funzionamento dei motori elettrici. Una delle più popolari è la MCSA(motor current signature analysis). Questa tecnica si basa sul monitoraggio della corrente di statore del motore, e si propone di associare allo specifico guasto preso in esame un certo valore di segnale di corrente misurato. Il segnale viene processato attraverso la trasformata di Fourier veloce (FFT): analizzando lo spettro del segnale è possibile individuare quale condizione di guasto o di malfunzionamento insorge nel motore, in quanto ad ognuna di queste corrisponde

una ”firma” in termini di corrente che presenta delle armoniche all'interno dello spettro. Queste firme dipendono sia dalle caratteristiche del motore che dalle sue condizioni di funzionamento. Si tratta di una tecnica che ha dimostrato di poter ottenere buoni risultati, ma che presenta un grande svantaggio: per ogni specifica condizione di funzionamento anomalo è richiesta strumentazione dedicata ed un algoritmo per il riconoscimento automatico della stessa.

Un'altra tecnica adoperata nell'ambito della manutenzione predittiva è la vibration signature analysis (VSA); quasi il 100 % dei guasti nei motori si riflette infatti in variazioni nelle vibrazioni. La VSA, in maniera simile alla MCSA, ha lo scopo di riconoscere le condizioni di funzionamento anomalo del motore, e di associare a queste una firma in termini di vibrazioni a cui esso è soggetto. Ogni motore è caratterizzato da una vibrazione propria, e l'analisi spettrale di tali vibrazioni consente di monitorare le condizioni di salute del motore. L'analisi nel dominio della frequenza risulta estremamente efficace, in quanto rispetto a quella nel dominio del tempo fornisce informazioni più dettagliate riguardo lo stato di funzionamento del motore. Come mostrato nella figura 1.2, sono state stabilite delle correlazione tra alcune specifiche componenti armoniche e possibili condizioni di guasto. Anche in questo caso, quindi, viene adoperata la FFT come strumento matematico di elaborazione dei dati raccolti, ovvero misure di accelerazione lungo i 3 assi.

POSSIBLE FAILURE RELATED TO THE VIBRATION FREQUENCY (RPM IS THE MOTOR SPINDLE)

Frequency	Possible Failure
1 x RPM	Unbalance, looseness, misalignment of gear or pulley, resonance, electrical problems, alternative forces
2 x RPM	Mechanical offset, misalignment, alternative forces, resonance
3 x RPM	Misalignment, axial mechanical breath
< 1 x RPM	Slip, oil whirl
Supply frequency	Electrical problems
RPM harmonics	Broken bars, Damaged gears, aerodynamic forces, hydraulic forces, mechanical set, alternative forces

Figure 1.2: Relazione tra RPM e frequenza delle vibrazioni

La tecnica che è stata scelta per il lavoro di tesi prevede l'analisi delle vibrazioni del motore elettrico per ricavare informazioni utili riguardo lo stato di funzionamento dello stesso.

1.3 Costruzione di un modello di apprendimento automatico

Nel corso degli ultimi decenni, approcci che prevedono l'utilizzo di modelli di machine learning sono stati ampiamente utilizzati nell'ambito della manutenzione predittiva per la diagnosi delle condizioni di funzionamento dei motori ad induzione. Metodologie di classificazione come alberi decisionali (decision trees) o reti neurali (neural networks) hanno dimostrato di essere molto efficienti quando combinati a tecniche di elaborazione dei segnali. Il paradigma di apprendimento utilizzato nel lavoro di tesi rientra nella categoria dell'apprendimento supervisionato: viene raccolta una certa quantità di campioni, che vengono opportunamente "etichettati" in base allo stato di funzionamento dell'oggetto vibrante, il motore elettrico in questo caso. Le misure di accelerazione vengono acquisite dal sensore, processate attraverso l'utilizzo della FFT e successivamente etichettate. Un passaggio fondamentale all'interno di questo processo è la scelta delle "features", che consente di ridurre il numero di variabili di input del modello predittivo che si intende costruire (Figura 1.3). Avere un numero di variabili di ingresso limitato consente infatti di ridurre il costo computazionale del modello e di migliorarne le performance.

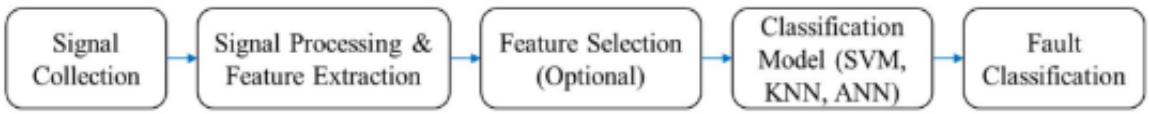


Figure 1.3: Processo di apprendimento automatico

Una scelta possibile, ad esempio, è quella di prelevare per uno dei tre assi un certo numero di frequenze all'interno dello spettro del segnale e i valori delle ampiezze corrispondenti. In questo caso le variabili di input sarebbero 2, un'ampiezza e una frequenza. Una volta determinato il criterio dietro la scelta delle frequenze e delle ampiezze da prelevare bisogna etichettare i dati ricavati dalle misure. Le etichette che vengono assegnate alle coppie rappresentano le variabili di uscita. Il set di dati etichettati così costruito diviene il "training set" del modello di apprendimento, ovvero l'insieme di dati sul quale questo viene addestrato. Scopo dell'algoritmo di addestramento è quello di far comprendere al modello la relazione che sussiste tra le variabili

di ingresso e quelle di uscita, cosicché esso sia in grado di eseguire previsioni anche qualora non conoscesse a priori il valore dell'uscita.

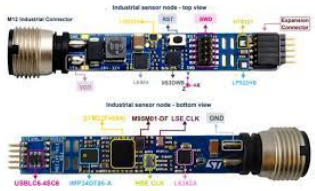
Chapter 2

Kit di sviluppo STM

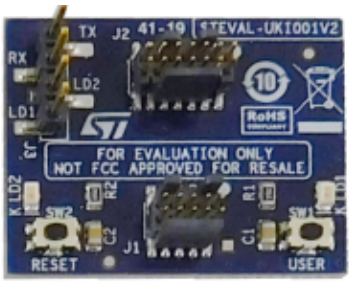
Per eseguire le misure di accelerazione è stato utilizzato un kit prodotto da STMicroeletronics, lo ”STEVAL-BFA001V2B”. Si tratta di un kit di progettazione pensato per il monitoraggio delle condizioni e la manutenzione predittiva, il cui layout soddisfa i requisiti IEC61000-4-2/4 e EN60947 per applicazioni industriali. Il kit consta di 3 componenti principali:

- Sensor board (IDP005V2) (Figura 2.1-a)
- Communicationn adapter board (STEVAL-UKI001V2) (Figura 2.1-b)
- Strumento di programmazione e debug (STLINK-V3MINI): (Figura 2.1-c)

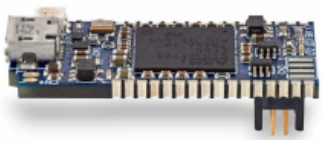
La sensor board è una scheda compatta che rappresenta l’hardware di misura e di controllo del kit. Su di essa troviamo diverse tipologie di sensori, come sensori di umidità, di temperatura o di pressione. Il dispositivo STLINK-V3MINI è invece una sonda di programmazione e debug stand-alone per i microcontrollori STM32. È composto da un modulo principale e da un adapter board che si interfaccia e si collega fisicamente alla sensor board. Il firware, che lavora sul microcontrollore a 32 bit ad elevate prestazioni STM32F469AI, ARM Cortex-M4, contiene algoritmi dedicati per l’elaborazione avanzata di segnali nel dominio del tempo e della frequenza. I risultati dell’analisi dei dati dei sensori possono essere inoltre visualizzati su un emulatore di terminale PC tramite connettività cablata.



(a) Sensor Board IDP005V2



(b) Adapter Board STEVAL-UKI001V2



(c) STLINK-V3MINI

Figure 2.1: Kit STEVAL-BFA001V2B

Il collegamento dei 3 dispositivi viene eseguito connettendo l'STLINK-V3MINI all'adapter board, la quale va a sua volta connessa alla sensor board. Il debugger va poi collegato tramite micro-USB al PC e ad un'alimentazione esterna la sensor board (Figura 2.2).

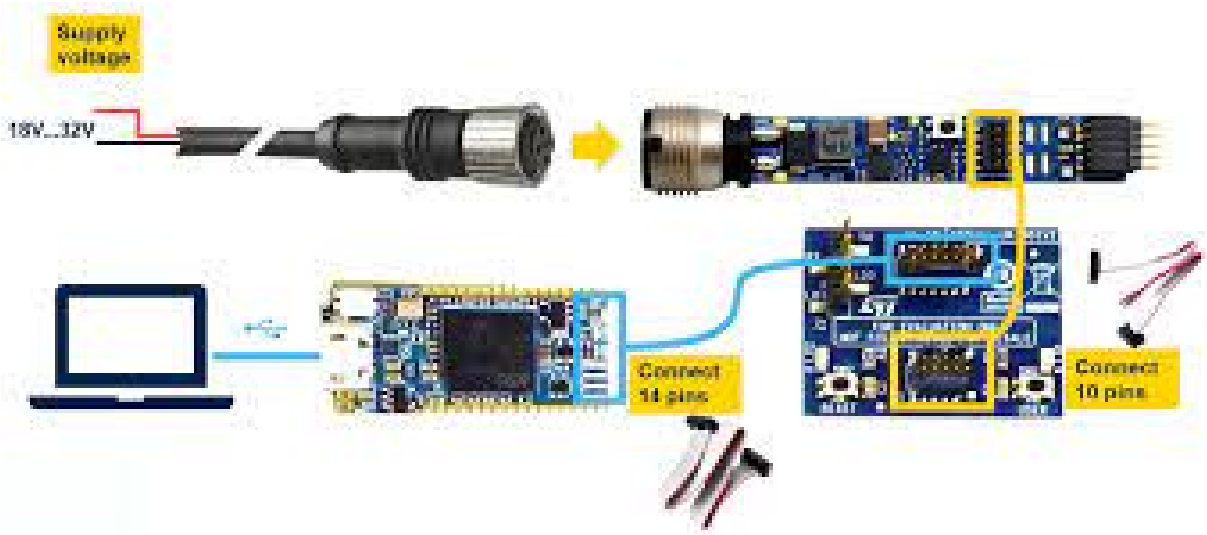


Figure 2.2: Collegamento dispositivi kit STEVAL-BFA001V2B

2.1 Accelerometro IIS3DWB

Come già accennato in precedenza, sulla sensor board è presente un vasto assortimento di sensori. Tra questi, quello utile per eseguire le misure di accelerazione è il sensore IIS3DWB, in quanto dotato di un accelerometro digitale a 3 assi capace di operare su un'ampia gamma di frequenze. La possibilità di operare su un intervallo di frequenze ultra-ampio, unito alla sensibilità molto stabile e la capacità di lavorare in un range di temperatura molto ampio (fino

a 105°C), rendono il IIS3DWB ideale per il monitoraggio delle vibrazioni nelle applicazioni industriali. Questo dispositivo è installato direttamente sull'oggetto che vibra: in questo modo l'energia delle vibrazioni viene trasformata in un segnale elettrico proporzionale all'accelerazione istantanea dell'oggetto vibrante.

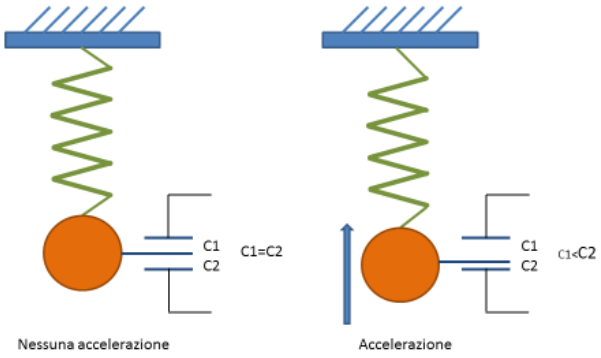


Figure 2.3: Modello di accelerometro capacitivo

L'IIS3DWB è un accelerometro capacitivo. Questo usa la variazione della capacità elettrica di un condensatore al variare della distanza tra le due armature per rilevare lo spostamento della massa (Figura 2.3).

Il principio di funzionamento di questo dispositivo si basa sul posizionare un peso installato su una delle due molle. Uno dei due elementi elastici è fissato alle armature del condensatore, l'altra, invece, al peso. La forza che agisce sul sensore causa lo spostamento del peso sulle molle, in questo modo varia la distanza tra l'elemento capacitivo e la massa, e ha luogo una variazione della capacità. In particolare, l'IIS3DWB in particolare è realizzato con la tecnologia MEMS(micro electro-mechanical systems), la più utilizzata in questo ambito. Il sensore ha un intervallo di accelerazione a scala completa selezionabile di ±2/±4/±8/±16 g ed è in grado di misurare accelerazioni con una larghezza di banda fino a 5 kHz con una velocità dati di uscita di 26,7 kHz.

2.2 Software STM32CUBEMX

L'utilizzo del software STM32CUBEMX, prodotto e distribuito da STM, è stato di notevole supporto, in quanto ha consentito di impostare e di attivare ad un livello alto, e dunque evitando di farlo a livello dell'IDE di programmazione, tutti i canali di comunicazione, le funzioni da utilizzare e le periferiche necessarie per eseguire le misure di accelerazione e trasmettere i dati

relativi alle stesse. Creato quindi un progetto su CubeMX e selezionato il microcontrollore utilizzato (Figura 2.4), basta consultare il datasheet del kit per vedere quali sono i pin utili per configurare e programmare l'accelerometro.

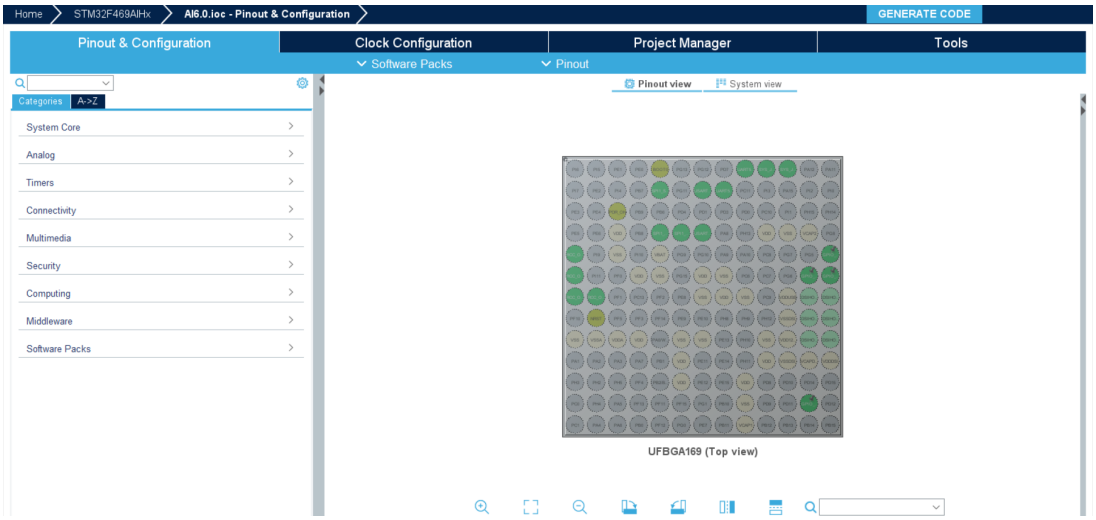


Figure 2.4: Interfaccia CubeMX

I pinout di interesse sono:

- PB3,PB4,PB5: sono i pin SPI, configurati all'attivazione del canale SP1
- PG2: corrisponde al GPIO di output che individua il chip select del sensore
- PD2,PC12: UART RX e TX per la comunicazione seriale dei dati

2.2.1 Protocollo SPI

La comunicazione con il sensore IIS3DWB avviene attraverso un protocollo di tipo SPI: si tratta di una comunicazione singolo Master e singolo Slave di tipo sincrono e full-duplex. Ad avviarla è il master, che abilita lo slave tramite il segnale CS ed impone il segnale clock (SPC) sulla linea di comunicazione. Sia il dispositivo master che quello slave sono dotati di registri a scorrimento interni la cui dimensione è uguale, in cui i bit vengono emessi e contemporaneamente immessi. I registri a scorrimento sono interfacce che consentono di impartire comandi e di trasmettere dati che arrivano serialmente e che vengono prelevati in parallelo. Ad ogni impulso di clock, i dispositivi che stanno comunicando sulle linee del bus SPI emettono un bit dal loro registro e lo rimpiazzano con un bit proveniente dal loro interlocutore. Questo avviene tramite le linee MISO/SDI e MOSI/SDO. Il segnale MISO è la linea attraverso cui il dispositivo (master o slave)

riceve il dato seriale emesso dalla controparte. Sullo stesso fronte di commutazione del clock, il dispositivo emette, con la stessa cadenza, il suo output ponendo il dato sulla linea MOSI (Figura 2.5).

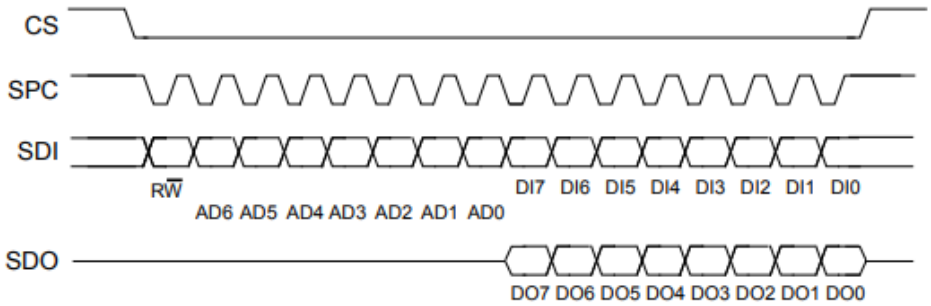


Figure 2.5: Protocollo di lettura e scrittura

Nel caso preso in esame, il ruolo di master è interpretato dal microcontrollore, mentre quello di slave dal sensore di accelerazione. La sincronizzazione avviene sul fronte di salita o di discesa del clock ed è regolata da due parametri configurabili:

- CPOL: determina la polarità del clock. Quando è impostato a 0, il clock si porta a livello logico basso quando si trova nel suo stato di riposo, mentre si porta a livello logico alto se impostato ad 1.
- CPHA: determina il fronte di clock in cui il ricevente campiona il segnale di ingresso

Le modalità di funzionamento più spesso utilizzate dai dispositivi in commercio sono quelle con CPHA=CPOL=0 e con CPHA=CPOL=1. Il segnale CS viene utilizzato per l'abilitazione della porta seriale ed è controllato dal master SPI, ovvero il microcontrollore. CS diventa basso all'inizio della trasmissione e torna alto alla fine. Attraverso l'uso del software CubeMX, è stata selezionato e configurato il protocollo SPI abilitando la comunicazione SPI1 in "Connectivity" ed impostandolo in modalità Master Full-Duplex (Figura 2.6).

Bisogna successivamente configurare i pin di interrupt INT1 ed INT2 corrispondenti ai pin EXTI1 ed EXTI2, ed il pin PG2, al quale corrisponde il pin di output dove il sensore scrive i risultati delle misure di accelerazione.

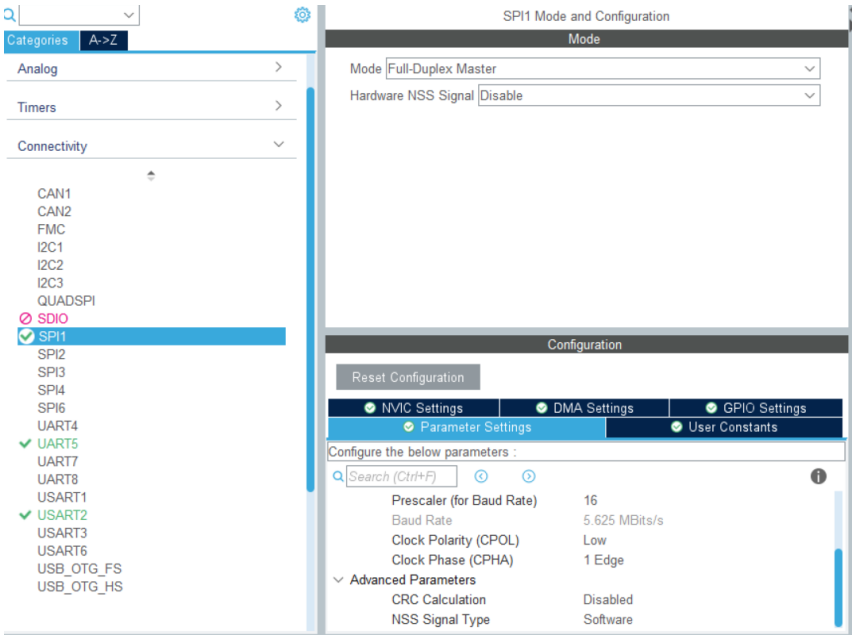


Figure 2.6: Configurazione SP1

2.2.2 Configurazione del segnale di clock

Dal diagramma a blocchi in figura 2.7, si può verificare che la CPU, attraverso il bus AHB MATRIX, può accedere alle linee AHBx/APBx per leggere e scrivere dati ad una velocità di 180MHz. Per ricevere la lettura delle misure di accelerazione, la CPU abilita la periferica relativa all'SP1 tramite il bus AHB/APB2 (che lavora a 90MHz) inviando un segnale di enable ed interrogandolo per salvare in memoria i dati. Attraverso CubeMX è possibile accedere al pannello che consente di configurare il segnale di clock abilitando la sorgente di quarzo esterna nella sezione RCC. Dal system Clock configuration è possibile poi impostare l'HSE(High-speed external) alla velocità di 24MHz.

2.2.3 Utilizzo del timer

L'utilizzo di una periferica timer risulta necessario. Supponendo infatti di voler accedere alle letture del sensore con frequenza pari a 5KHz, senza l'utilizzo di un timer quello che succederebbe è che si potrebbero verificare casi in cui si andrebbe ad interrogare il sensore attraverso l'SPI ad una velocità maggiore rispetto a quella con cui questo acquisisce le misure di accelerazione, con il risultato di ottenere o la stessa misura dell'istante precedente, oppure un valore NULL. Per sincronizzare le misure alla velocità di 5KHz viene quindi abilitato ed interrogato un timer attraverso il bus AHB/APB1, operante a 45MHz.

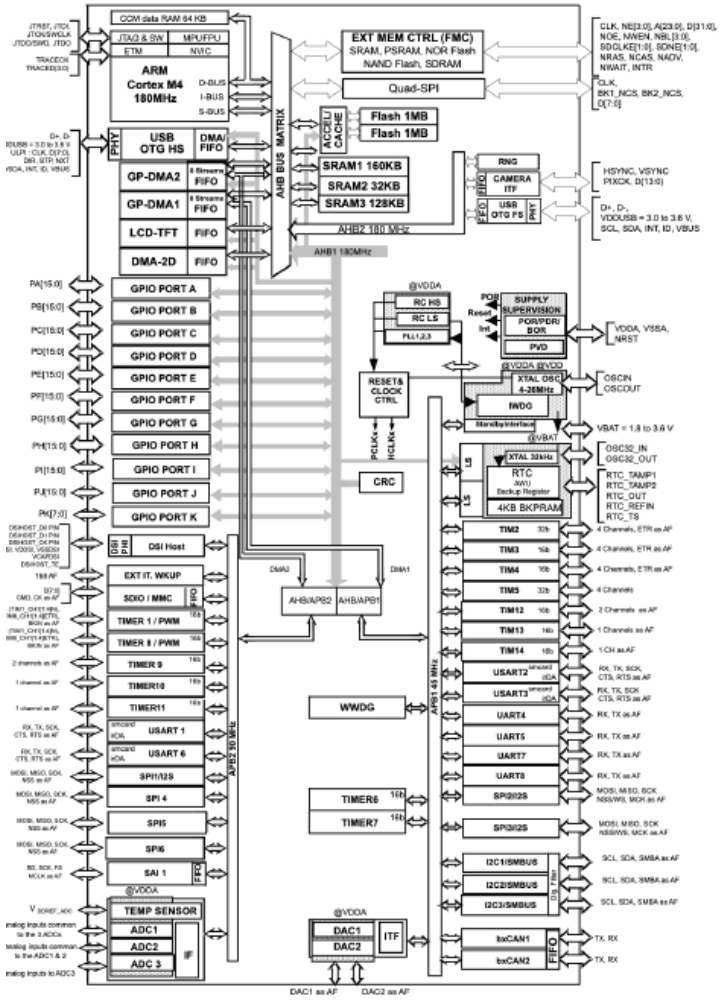


Figure 2.7: Diagramma a blocchiSTM32F469xx

Per generare dunque un segnale di sincronia alla frequenza scelta, bisogna impostare un numero di conteggi pari a 9000.

2.2.4 Interfaccia UART

L'interfaccia UART(Universal Asynchronous Receiver-Transmitter) è un dispositivo hardware che converte flussi di bit di dati da un formato parallelo ad uno seriale asincrono e viceversa. I due segnali di ciascun dispositivo UART sono denominati:

- Trasmettitore(Tx)
- Ricevitore(Rx)

L'UART trasmittente è collegato a un bus dati di controllo che invia i dati in forma parallela. Da qui, i dati vengono trasmessi sulla linea di trasmissione in serie, bit per bit, all'UART

ricevente. Questo, a sua volta, convertirà i dati seriali in parallelo per il dispositivo ricevente. Un dispositivo UART presenta quindi un pin di trasmissione e ricezione dedicato per l'una o l'altra. La velocità di trasmissione delle informazioni (baud rate) deve essere uguale per dispositivo ricevente e trasmittente. L'interfaccia UART non utilizza un segnale di clock per sincronizzare i dispositivi che comunicano, in quanto trasmette i dati in modo asincrono. Il punto di sincronizzazione è gestito avendo lo stesso baud rate su entrambi. Nel lavoro di tesi, la porta UART presente sulla sensor board che è stata abilitata è la UART5, e il baud rate impostato è pari a 230400.

2.3 Ambiente di programmazione STMCubeIDE

Una volta terminata questa fase preliminare sul CubeMX, bisogna spostarsi all'interno di un ambiente di programmazione, in questo caso STMCubeIDE. Da CubeIDE è possibile creare un nuovo progetto partendo dalla configurazione appena determinata su CubeMX. In questo modo verranno generate automaticamente le librerie necessarie per inizializzare ed utilizzare le periferiche configurate.

Per effettuare le operazioni di misura con il sensore è necessario inviare un'istruzione tramite SPI (Transmit) contenente l'indirizzo che indica l'operazione da svolgere. Per fare un esempio immediato, quando viene definita la variabile spiSndX[0] = 0x29—0x80, quello che viene comunicato tramite la funzione transmit è di eseguire l'operazione descritta dal registro 29, ovvero quella che si occupa della lettura degli 8 bit più significativi relativi all'accelerazione lungo l'asse X. Essendo un'operazione di lettura va settato il bit più significativo ad 1; da qui l'utilizzo dell'OR con il valore 0x80. In modo del tutto analogo, per ricevere il risultato di misura bisogna utilizzare un'istruzione tramite SPI di Receive, così da memorizzare all'interno di una struttura dati definita dall'utente i valori delle misure di accelerazione ricavate dall'accelerometro. Questa operazione va eseguita poi per ognuno dei tre assi. Bisogna inoltre evidenziare l'importanza del timer, che attraverso un flag (flag elapsed) assicura che le misure vengano ogni inverso della frequenza di campionamento impostata dall'utente. L'ultimo passo consiste nel visualizzare a schermo i risultati delle operazioni di misura attraverso l'interfaccia UART, passando dalla sensor board alla debugging board attraverso l'utilizzo di un cavo SWD e dell'adapted board così da poterli leggere sulla com seriale di un PC.

Chapter 3

Elaborazione dei dati attraverso la FFT

Lo strumento matematico scelto per elaborare le misure di accelerazione ricavate attraverso il kit utilizzato è la trasformata di Fourier veloce. Il motivo dietro la scelta della FFT come strumento matematico di elaborazione dei dati risiede nel fatto che l'analisi nel dominio della frequenza dei campioni raccolti è in grado di fornire informazioni molto dettagliate riguardo lo stato in cui il motore si trova ad operare. Molte anomalie nel funzionamento dei motori elettrici, infatti, sono il riflesso di ampiezze altrettanto anomale in corrispondenza di specifiche frequenze o intervalli di frequenze, che possono essere analizzati per rilevare eventuali condizioni di malfunzionamento o di guasto.

3.1 La DFT

Prima di approfondire il discorso relativo alla FFT, tuttavia, bisogna introdurre alcune nozioni preliminari. Dal momento che stiamo lavorando con un insieme discreto di campioni, ovvero le N misure di accelerazione ricavate dal sensore, avremo un range di frequenze testabili che varia tra 0Hz (la componente continua del segnale in ingresso) e la frequenza di Nyquist, ovvero $F_n = F_s/2$, con $Fs =$ frequenza di campionamento (nel nostro caso pari a 20KHz),che determina un limite superiore oltre il quale non è possibile andare. Poiché abbiamo un set di campioni finito pari ad N, lo strumento matematico di cui si ha bisogno non è la trasformata di Fourier, che lavora con un segnale in ingresso definito su un insieme tempo-continuo, ma la trasformata di Fourier discreta, ovvero la DFT. La DFT lavora su collezioni di dati discrete raccolti entro una finestra temporale di durata finita. La relazione che ci permette di determinare la durata della finestra temporale quando andiamo a raccogliere un numero N di campioni con una certa

frequenza di campionamento è $T = N\frac{1}{Fs}$. Se N fosse uguale a 20000, ad esempio, parlare di una finestra temporale di 1 secondo o di 20000 campioni raccolti a frequenza 20KHz sarebbe equivalente. Se abbiamo N campioni prelevati a frequenza F_s, otteniamo in uscita dalla DFT un numero discreto di oscillazioni sinusoidali. Lo spettro risulta quindi costituito da componenti sinusoidali a frequenze che variano tra 0 ed F_n, la cui distanza è pari a $\Delta f = \frac{F_n}{N/2}$, grandezza definita come risoluzione spettrale. L'equazione che mostra la definizione matematica della DFT è la seguente:

$$X(k) = \sum_{n=0}^{N-1} x(n) e^{\frac{-2\pi i n k}{N}}$$

dove

- $x(n)$: il segnale tempo discreto.
- N: il numero di campioni
- n: l'indice tempo-discreto
- k: l'indice in frequenza

È possibile rendere più compatta l'equazione appena descritta definendo il seguente termine:

$$W_N = e^{\frac{-2i\pi}{N}}$$

3.2 L'algoritmo di ottimizzazione della FFT

La DFT è implementabile su un calcolatore, in quanto richiede un numero finito di passaggi per essere eseguita, il che la rende uno strumento di elaborazione digitale dei segnali estremamente utile. Il grande svantaggio nell'utilizzarla, tuttavia, risiede nel fatto che il tempo di calcolo necessario per eseguirla è direttamente proporzionale ad N^2. il che la rende molto inefficiente. Ecco perché, al posto della DFT, si preferisce utilizzare una versione ottimizzata, la FFT. La famiglia di algoritmi di ottimizzazione sviluppati per realizzare l'FFT consta di un gran numero di procedure atte a rendere più efficiente la DFT. Queste si dividono principalmente in due categorie: algoritmi di decimazione in tempo (DIT) ed algoritmi di decimazione in frequenza (DIF). La differenza principale tra queste due tipologie di algoritmi è che nel caso della decimazione in tempo, l'input viene decomposto in sottosequenze , che

dividono la sequenza di campioni $x(n)$ in campioni pari e dispari. Nel caso della decimazione in frequenza, invece, è l'output che viene decomposto secondo questa procedura. Nel primo caso, quindi, l'operazione di divisione viene eseguita nel dominio del tempo, mentre nel secondo in quello della frequenza. Alla base di questi algoritmi vi è l'assunzione che il numero N di campioni sui quali si va ad eseguire la trasformata sia una potenza di 2. Al giorno d'oggi esistono delle procedure e delle routine per il calcolo di FFT la cui lunghezza non è un multiplo di 2, ma che sono meno efficienti e richiedono maggiore potenza di calcolo. Nel caso del lavoro di tesi, lavorando con un'architettura embedded, si è scelto di utilizzare un algoritmo basato su FFT di lunghezza pari ad un multiplo di 2. La procedura di ottimizzazione utilizzata per implementare la FFT è l'algoritmo DIF radix-2. Questa procedura si basa su un approccio del tipo divide-and-conquer: la DFT viene divisa in due sommatorie, la prima delle quali viene computata per i primi N/2 punti, e la seconda per la restante metà, ottenendo la seguente relazione:

$$X(k) = \sum_{n=0}^{(N/2)-1} x(n)W_N^{kn} + \sum_{n=N/2}^{N-1} x(n)W_N^{kn}$$

Sostituendo nella seconda sommatoria $n = n + \frac{N}{2}$ è possibile ricondursi ad una forma compatta

$$X(k) = \sum_{n=0}^{(N/2)-1} x(n)W_N^{kn} + W_N^{KN/2} \sum_{n=0}^{(N/2)-1} x(n + \frac{N}{2})W_N^{kn}$$

È possibile verificare che $W_N^{kN/2} = (-1)^k$, e mettendo insieme le 2 sommatorie si ottiene questa relazione:

$$X(k) = \sum_{n=0}^{(N/2)-1} [x(n) + (-1)^k x(n + \frac{N}{2})]W_N^{kn}$$

A questo punto avviene la decomposizione dell'uscita nelle due sequenze di campioni pari e dispari.

$$X(2r) = \sum_{r=0}^{(N/2)-1} [x(r) + x(n + \frac{N}{2})]W_N^{2rk}$$

$$X(2r+1) = \sum_{r=0}^{(N/2)-1} [x(r) - x(n + \frac{N}{2})]W_N^{(2r+1)k}$$

Per eseguire queste semplificazioni basta verificare che $(-1)^{2r} = 1$ e che $(-1)^{(2r+1)} = -1$. Questo procedimento di decimazione può essere iterato un numero $d = log_2 N$, cosicché alla fine avremo d stadi di decimazione. Quello che fa l'algoritmo radix-2 è dunque dividere l'intero processo computazionale della DFT in processi più piccoli di moltiplicazioni ed addizioni, chiamati "radix-2 butterflies" (Figura 3.1).

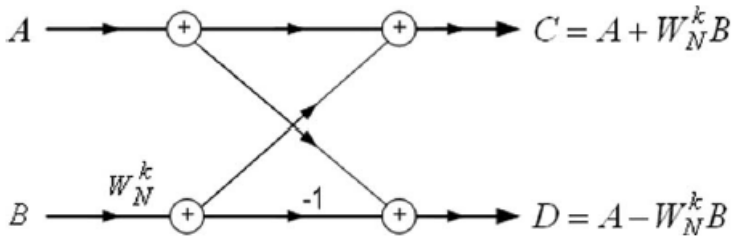


Figure 3.1: Radix-2 butterfly

Ad ogni stadio di decimazione avremo un numero pari ad $\frac{N}{2}$ di butterflies come quello mostrato in figura. Quello che bisogna evidenziare dell'applicazione di questo algoritmo di ottimizzazione è che attraverso il procedimento di decimazione in frequenza il numero di moltiplicazioni complesse da eseguire si riduce ad $\frac{N}{2} log_2 N$, mentre il numero di addizioni complesse ad $N log_2 N$, riducendo di molto il costo computazionale della DFT.

3.3 Utilizzo delle librerie DSP

Attraverso l'utilizzo di CubeMX, che consente di scaricare il plugin X-CUBE ALGOBUILD,è possibile implementare all'interno del progetto la libreria software CMSIS DSP. All'interno della libreria sono presenti diverse funzioni, ognuna delle quali copre una categoria specifica. Tra queste sono presenti anche quelle che permettono di implementare la FFT. U Una volta inclusa la libreria, bisogna definire il numero di campioni rispetto ai quali viene eseguita la FFT (che deve essere una potenza di 2) e la dimensione del vettore che ospiterà i risultati della trasformata, che sarà pari alla metà del numero di campioni. Questo perché in uscita dalla FFT avremo $N/2$ parti reali ed $N/2$ parti complesse di cui però bisogna effettuare il modulo, per cui l'output sarà costituito da $N/2$ valori. Nel caso preso in esame il numero di campioni scelto è $N = 2048$. L'ultima grandezza che resta da definire è la risoluzione spettrale che, come già visto in precedenza, è pari a:

$$\Delta f = \frac{F_n}{N/2}$$

Lo step sul quale bisogna soffermarsi è l'inizializzazione del modulo CFFT/CIFFT (Figura 3.2). La funzione di inizializzazione, oltre a prendere in la lunghezza della FFT, permette di settare altri due parametri. Il primo è l'ifft flag, che permette di impostare la direzione della trasformata; se settato ad 1 consente di eseguire la trasformata inversa. Il secondo, invece, è il bitReverseFlag. Se si assegna 1 a questo parametro, viene abilitata l'ordinamento inverso in termini di bit dell'output.

```
/* Inizializzo il modulo CFFT/CIFFT, ifftFlag = 0, doBitReverse = 1 */
arm_cfft_radix2_instance_f32 S;
arm_cfft_radix2_init_f32(&S, FFT_SIZE, 0, 1);
```

Figure 3.2: Inizializzazione modulo CFFT/CIFFT

Il motivo per cui questa operazione è richiesta si evidenzia osservando il diagramma che illustra in modo in cui opera il radix-2 (Figura 3.3).

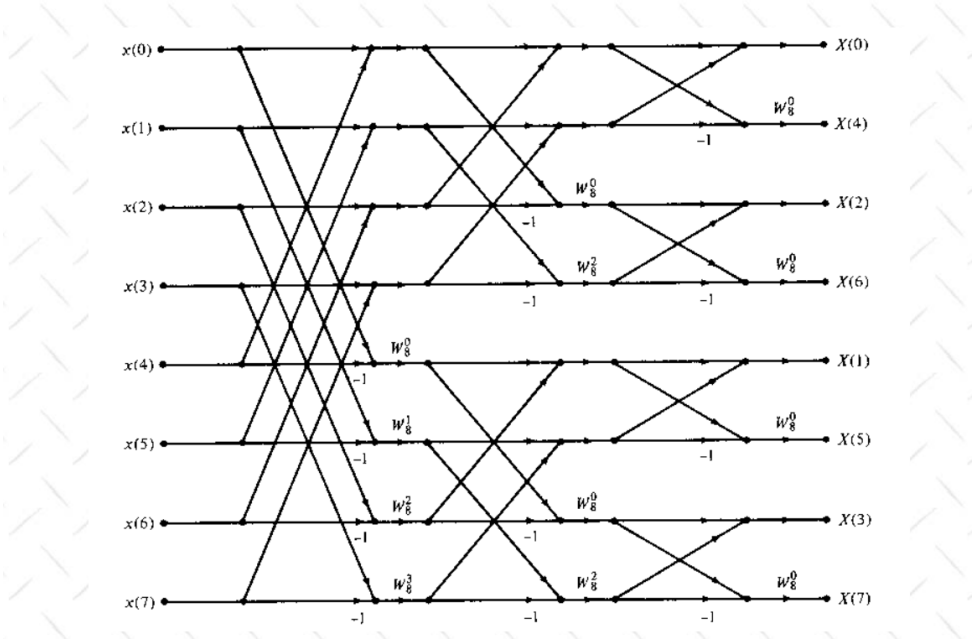


Figure 3.3: Decimazione in frequenza con N=8

Mentre la sequenza dei campioni in input è in ordine naturale, l'output è in ordine di bit inverso (Figura 3.4), per cui risulta necessario settare il bitReverseFlag ad 1.

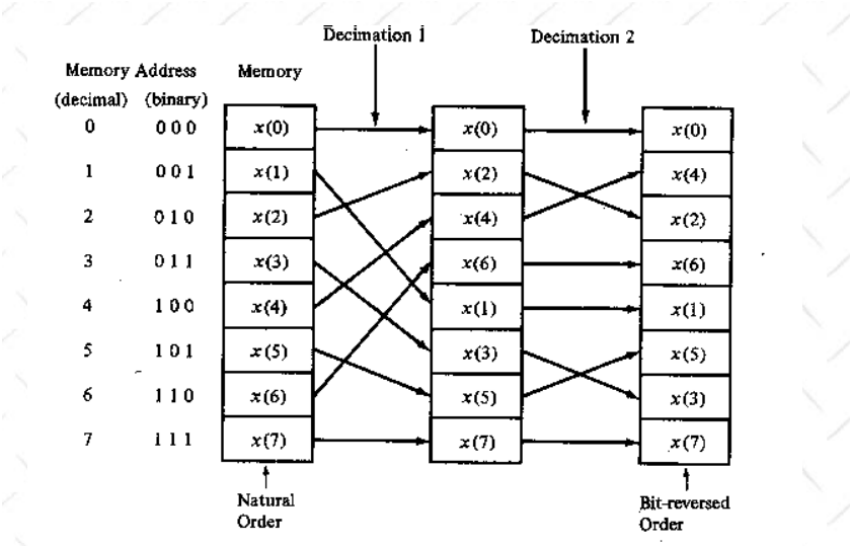


Figure 3.4: Ordine di bit dell'output del radix-2

A questo punto non resta che immagazzinare i dati che l'FFT dovrà processare all'interno di un buffer di input. Il modulo CFFT/CIFFT tratta il vettore in ingresso come se questo fosse diviso in coppie di valori, una parte reale ed una immaginaria, per cui all'atto in cui riempiamo il vettore bisogna opportunamente settare a zero la parte immaginaria, dal momento che i dati di input sono numeri reali. Fatto questo è possibile eseguire la FFT attraverso il radix-2, per poi usare una funzione apposita presente all'interno della libreria per ottenere i valori del moduli dei numeri complessi in uscita (Figura 3.5).

```
//Raccolgo i campioni

for(int j=0;j<SAMPLES;j+=2){
//Parte reale
fft_in_buf_X[j] = ((float32_t)((float32_t)Vettx[j]));
//Parte immaginaria
fft_in_buf_X[j+1] = 0;
}

/* Processo i dati attraverso il modulo CFFT/CIFFT */
arm_cfft_radix2_f32(&S, fft_in_buf_X);

/* Processo i dati attraverso il modulo Complex Magnitude */
arm_cmplx_mag_f32(fft_in_buf_X, fft_out_buf_realX, FFT_SIZE);
```

Figure 3.5: Elaborazione campioni tramite il modulo CFFT/CIFFT

3.4 Limiti della FFT

La FFT risulta essere un grande miglioramento in termini di efficienza computazionale rispetto alla DFT. Tuttavia ci sono dei limiti che caratterizzano l'uso della FFT e l'accuratezza dei risultati che ne derivano, e sono principalmente legati al fenomeno della dispersione spettrale (spectral leakage). Come accennato in precedenza, quando utilizziamo la FFT, che è il risultato di un algoritmo di ottimizzazione della DFT, abbiamo in ingresso una collezione di campioni raccolti con una certa frequenza di campionamento F_S entro una finestra temporale la cui durata è finita. Vi sono quindi tre fattori che sono di importanza fondamentale per capire come interpretare i valori di uscita in termini di ampiezze e frequenze corrispondenti quando viene eseguita una FFT: la finestra temporale di osservazione, il numero di campioni raccolti e la frequenza di campionamento. Consideriamo un caso esemplificativo molto semplice, una sinusoide di ampiezza unitaria che ha frequenza pari a 10 Hz che viene campionata ad una frequenza di 100Hz e con una finestra di osservazione pari ad 1 secondo. Se riprendiamo la formula usata in precedenza per il calcolo della finestra temporale e ribaltandola per calcolare il numero di campioni raccolti otteniamo $N = T \cdot F_s = 100$. A questo punto consideriamo due casi: il primo è quello in cui eseguiamo una FFT su 100 punti, che corrisponde esattamente alla quantità di campioni del segnale, mentre nel secondo utilizziamo la FFT assumendo $N = 128$, ovvero la potenza di 2 più vicina.

Osservando i due plot (Figura 3.6), si può osservare come nel secondo caso si ha una situazione essenzialmente diversa dalla prima. Quello che ci si aspetta infatti è di avere due impulsi a frequenza 10 e -10, ognuno dei quali con ampiezza pari a 0.5, che è quello che succede nel secondo caso. Quando invece eseguiamo l'FFT con $N = 128$ campioni, osserviamo che ci sono dei valori di ampiezza diversi da 0 in corrispondenza di frequenze diverse da 10 e -10. Questo fenomeno è dovuto alla differente risoluzione spettrale, che nel secondo caso è pari a 0.78125 e non ad 1 come nel primo. Poiché la risoluzione spettrale non è unitaria, sull'asse x non si potrà mai avere una frequenza pari esattamente a 10Hz. Il massimo che si può fare è prendere le due frequenze più vicine, che in questo caso sono 9.375Hz e 10.15Hz. Ecco perché attorno a 10Hz (e a -10Hz) sembra che lo spettro si sia "distribuito".

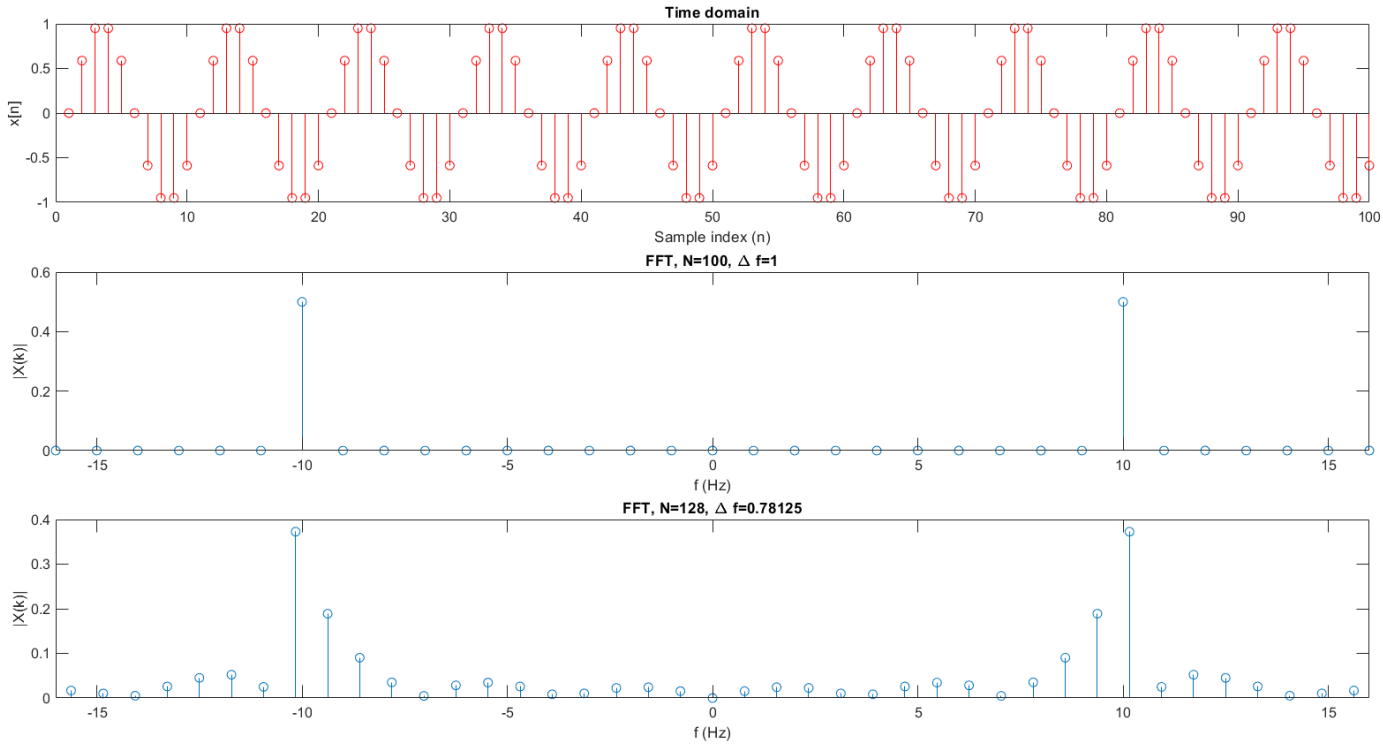


Figure 3.6: Esempio di spectral leakage

Chapter 4

Lo sviluppo di un modello di apprendimento automatico

La parte di acquisizione e di elaborazione delle misure di accelerazione rappresenta lo step preliminare per la realizzazione di un modello di apprendimento automatico. Tuttavia, prima di parlare dello sviluppo di tecniche di machine learning o deep learning, bisogna prima inquadrare correttamente il problema da affrontare. In primis bisogna definire la tipologia di apprendimento, che nel caso del lavoro di tesi rientra nel paradigma dell'apprendimento supervisionato. Questo implica che il modello necessita di un dataset contenente due tipologie di dati:

- i dati di input: sono i dati relativi alle misure di accelerazione acquisite con il sensore e processate attraverso la FFT, e contengono le informazioni riguardo lo stato di funzionamento del motore.
- i dati di output: corrispondono alle etichette che vengono assegnate ai dati di input, e che serviranno al modello per comprendere la relazione che li lega agli ingressi.

Questo dataset viene quindi utilizzato per addestrare il modello (da qui il nome di training set). Un ulteriore specifica da chiarire è relativa al tipo di problema che si va ad affrontare. Se si vuole monitorare lo stato di salute di un motore, ci si potrebbe chiedere ad esempio se questo sta lavorando in condizioni di funzionamento ordinario o meno. Questo caso è modellabile come un problema di classificazione binaria, in cui l'uscita del modello di apprendimento potrebbe essere 0 nel caso di funzionamento normale, 1 nel caso di funzionamento anomalo. Se invece si volesse conoscere qual è la vita utile del motore se questo lavorasse sotto certe condizioni di

stress per un certo periodo di tempo, il problema non sarebbe più modellabile come nel caso precedente. Ecco quindi che bisogna operare una distinzione in:

- Problemi di classificazione: l'uscita del modello è una variabile categorica, il cui insieme di valori possibili è costituito da un insieme finito di categorie
- Problemi di regressione: l'uscita può assumere valori continui

Nel lavoro di tesi è stato affrontato un problema di classificazione multi-classe. Definito quindi nel dettaglio il tipo di approccio che viene utilizzato, bisogna costruire il modello di apprendimento.

4.1 Machine learning e Deep learning

Quando si utilizza la parola machine learning, ci si riferisce ad una branca dell'intelligenza artificiale che si propone di sviluppare sistemi che siano in grado di imparare dall'esperienza senza essere programmati in modo esplicito. La chiave risiede nel fornire in ingresso al modello di ML dataset di grandi dimensioni contenenti informazioni utili relative alla particolare applicazione per la quale esso viene progettato.

Per quanto gli algoritmi di machine learning abbiano riscosso un enorme successo in molti campi d'applicazione, ve ne sono alcuni nei quali sembrano fare più fatica, come ad esempio la classificazione delle immagini. In generale, i campi applicativi nei quali non eccellono sono quelli nei quali bisogna lavorare con tipi di dato non strutturato, cioè dati che non sono organizzati entro modelli o schemi predefiniti. Ecco allora che una categoria degli algoritmi di ML, gli artifical neural networks (ANN), diventa particolarmente interessante. Essi traggono ispirazione dal modo in cui il nostro sistema nervoso elabora e processa le informazioni. Alla base della loro struttura vi è l'unità di elaborazione fondamentale, il neurone artificiale. I neuroni artificiali sono connessi tra loro, ed ogni connessione trasmette dei segnali la cui intensità viene modulata in base ad un fattore di peso che viene modificato durante il processo di apprendimento (Figura 4.1).

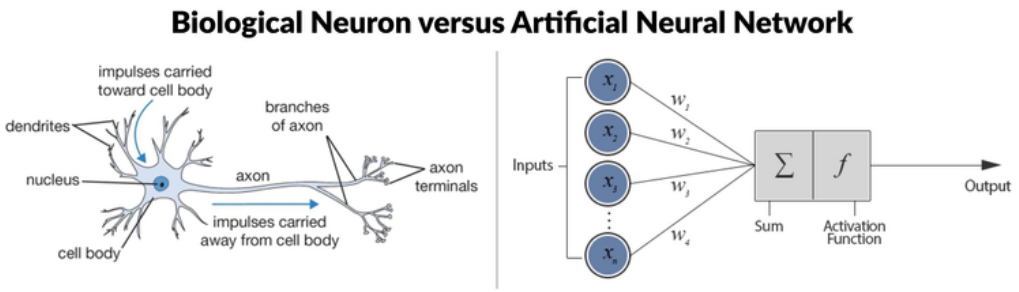


Figure 4.1: Comparazione tra neurone e neurone artificiale

Un ANN è organizzato a strati: vi è uno strato di input che riceve i dati in ingresso, e uno di output che restituisce il risultato. Tra i due possono essere presenti zero o più strati, che vengono definiti ”hidden layers” (strati nascosti). Si tratta di layer intermedi, che eseguono elaborazioni matematiche attraverso quella che viene definita funzione di attivazione. Sviluppando la teoria dietro la costruzione di modelli di apprendimento che cercassero di imitare il funzionamento del cervello umano, si è giunti a progettare quelli che vengono definiti Deep Neural Network (DNN) (Figura 4.2). A differenza degli ANN più semplici, i DNN adoperano più di un hidden layer, ed implementano neuroni artificiali più complessi. Questa ricerca ha aperto le porte al deep learning, che può essere interpretato come la capacità dei DNN di ricevere in input dei dati ”grezzi” e riconoscere lo schema entro il quale incasellare questi dati.

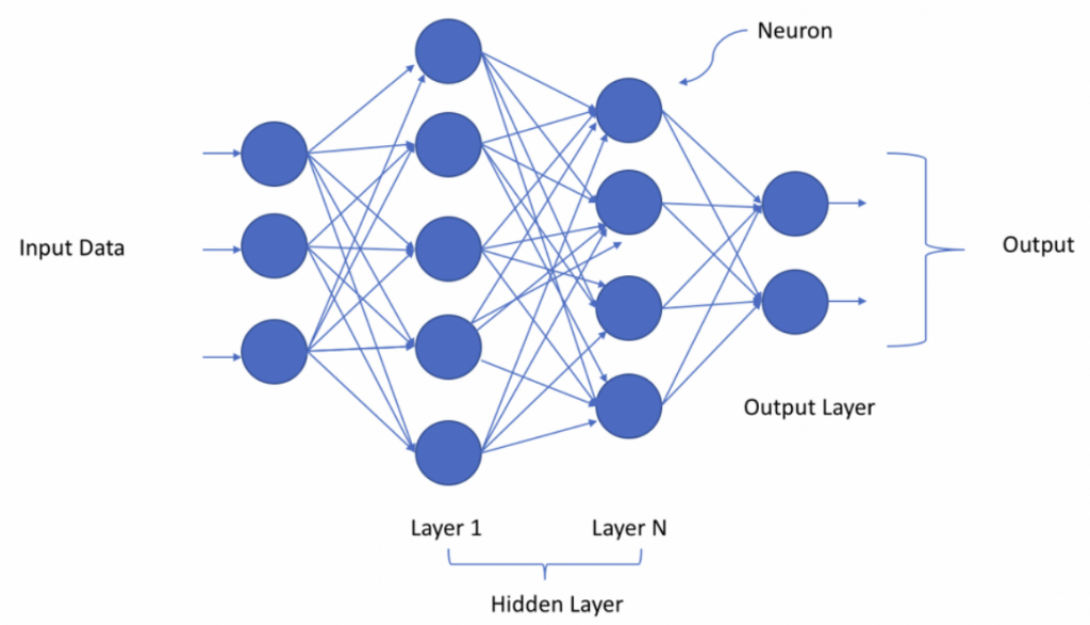


Figure 4.2: Deep Neural Network

4.2 Creazione di un modello sequenziale

Lo sviluppo del modello di rete neurale è avvenuto in ambiente Python e la piattaforma utilizzata è stata Google Colab. La prima cosa da fare è la scelta delle features al fine di determinare il numero di varibiali di ingresso del modello di rete neurale. Si è optato per 6 variabili di input, ovvero 3 coppie ampiezza-frequenza; in particolare sono state prelevate le 5 ampiezze maggiori per ognuno dei 3 assi e le corrispondenti frequenze. A questo punto bisogna costruire un dataset sul quale lavorare. Attraverso l'interfaccia UART, che consente la comunicazione tra il PC e il microcontrollore, è possibile ricevere i dati relativi alle misure di accelerazione acquisite dal sensore visualizzati dalla COM del monitor seriale. I dati vengono importati su Excel, dove viene costruita una tabella rispettando il formato attraverso il quale i dati verranno passati al modello di network neurale. Avremo quindi una tabella costituita da un numero di righe pari ad n, ovvero il numero di campioni acquisiti, e 7 colonne (6 per le ampiezze e le frequenze ed una per le etichette). Etichettati correttamente i dati acquisiti, bisogna importare il dataset all'interno di Python. Ciò è reso possibile grazie alla libreria Panda, che mette a disposizione funzioni studiate per analizzare e manipolare i set di dati. A questo punto bisogna dividere il dataset, separando le variabili di ingresso dalle etichette, ovvero le variabili di uscita. Divisi quindi input ed output, è il momento di andare a costruire il modello vero e proprio.

La rete neurale è stata sviluppata utilizzando Keras, una libreria software di alto livello che mette a disposizione dell'utente le funzioni necessarie per costruire un modello di NN.

In particolare, per il lavoro di tesi si è scelto di creare un modello di tipo sequenziale. Si tratta di un modello semplice, nel quale i vari layer della rete sono in ordine sequenziale, da cui il nome. L'aggiunta dei layer avviene tramite l'utilizzo del metodo add. Ogni layer aggiunto è un Dense layer, il tipo più comune e più utilizzato. Inoltre, per il primo layer bisogna specificare la dimensione dell'input. Per ogni strato bisogna indicare il numero di neuroni che lo compone, nonché la funzione di attivazione. Per capire cos'è una funzione di attivazione, è possibile fare un'analogia con il cervello umano: quando riceviamo un gran numero di informazioni, il nostro cervello cerca di analizzarle e di classificarle come informazioni utili o non utili. Una funzione di attivazione, con le dovute proporzioni, realizza proprio questa cosa: se l'input che riceve la funzione è sufficientemente grande da superare una certa soglia, quello che accade è che questa "spara", altrimenti non fa niente. Essenzialmente, una funzione di attivazione decide se un neurone deve essere attivato o meno, determinando quello che viene definito "firing pattern". Bisogna però capire chi è l'input della funzione di attivazione. Quando un neurone riceve in

ingresso un numero n di valori in ingresso, questo esegue una trasformazione lineare utilizzando i pesi associati alle diverse features ed un fattore costante chiamato ”bias” (Figura 4.3).

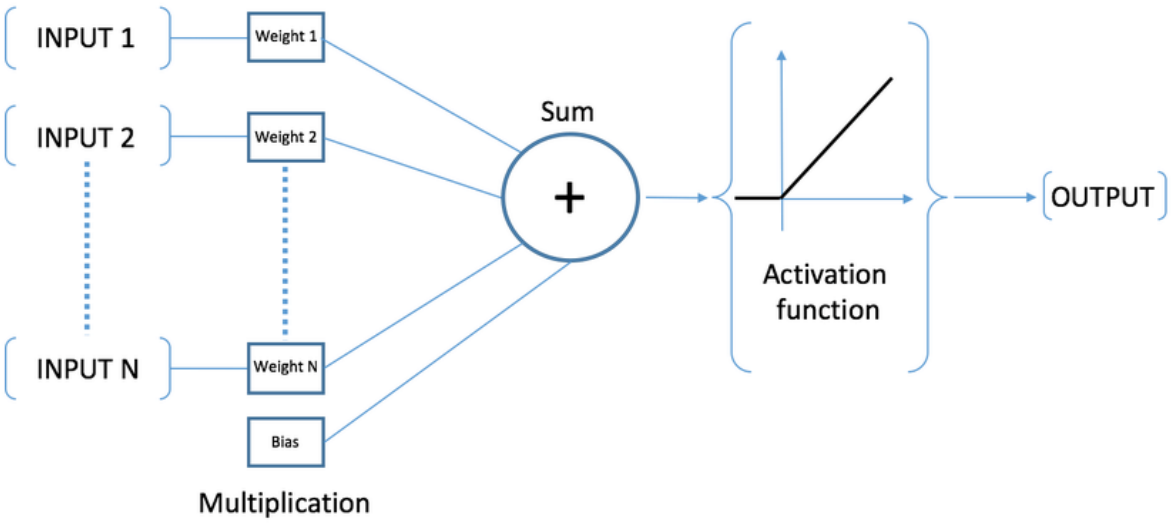


Figure 4.3: Elaborazione dei dati del neurone artificiale

Se non utilizzassimo alcuna funzione di attivazione, il neurone non saprebbe discernere se l'informazione che ha processato è utile o meno. È a questo punto che viene applicata la funzione di attivazione, attraverso la quale viene elaborato il risultato della trasformazione lineare, e questo rappresenta l'output del neurone. Senza l'utilizzo di una funzione di attivazione, quello che accadrebbe è che, indipendentemente da quanti hidden layer vengono aggiunti, ogni neurone di ogni layer esegue una trasformazione lineare usando i pesi e il bias. L'output di quel neurone diverrà l'input di quello del layer successivo, che eseguirà la stessa operazione. La composizione di due funzioni lineari, tuttavia, continua ad essere una funzione lineare, per cui tutti gli hidden layers si comporteranno alla stessa maniera. In questo modo è vero che si semplifica la rete neurale, ma gli si impedisce di imparare pattern complessi dai dati che ha a disposizione. Questo è reso possibile grazie all'uso di funzioni di attivazione che introducono una non linearità.

4.2.1 Tuning degli iperparametri

Arrivati a questo punto resta ancora una questione aperta: quanti hidden layers bisogna inserire, e quanti neuroni dovrebbe avere ognuno di essi? In realtà questi non sono gli unici aspetti che sono stati trascurati. Quando si va ad addestrare il modello, infatti, non basta solo fornirgli in ingresso il training set. Esiste una serie di variabili che bisogna tenere in considerazione

che regolano il processo di addestramento: gli iperparametri. Il numero di hidden layers, ad esempio, è un iperparametro. Il problema è che, differentemente dai parametri del modello, che sono strettamente legati ai dati rispetto ai quali viene addestrato, gli iperparametri sono variabili di configurazione. C'è un'ampia gamma di iperparametri, e senza una procedura automatica per determinare qual è la migliore combinazione da utilizzare, bisognerebbe provare empiricamente quale di queste ottiene il risultato migliore.

La procedura per selezionare gli iperparametri ottimi per il nostro particolare modello è la gridsearch (griglia di ricerca). L'idea alla base di questa tecnica è molto semplice: si costruisce lo scheletro del modello che si intende ottimizzare, definendo quindi i parametri che lo caratterizzano (Figura 4.4), e per ogni iperparametro o sottoinsieme di iperparametri si va ad eseguire una ricerca automatica esaustiva per cercarne la combinazione migliore.

```
#Definisco il modello da usare per la grid_search

def define_model():
    my_model = Sequential()
    my_model.add(Dense(10,activation = 'relu',input_dim = 6))
    my_model.add(Dense(10,activation = 'relu'))
    my_model.add(Dense(3,activation = 'softmax'))
    my_model.compile(metrics = ['accuracy'])
    return my_model
```

Figure 4.4: Definizione scheletro del modello

Questo è reso possibile grazie alla definizione di un "dizionario" (Figura 4.5), ovvero un insieme finito di valori entro il quale l'iperparametro può variare.

```
#Eseguo il tuning per la batch_size e gli epochs. Definisco i range in cui varieranno i  parametri

my_model = KerasClassifier(model=define_model,verbose = 0)
epochs = [10,50,100]
batch_size = [10,20,40,60,80,100]

#Definisco il m dizionario

param_grid = dict(epochs=epochs, batch_size=batch_size)

#Eseguo la gridsearch(Per ottenere risultati non banali devo eseguirla più di una volta)

grid = GridSearchCV(estimator=my_model, param_grid = param_grid, n_jobs=-1, cv =5)
grid.fit(X_train, Y_train)
```

Figure 4.5: Esempio di gridsearch

In questo modo, ad esempio, è possibile cercare quanti neuroni per layer è meglio inserire, o qual è il miglior optimizer per quel particolare modello. L'implementazione della gridsearch

è stata possibile grazie alla libreria scikeras, che permette di costruire un ponte tra Keras e Scikit-learn, una delle librerie open source più note che permettono di costruire modelli di apprendimento automatico.

Tra gli iperparametri per i quali è stata effettuata la gridsearch vi sono:

- Batch size: corrisponde al numero di campioni che vengono propagati alla rete neurale. Se ad esempio in ingresso alla rete vi sono 1000 campioni e la batch size è pari a 10, vengono prelevati i primi 10 campioni e il network viene addestrato, poi i secondi 10 e così via.
- Epochs: corrisponde al numero di volte in cui l'algoritmo di apprendimento scorrerà lungo l'intero dataset.
- Optimizers: gli "ottimizzatori" sono algoritmi o procedure utilizzati per modificare gli attributi della rete, come i pesi o il tasso di apprendimento.
- Numero di neuroni per layer: questo è l'iperparametro che richiede la maggiore potenza computazionale per essere ricavato. Basti pensare a quante possibili combinazioni di neuroni è possibile trovare se si utilizzano anche solo 2 hidden layers e si definisce un dizionario in cui il numero di neuroni varia, ad esempio, tra 1 e 25.
- Funzione di attivazione

4.3 Risultati dell'addestramento

La struttura del modello costruito consta di 2 layer nascosti, ai quali bisogna aggiungere il layer di input e quello di output; attraverso la gridsearch è possibile determinare la combinazione migliore di iperparametri. Il numero di neuroni del layer di input e di quelli nascosti è pari a 25, mentre la batch size e gli epochs sono rispettivamente 10 e 100. Per quanto riguarda la funzione di attivazione, quella che ha dimostrato di ottenere risultati migliori in fase di addestramento è una funzione non lineare chiamata ReLu (rectifier linear unit)(1):

$$f(x) = max(0, x) = \begin{cases} x & x > 0 \\ 0 & x \leq 0 \end{cases} \tag{1}$$

Sia per il layer di input che per tutti quelli successivi la regola di attivazione utilizzata è stata la ReLu(Figura 4.6). L'ultimo layer ha un numero di neuroni pari al numero di classi che il classificatore deve distinguere, ed una regola di attivazione differente, definita Softmax. Questa riceve in input un vettore contenente K valori reali e restituisce in uscita un vettore di K numeri reali la cui somma complessiva è 1, che possono quindi essere interpretati come probabilità. Quello che accade infatti è che quando si lavora con modelli di rete neurale con diversi hidden layers, il penultimo layer manda in uscita al layer di output valori che non sono correttamente scalati e con i quali può risultare difficile lavorare. In questi casi la Softmax risulta estremamente utile, in quanto converte questi valori in una distribuzione di probabilità normalizzata, che può essere facilmente visualizzata dall'utente.

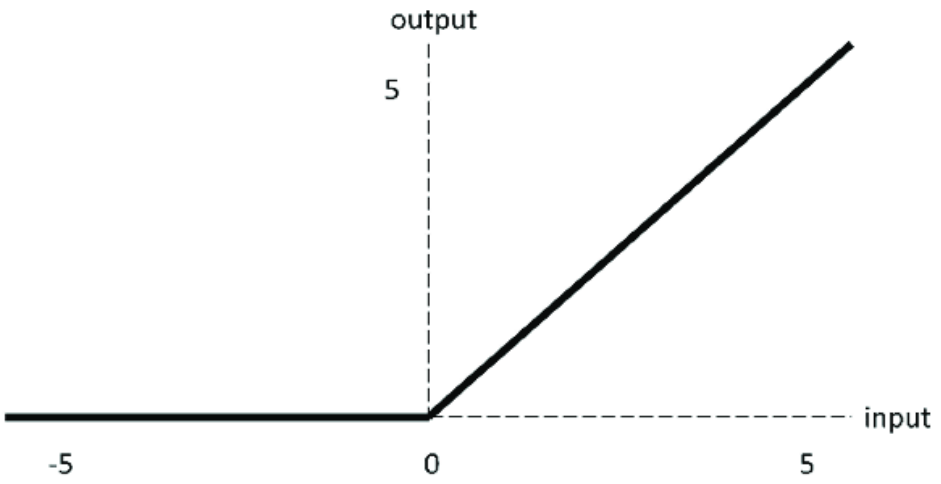


Figure 4.6: Grafico della ReLu

A questo punto non resta che compilare il modello attraverso il metodo compile. Il metro di giudizio che si è deciso di adottare nel lavoro di tesi per dare una stima della bontà del modello di rete neurale è l'accuracy, che va specificato all'atto della compilazione del modello. Utilizzando il metodo fit, che riceve in ingresso il training set opportunamente diviso in coppie ampiezza-frequenza e le relative etichette, è possibile addestrare il modello. Con il metodo evaluate, infine, si può valutare l'accuracy del modello rispetto al set di dati sul quale è stato addestrato.

Bisogna ora soffermarsi su un particolare molto interessante. Le etichette che vengono assegnate alle varie sestuple sulla base delle quali viene addestrato il modello sono dei valori scalari che permettono di codificare un certo stato di operatività del motore. Supponiamo ad

esempio di voler distinguere tre casi:

- Motore spento
- Motore in condizioni di funzionamento ordinario
- Motore in condizioni di funzionamento anomalo

Ipotizziamo allora di assegnare delle etichette ad ognuno di questi stati di funzionamento: 0 per il primo, 1 per il secondo e 2 per il terzo. A questo punto c'è una domanda che dovrebbe sorgere spontanea: se la softmax restituisce in output un vettore di K scalari il cui valore è compreso tra 0 e 1 (in questo caso specifico $K = 3$), com'è possibile riconoscere quando il motore si trova allo stato di funzionamento 2? La risposta a questa domanda risiede proprio nel modo in cui la softmax restituisce l'uscita, ovvero come una distribuzione normalizzata di probabilità. Se si intende addestrare il modello dandogli in ingresso le etichette relative alle varie misure di accelerazione, queste devono essere codificate nello stesso identico modo in cui la softmax restituisce l'output del modello: come una distribuzione normalizzata di probabilità. Ecco perché non è possibile addestrare il modello se non si manipolano prima le labels in modo tale da codificarle in un formato adatto alla funzione di attivazione che stiamo utilizzando. Attraverso il metodo *to_categorical*, appartenente alla libreria *utils* di Keras, è possibile trasformare un vettore colonna di scalari in una matrice binaria che rappresenta l'input della funzione:

Si evidenziare il modo in cui vengono mappati gli scalari all'interno della matrice binaria: dipendentemente dalla posizione dell'1, si possono distinguere 3 combinazioni differenti, che corrispondono allo 0, all'1 o al 2. Alla luce di quanto detto, è chiaro come va interpretato questo output: ognuno di quegli scalari indica la probabilità che un'etichetta sia pari ad un certo valore in base alla posizione che esso occupa. Poiché siamo in fase di addestramento del modello, è naturale che sapremo con certezza a quale risultato corrisponde ogni ingresso, in quanto siamo noi ad etichettarli. Nel caso dell'etichetta 0, ad esempio, avremo un 1 nella posizione all'estrema sinistra, e due valori pari a 0, che indica che c'è il 100% di probabilità che quell'etichetta corrisponda a 0. Sulla base di queste informazioni viene addestrata la rete neurale, che quando va ad eseguire delle prediction su dati non etichettati, restituisce in uscita un vettore riga con 3 scalari compresi tra 0 e 1 la cui somma è pari ad 1 e che restituiscono una stima in termini di probabilità delle labels da assegnare a quei dati.

4.4 La creazione di un sensore ”intelligente”

A questo punto bisogna fare un passo indietro, e tornare a parlare della sensor board. Come è stato già affermato in precedenza, questa è dotata di diversi sensori capaci di misurare varie grandezze, come l'accelerometro IIS3DWB utilizzato per eseguire l'analisi vibrazionale del motore elettrico. È evidente, tuttavia, che i sensori da soli non sono in grado di classificare le misure che ricavano. Indipendentemente dal numero di campioni di accelerazione che l'accelerometro acquisisce, infatti, questo non sarà mai in grado di assegnargli un'etichetta. La parte finale del lavoro di tesi si concentra proprio su questo aspetto: portare il modello di rete neurale profonda sviluppato all'interno della board, così da realizzare un sensore smart.

4.4.1 Uso del software STM32CubeAI

Questo è stato reso possibile grazie al pacchetto di espansione software sviluppato e distribuito da STM, STM32CubeAI, dedicato a progetti di AI che lavorano su MCU basate sui microcontrollori della famiglia Arm Cortex-M. Esso consente di convertire modelli precedentemente addestrati in framework di lavoro specifici (come Keras o TensorFlow lite) in una libreria ottimizzata in termini di utilizzo di memoria (sia RAM che flash) che viene automaticamente integrata all'interno del progetto (Figura 4.7).

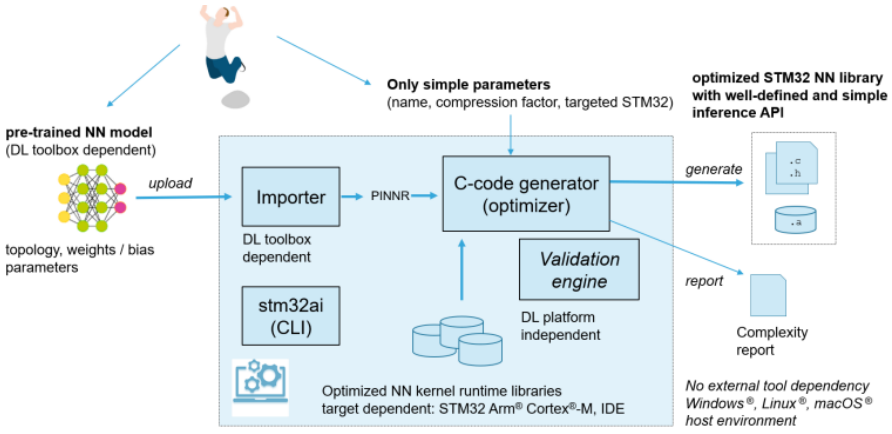


Figure 4.7: Creazione del codice ottimizzato in C

Prima di utilizzare CubeAI bisogna salvare e scaricare il progetto da Google Colab (Figura 4.8).

```
#Salvo il modello e lo converto

model.save('my_model' + '.h5')
converter = tf.lite.TFLiteConverter.from_keras_model(model)
converter.optimization = [tf.lite.Optimize.OPTIMIZE_FOR_SIZE]
tflite_model = converter.convert()
open('my_model' + '.tflite','wb').write(tflite_model)
```

Figure 4.8: Salvataggio del modello

Qui bisogna prestare attenzione, perché il formato in cui va salvato il modello non è univoco. Una possibile scelta è salvarlo nel formato classico dei modelli realizzati con Keras, ovvero l'h5. Un'altra possibilità è quella di convertirlo nel formato tflite (TensorFlow lite), più leggero in termini di byte occupati. Una volta salvato il modello, è possibile importarlo all'interno del progetto di CubeMX attraverso l'opzione add network (Figura 4.9).

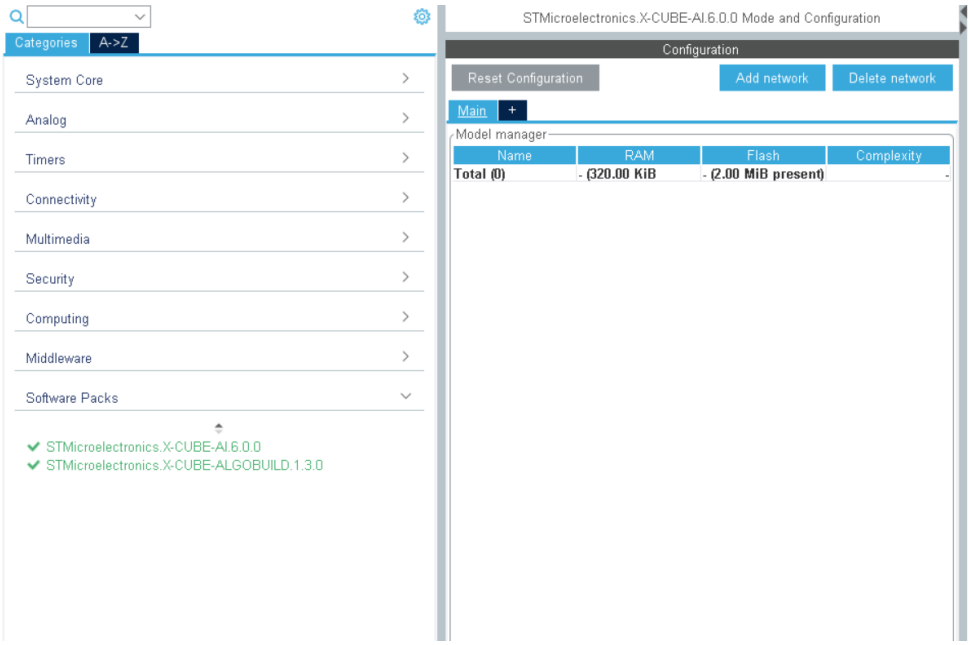


Figure 4.9: Interfaccia CubeAI

Prima di poter proseguire con la generazione automatica del codice, bisogna analizzare il modello. L'analisi del modello genera un report, che ne riassume le caratteristiche generali e quanto spazio in termini di ram e di memoria flash andrà ad occupare quando verrà caricato all'interno del microcontrollore.

Importato il modello, bisogna passare alla fase di generazione del codice. Come nei casi precedenti, MX si occupa di questa parte, mettendo a disposizione dell'utente tutte le librerie di cui necessita per eseguire l'inferenza.

Questa rappresenta essenzialmente la fase di test del modello di rete neurale, la fase in cui l'algoritmo va a classificare dati di cui non conosce l'etichetta a priori. La prima cosa da fare è dichiarare un puntatore al nostro modello, per poi crearne un'istanza attraverso l'utilizzo della funzione *create*, che riceve in ingresso proprio il puntatore che abbiamo dichiarato (Figura 4.10).

```
//Puntatore al modello
ai_handle network = AI_HANDLE_NULL;

//Creo un'istanza del NN
ai_error neural_network;
ai_err = ai_network_create(&network, AI_NETWORK_DATA_CONFIG);
if (ai_err.type != AI_ERROR_NONE)
{
    sprintf(&buffer2,"Errore nella creazione del network");
    uprintf(buffer2);
}
```

Figure 4.10: Creazione dell'istanza del NN

Se la creazione è andata a buon fine, nessun messaggio di errore viene visualizzato sul terminale. A questo punto bisogna ricavare i parametri della rete neurale, come i pesi e gli output delle funzioni di attivazione (Activations), che vengono immagazzinati all'interno di una struttura dati specializzata definita all'interno delle librerie generate. Grazie al metodo *weights_get* è possibile ricavare i pesi della rete (Figura 4.11).

```
//Ricavo i parametri del modello

  ai_network_params ai_params = {
  AI_NETWORK_DATA_WEIGHTS(ai_network_data_weights_get()),
  AI_NETWORK_DATA_ACTIVATIONS(activations)
};
```

Figure 4.11: Parametri del modello

Attraverso la funzione *init* è possibile verificare che l'inizializzazione della rete neurale sia avvenuta correttamente (Figura 4.12).

Prima di eseguire l'inferenza, bisogna costruire una struttura dati che possa contenere un numero n di campioni rispetto ai quali eseguire la stessa e che siano immagazzinati nello stesso

```
//Verifico che il NN sia inizializzato correttamente

if (!ai_network_init(network, &ai_params))
{
    sprintf(buffer2,"Errore");
    uprintf(buffer2);
}
```

Figure 4.12: Inizializzazione del NN

formato rispetto al quale il modello è stato addestrato, ovvero una matrice con un numero di righe pari al numero di campioni rispetto ai quali verrà computata l'inferenza e 6 colonne. La matrice viene riempita con le misure acquisite dal sensore e processate attraverso l'FFT. A questo punto è possibile procedere con l'inferenza grazie alla funzione *run* (Figura 4.13), che prende in ingresso la matrice contenente i campioni e quella dove verranno inseriti i valori di output, ovvero le etichette assegnate.

```
//Eseguo l'inferenza
inferenza = ai_network_run(network, &ai_input[0], &ai_output[0]);
if (inferenza != 1) {

    sprintf(buffer2,"Errore");
    uprintf(buffer2);

}
```

Figure 4.13: Esecuzione dell'inferenza

Chapter 5

Conclusioni e sviluppi futuri

Grazie all'utilizzo di CubeAI è stato possibile sfruttare la potenza di calcolo del microcontrollore Arm Cortex-M4 per far lavorare l'algoritmo di deep learning direttamente all'interno della board. Questa idea è alla base dell'edge computing, ovvero l'elaborazione e l'analisi dei dati in prossimità di dove questi vengono generati. La possibilità di gestire le elaborazioni a livello locale risolve molti problemi che si potrebbero incontrare se si utilizzassero algoritmi di AI che lavorano in cloud, come ad esempio:

- Problemi di latenza
- Problemi legati alla larghezza di banda limitata
- Problemi relativi alla privacy dei dati. Poiché questi vengono processati a livello locale, non c'è il rischio che vengano rubati durante la fase di trasferimento via internet oppure mentre sono immagazzinati nel cloud.

Questo è reso possibile grazie alla combinazione di microcontrollori sempre più performanti e di tecniche di ottimizzazione che permettono ad algoritmi di machine learning e deep learning di lavorare su architetture embedded. In questo modo è possibile sviluppare una strategia di manutenzione che elimina la necessità di controlli periodici, in quanto il riconoscimento della condizione di malfunzionamento del motore avviene all'insorgere della stessa, che viene gestita a livello locale grazie all'ausilio di microcontrollori pensati per lavorare con algoritmi di AI opportunamente ottimizzati per funzionare su hardware dalle risorse limitate in termini di memoria RAM e flash.

5.1 Sviluppi futuri

Quella esplorata all'interno del lavoro di tesi è solo una delle tante strade che è possibile seguire per implementare una strategia manutentiva che impiega la tecnologia dei microcontrollori ed algoritmi di apprendimento automatico capaci di analizzare grandi moli di dati e sfruttarle per stabilire pattern complessi.

Ci sono diversi modi per costruire un modello di rete neurale che non sono necessariamente basati sul fornire allo stesso l'etichetta dell'input che riceve. Questo tipo di apprendimento, definito unsupervisioned learning, si adatta perfettamente alla tipologia di dati non strutturati per la quale il deep learning è stato sviluppato, in quanto sarà la rete stessa ad identificare una struttura alla base dei dati che gli vengono forniti. All'interno di questa categoria ricadono, ad esempio, gli algoritmi di clustering (Figura 5.1), che prevedono che i dati vengano raggruppati in insiemi detti "cluster". Questi algoritmi calcolano una distanza tra i dati di input, in base alla quale vengono creati i suddetti insiemi.

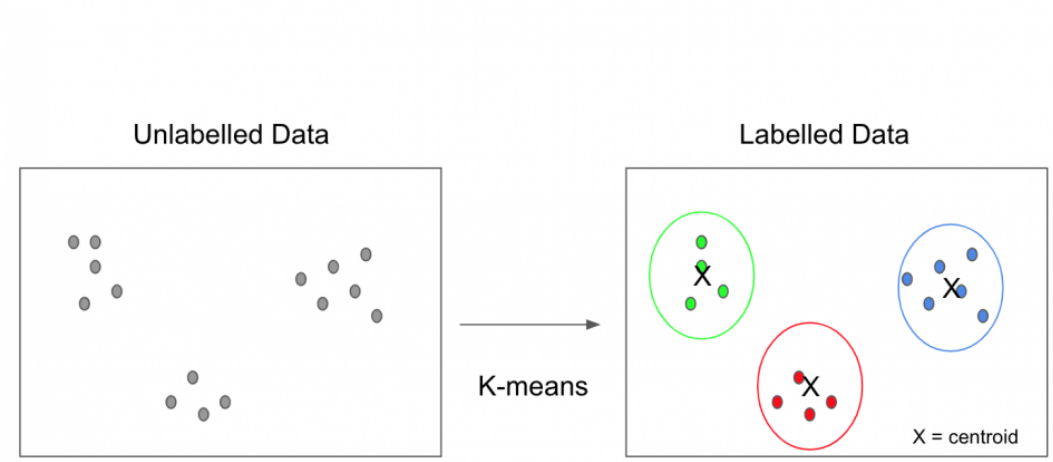


Figure 5.1: Esempio di algoritmo di clustering

Anche la definizione del tipo di modello non è univoca. Quello sequenziale è uno dei due modelli che è possibile costruire grazie alla libreria Keras, che mette a disposizione dell'utente anche l'API functional. Questo consente di sviluppare modelli di apprendimento complessi multi-input e multi-output. In questo modo si potrebbe eseguire una diagnosi più specifica per

rilevare un'eventuale condizione di guasto o di malfunzionamento del motore elettrico che non preveda esclusivamente l'analisi vibrazionale ma anche misure ricavate da altri sensori, come nel caso dell'analisi delle correnti di statore vista in precedenza con la MCSA.

Spingendosi oltre la semplice classificazione, sarebbe inoltre possibile addestrare un modello di rete neurale che non sia solo in grado di classificare dei dati in base a delle etichette, ma che sia capace di predire valori continui. Le vie da esplorare per sviluppare strategie di manutenzione predittiva sono numerose, ma hanno tutte un minimo comune denominatore: sviluppare sistemi completamente automatizzati sia dal punto di vista della raccolta e dell'elaborazione dei dati, sia dal punto di vista dell'interpretazione di quegli stessi dati.

Bibliography

[1] Mourad Benmessaoud and Mekkakia Maaza Nasreddine. Optimization of mems capacitive accelerometer. *Microsystem Technologies*, 19:713–720, 2013.

[2] Luis Miguel Contreras-Medina, Rene de Jesus Romero-Troncoso, Eduardo Cabal-Yepez, Jose de Jesus Rangel-Magdaleno, and Jesus Roberto Millan-Almaraz. Fpga-based multiple-channel vibration analyzer for industrial applications in induction motor failure detection. *IEEE Transactions on Instrumentation and Measurement*, 59(1):63–72, 2009.

[3] Christian Janiesch, Patrick Zschech, and Kai Heinrich. Machine learning and deep learning. *Electronic Markets*, 31(3):685–695, 2021.

[4] Nikhil Ketkar and Nikhil Ketkar. Introduction to keras. *Deep learning with python: a hands-on introduction*, pages 97–111, 2017.

[5] Navin Kumar Manaswi and Navin Kumar Manaswi. Understanding and working with keras. *Deep learning with applications using Python: Chatbots and face, object, and speech recognition with TensorFlow and Keras*, pages 31–43, 2018.

[6] Muhammad Sarfraz Moiz, Shazaib Shamim, Muhammad Abdullah, Hamdan Khan, Imtiaz Hussain, Anas Bin Iftikhar, and TD Memon. Health monitoring of three-phase induction motor using current and vibration signature analysis. In *2019 International Conference on Robotics and Automation in Industry (ICRAI)*, pages 1–4. IEEE, 2019.

[7] Levent Sevgi. Numerical fourier transforms: Dft and fft. *IEEE Antennas and Propagation Magazine*, 49(3):238–243, 2007.

[8] STMicroelectronics. Multi-sensor predictive maintenance kit with io-link stack v.1.1. 2020.

[9] STMicroelectronics. Artificial intelligence (ai) software expansion for stm32cube. pages 1–3, 2023.

[10] Minh-Quang Tran, Meng-Kun Liu, Quoc-Viet Tran, and Toan-Khoa Nguyen. Effective fault diagnosis based on wavelet and convolutional attention neural network for induction motors. *IEEE Transactions on Instrumentation and Measurement*, 71:1–13, 2021.

[11] Martin Vetterli, Henri J Nussbaumer, et al. Simple fft and dct algorithms with reduced number of operations. *Signal processing*, 6(4):267–278, 1984.

[12] Wikipedia. Rettificatore (reti neurali). 2021.