UNIVERSITÀ
DI SIENA
1240

Dipartimento ingegneria dell'informazione e scienze matematiche

Corso di laurea in ingegneria informatica e dell'informazione

Tecniche di Machine Learning su Piattaforme Embedded in Ambiente Arduino

Relatori:
Prof. Alessandro Pozzebon

Tesi di laurea di:
Samuele Maggiori

Anno Accademico 2020-2021

Indice

Introduzione

In questo lavoro di tesi verranno analizzati tutti i vari passaggi necessari per la realizzazione di un guanto capace di rivelare quando l'indossatore schiocca le dita. Lo scopo è quello di fornire le basi per la realizzazione di un generico algoritmo di Tiny Machine Learning (TinyML), che potrà poi essere portato su un qualsiasi microcontrollore.

Ho deciso di dividere la tesi in due capitoli principali: uno per la descrizione della strumentazione utilizzata, e l'altro dedicato alla spiegazione di come ho realizzato il progetto.

Nel primo capitolo saranno elencati tutti i dispositivi utilizzati, soffermandosi anche sulla teoria dietro al funzionamento dei sensori con cui vengono raccolti i dati.

Nel secondo capitolo verrà spiegato invece come acquisire i dati dai sensori, creare ed addestrare una rete neurale, ed infine caricarla su una scheda Arduino. Verranno inoltre riportate le fondamentali funzioni Python ed Arduino utilizzate nei vari codici, e la loro implementazione nel progetto.

Capitolo 1

Strumentazione

In questo capitolo ho elencato i componenti principali utilizzati per il progetto, iniziando dalla scheda Arduino "Nano 33 BLE Sense" sulla quale caricherò come vedremo nel prossimo Capitolo un algoritmo di machine learning, proseguendo con la descrizione dei sensori di Hall, dei quali spiegherò la teoria alla base del loro funzionamento, ovvero "l'effetto Hall" da cui prendono il nome.
Infine darò una breve descrizione dell'applicativo web "Edge Impulse" [1], sul quale è possibile creare una rete neurale, allenarla e ricevere una libreria ZIP Arduino, contenente il modello della rete.

1.1 Arduino Nano 33 BLE Sense

L'Arduino Nano 33 BLE Sense è stata una delle prime schede sul mercato a supportare l'utilizzo dei modelli di machine learning costruiti con la libreria "TensorFlow Lite"[2]; sulla scheda di sviluppo sono inclusi anche vari sensori, che si possono trovare elencati nella relativa pagina ufficiale, [3] sul sito Arduino, con i relativi datasheet.
Possiamo utilizzarli direttamente nel codice Arduino, tramite l'Arduino IDE[4] ad esempio, installando le varie librerie, specifiche per ogni sensore.

1.1.1 Esempio [Libreria LSM9DS1: Accelerometro]

La libreria LSM9DS1 corrisponde all'accelerometro montato sull'Arduino, dopo averla installata tramite il gestore librerie dall'IDE, tramite il codice che vediamo qua sotto, è possibile scrivere sul monitor seriale i valori ottenuti dall'accelerometro.

```
1 #include <Arduino_LSM9DS1.h>
2 void setup(){
3     Serial.begin(115200);
```

```
    // Start accelerometer
    if (!IMU.begin()) {
        Serial.println("Failed to initialize IMU!");
        while (1);
    }
}//end setup

void loop(){
    float x, y, z
    IMU.readAcceleration(x, y, z);
}//end loop
```

1.1.2 Lista dei sensori sulla board Arduino

Come accennato in precedenza la scheda Arduino presenta dei sensori già saldati:

- IMU[1][LSM9DS1]
- Microfono [MP34DT05]
- Luce, Prossimità [APDS9960]
- Pressione barometrica [LPS22HB]
- Temperatura ed Umidità [HTS221]

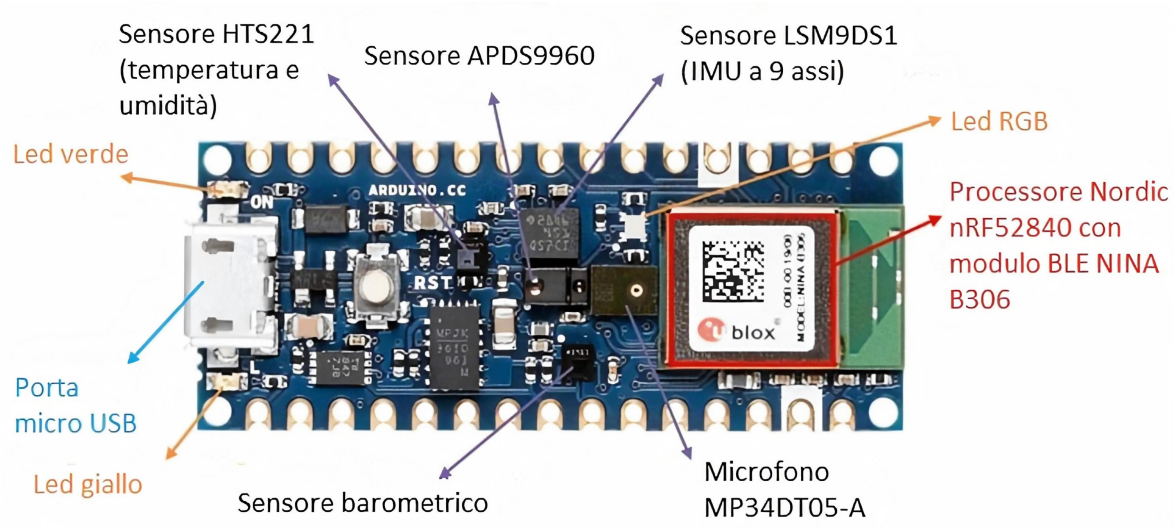


Figura 1.1: Scheda Arduino Nano 33 BLE - Nome e Posizione dei Componenti

[1]IMU: Inertial Measurement Unit

ATTENZIONE:

L'Arduino lavora a 3.3 V, di conseguenza accetta solamente voltaggi pari od inferiori a **3.3 V**: se collegata a tensioni di valori superiori, la scheda potrebbe danneggiarsi

1.1.3 Specifiche Fondamentali della Scheda

Di seguito ho riportato le informazioni ottenute dalla pagina ufficiale sul sito Arduino, relative alla scheda di sviluppo: "Nano 33 BLE Sense"[4].

MICROCONTROLLER	nRF52840
OPERATING VOLTAGE	3.3V
INPUT VOLTAGE	4.5V - 21V
DC CURRENT PER I/O PIN	15 mA
CLOCK SPEED	64MHz
CPU[2] FLASH MEMORY	1MB (nRF52840)
SRAM[3]	256KB (nRF52840)
EEPROM[4]	none
DIGITAL INPUT / OUTPUT PINS	14
PWM[5] PINS	all digital pins
UART[6]	1
SPI[7]	1
I2C[8]	1
ANALOG INPUT PINS	8 (ADC 12 bit 200 ksamples)
ANALOG OUTPUT PINS	Only through PWM (no DAC)
EXTERNAL INTERRUPTS	all digital pins
LED_BUILTIN	13
USB Native in the	nRF52840 Processor

[2]CPU: Central Processing Unit

[3]SRAM: Static Random-Access Memory

[4]EEPROM: Electronically Erasable Programmable Read-Only Memory

[5]PMW: Pulse-Width modulation

[6]UART: Universal asynchronous receiver-transmitter

[7]SPI: Serial Peripheral Interface

[8]I2C: Inter-Integrated Circuit

1.2 Effetto Hall

L'effetto Hall è un fenomeno legato al passaggio di una corrente I attraverso un conduttore, in una zona in cui è presente un campo magnetico, tramite questo fenomeno è possibile determinare due importanti caratteristiche del conduttore in questione: il segno dei portatori di carica e la loro densità [5].
Prende il nome dal fisico statunitense Edwin Hall, colui che ha scoperto questo effetto nel 1879 [6].
Indichiamo con $\vec{I}$ e $\vec{J}$ i due vettori che descrivono la corrente e la densità di corrente, rispettivamente, il cui verso è definito come: il verso in cui scorrerebbe la corrente, se i portatori di carica fossero positivi [5].
Rispetto alla velocità $\vec{v}$ dei portatori di carica, se questi ultimi sono portatori di carica positivi, allora $\vec{v}$ e $\vec{J}$ saranno paralleli; se invece i portatori di carica sono negativi i due vettori saranno antiparalleli [5].

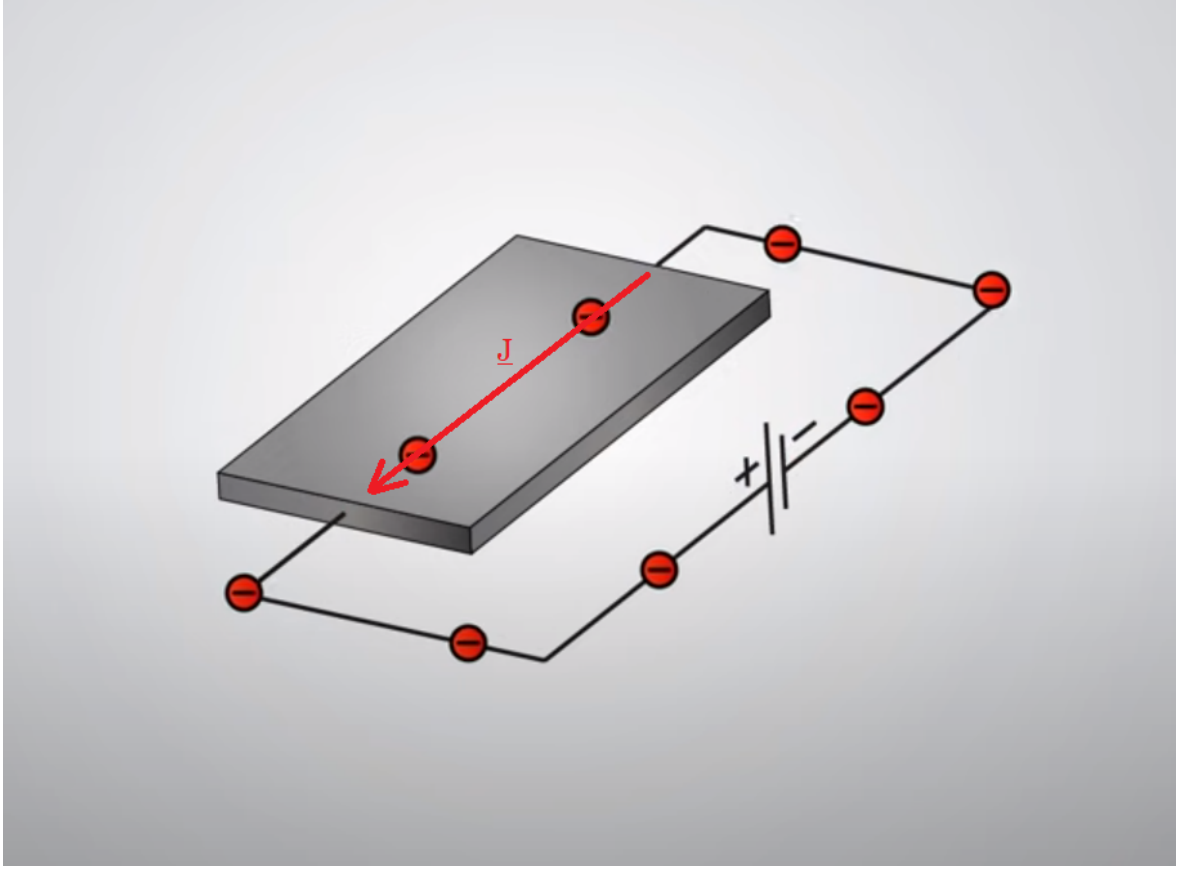


Figura 1.2: Scorrimento di una corrente elettrica in una lastra di metallo

Supponiamo di avere un conduttore largo e sottile di superficie S, spessore d, sottoposto ad un passaggio di corrente $\vec{I}$, a cui corrisponde densità di corrente $\vec{J}$, come illustrato in Figura 1.2.

Adesso si ipotizzi che la lastra di metallo sia soggetta ad un campo magnetico $\vec{B}$, come in Figura 1.3: tale campo modificherà il normale percorso della corrente, imponendogli una forza, in funzione del campo magnetico, detta "Forza di Lorentz".

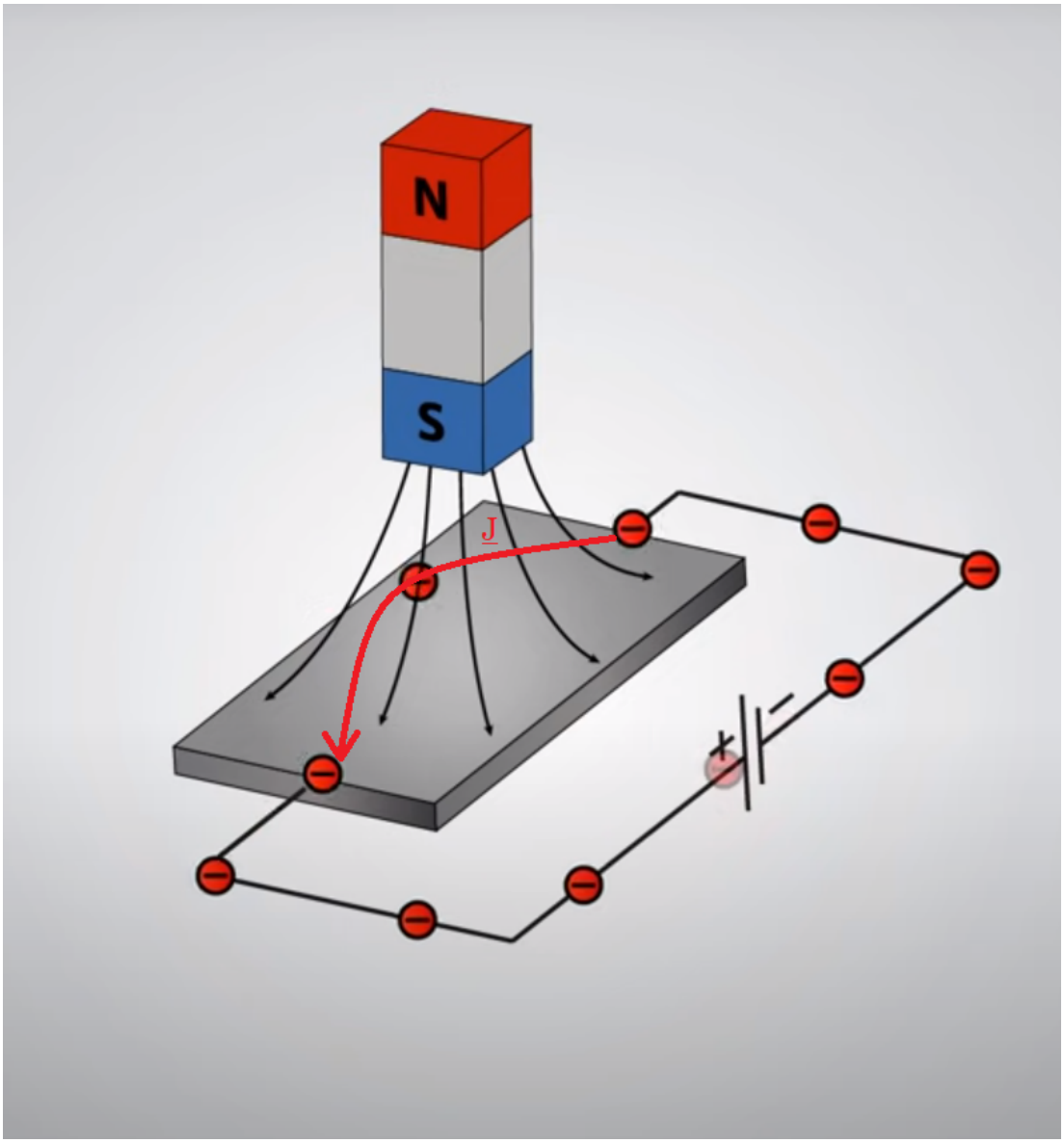


Figura 1.3: Effetto della forza di Lorentz

Quindi, su ciascun portatore la forza impressa è:

$$\vec{F} = q\vec{v} \times \vec{B} \tag{1.1}$$

In definitiva, a prescindere dal segno dei portatori di carica, ossia a prescindere dal verso effettivo della corrente, l'azione del campo magnetico è sempre quella di spingere trasversalmente (rispetto alla direzione della corrente) i portatori di carica, addensandoli su un bordo del conduttore, come rappresentato in Figura 1.4, portando così ad avere una differenza di potenziale $\Delta\varphi$ tra i bordi opposti del conduttore, che può essere misurata.
Basta allora stabilire il segno di questa differenza di potenziale per stabilire il segno dei portatori di carica.
La presenza di questa differenza di potenziale trasversale nei conduttori percorsi da corrente e soggetti ad un campo magnetico ad essa ortogonale, prende appunto il nome di "effetto Hall".[5].

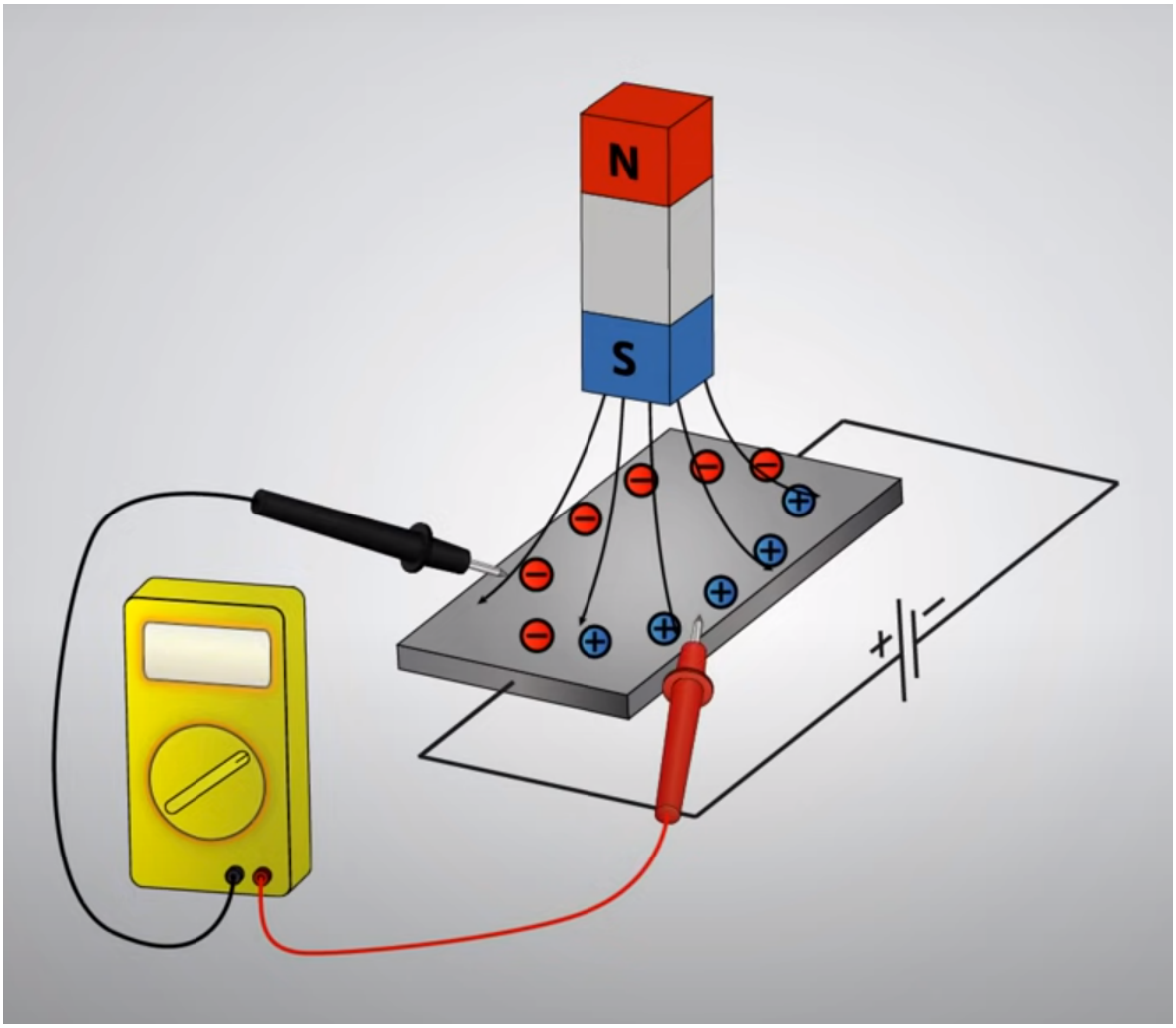


Figura 1.4: Misurazione dell'effetto Hall

Il fatto che la forza di Lorentz faccia accumulare i portatori di carica su un bordo del conduttore, con conseguente creazione di una differenza di potenziale tra i due bordi opposti del conduttore, fa sì che si crei un campo elettrico $\vec{E_h}$ trasversale, diretto cioè ortogonalmente alla direzione della corrente.
Di conseguenza, quanta più corrente scorre, tanti più portatori vengono spinti dalla forza di Lorentz, e tanto maggiore sarà il valore di $\vec{E_h}$; quindi se scorre più corrente, maggiore sarà la forza $\vec{F_h}$ che il campo elettrico trasversale esercita sugli stessi portatori [5].

1.2.1 Differenza di potenziale data dall'effetto Hall

Ricordiamo che comunque la forza elettrica $\vec{F_h}$ si oppone sempre alla forza magnetica, quindi, a prescindere dal segno dei portatori di carica, arriverà un momento in cui le due forze si equilibrano perfettamente, cioè quando la forza di Lorentz risulta nulla:[5].

$$\vec{F} = q\vec{v} \times \vec{B} + q\vec{E_h} = 0 \tag{1.2}$$

Ricavando il valore del campo elettrico $\vec{E_h}$, all'equilibrio:

$$\vec{E_h} = -\vec{v} \times \vec{B} \tag{1.3}$$

La velocità da considerare deve essere quella media dei portatori di carica, per cui la relazione diventa:

$$\vec{E_h} = -\langle\vec{v}\rangle \times \vec{B} \tag{1.4}$$

Indichiamo con N il numero totale di portatori di carica per unità di volume, possiamo quindi ricavare la densità di corrente:

$$\vec{J} = Nq\langle\vec{v}\rangle \tag{1.5}$$

Da cui ricaviamo che:

$$\langle\vec{v}\rangle = \frac{\vec{J}}{Nq} \tag{1.6}$$

Sostituendo nell'espressione del campo elettrico (1.4) , otteniamo:

$$\vec{E_h} = -\frac{1}{Nq} \cdot (\vec{J} \times \vec{B}) \tag{1.7}$$

Per definizione, la densità di corrente che scorre lungo il conduttore è data, in modulo, dal rapporto tra la corrente I, e l'area S del conduttore attraversata dalla corrente stessa: considerando che abbiamo definito il conduttore con uno spessore d e e larghezza l,

si deduce che:

$$S = d \cdot \tag{1.8}$$

per cui:

$$J = \frac{I}{d \cdot} \tag{1.9}$$

Sostituendo nella espressione del campo elettrico (1.7) e calcolando il modulo di tale campo, abbiamo:

$$\vec{E_h} = -\frac{J \cdot B}{Nq} sin\theta = -\frac{I \cdot B}{Nq \cdot d \cdot l} sin\theta \tag{1.10}$$

Dove con θ viene indicato l'angolo formato dai vettori $\vec{J}$ e $\vec{B}$.
Se consideriamo $\theta = \pi/2$, ipotizzando che il campo $\vec{B}$ sia ortogonale alla direzione della corrente, avremo che il modulo del campo elettrico trasversale è:

$$\vec{E_h} = -\frac{I \cdot B}{Nq \cdot d \cdot l} sin\theta \tag{1.11}$$

Dal valore del campo all'equilibrio, si può risalire al valore della differenza di potenziale $\vec{V_h}$ all'equilibrio:

$$\vec{V_h} = -\vec{E_h} \cdot l = -\frac{I \cdot B}{Nq \cdot d} sin\theta \tag{1.12}$$

Si deduce che, a parità di forma del conduttore e a parità di campo magnetico, la differenza di potenziale che si manifesta ai bordi del conduttore (una volta raggiunto l'equilibrio) cresce proporzionalmente al crescere di $\vec{I}$, ed al diminuire del fattore $N \cdot q$, che è appunto la densità spaziale di carica e che si può ricavare dalla relazione (1.11) [5].

1.3 Sensori ad Effetto Hall

La famiglia A130 (A1301 e A1302) è composta da sensori raziometrici lineari che sfruttano l'effetto Hall per fornire in uscita una tensione proporzionale al campo magnetico applicato. Tutti i dispositivi appartenenti alla famiglia A130, in assenza di un campo magnetico significativo, presentano in uscita una tensione così detta "a riposo", pari al 50% della tensione di alimentazione. Lo specifico sensore che andremo ad utilizzare (A1301) ha una sensibilità tipica pari ad 1.3 $\frac{\text{mV}}{\text{G}}$ [7].

Il circuito integrato ad effetto Hall, presente in ogni sensore, include un sensore di Hall, un amplificatore lineare ed una struttura CMOS di Classe A come stadio di uscita, lo schema a blocchi del sensore è riportato in Figura 1.5.
Integrare il sensore di Hall e l'amplificatore su un unico chip minimizza molti dei problemi normalmente associati a segnali analogici a basso livello di tensione [7].
Un'alta precisione nei livelli di uscita, viene inoltre ottenuta grazie ad una regolazione dell'offset, effettuata come ultimo passo del processo di produzione [7].

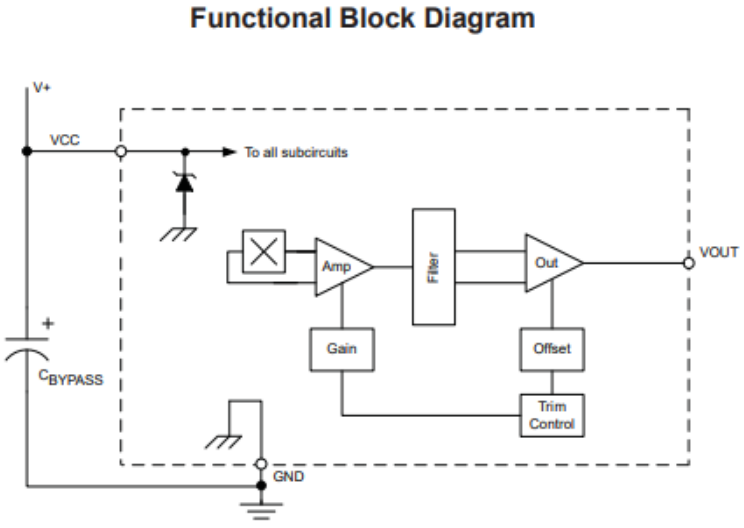


Figura 1.5: Sensore ad effetto Hall A1302 - Functional Block Diagram

DEVICE CHARACTERISTICS over operating temperature range, T_A, and V_{CC} = 5 V, unless otherwise noted

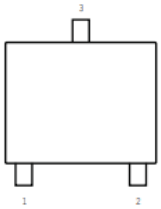
Characteristic	Symbol	Test Conditions	Min.	Typ.	Max.	Units
Electrical Characteristics						
Supply Voltage	V_{CC}	Running, T_J < 165°C	4.5	–	6	V
Supply Current	I_{CC}	Output open	–	–	11	mA
Output Voltage	$V_{OUT(High)}$	I_{SOURCE} = –1 mA, Sens = nominal	4.65	4.7	–	V
	$V_{OUT(Low)}$	I_{SINK} = 1 mA, Sens = nominal	–	0.2	0.25	V
Output Bandwidth	BW		–	20	–	kHz
Power-On Time	t_{PO}	$V_{CC(min)}$ to 0.95 V_{OUT}; B = ±1400 G; Slew rate = 4.5 V/µs to 4.5 V/100 ns	–	3	5	µs
Output Resistance	R_{OUT}	I_{SINK} ≤ 1 mA, I_{SOURCE} ≥ –1 mA	–	2	5	Ω
Wide Band Output Noise, rms	V_{OUTN}	External output low pass filter ≤ 10 kHz; Sens = nominal	–	150	–	µV
Ratiometry						
Quiescent Output Voltage Error with respect to ΔV_{CC}[1]	$\Delta V_{OUTQ(V)}$	T_A = 25°C	–	–	±3.0	%
Magnetic Sensitivity Error with respect to ΔV_{CC}[2]	$\Delta Sens_{(V)}$	T_A = 25°C	–	–	±3.0	%
Output						
Linearity	Lin	T_A = 25°C	–	–	±2.5	%
Symmetry	Sym	T_A = 25°C	–	–	±3.0	%
Magnetic Characteristics						
Quiescent Output Voltage	V_{OUTQ}	B = 0 G; T_A = 25°C	2.4	2.5	2.6	V
Quiescent Output Voltage over Operating Temperature Range	$V_{OUTQ(\Delta T_A)}$	B = 0 G	2.2	–	2.8	V
Magnetic Sensitivity	Sens	A1301; T_A = 25°C	2.0	2.5	3.0	mV/G
		A1302; T_A = 25°C	1.0	1.3	1.6	mV/G
Magnetic Sensitivity over Operating Temperature Range	$Sens_{(\Delta T_A)}$	A1301	1.8	–	3.2	mV/G
		A1302	0.85	–	1.75	mV/G

[1]Refer to equation (4) in Ratiometric section on page 4.
[2]Refer to equation (5) in Ratiometric section on page 4.

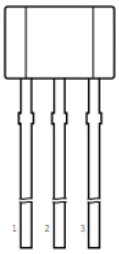
Figura 1.6: Sensore ad effetto Hall A1302 - Datasheet

Pin-out Drawings

Package LH



Package UA



Terminal List

Symbol	Number		Description
	Package LH	Package UA	
VCC	1	1	Connects power supply to chip
VOUT	2	3	Output from circuit
GND	3	2	Ground

Figura 1.7: Sensore ad effetto Hall A1302 - Pin Out Drawings

Come possiamo vedere in Figura 1.7 sono disponibili due tipi di package per lo stesso dispositivo: LH, a 3-pin SOT23W specifico per il montaggio superficiale, ed il pacchetto UA, a 3-pin ultramini SIP per il montaggio a foro passante.
Entrambi i pacchatti sono senza piombo (Pb) ed i leadframes sono completamente placcati in stagno opaco.

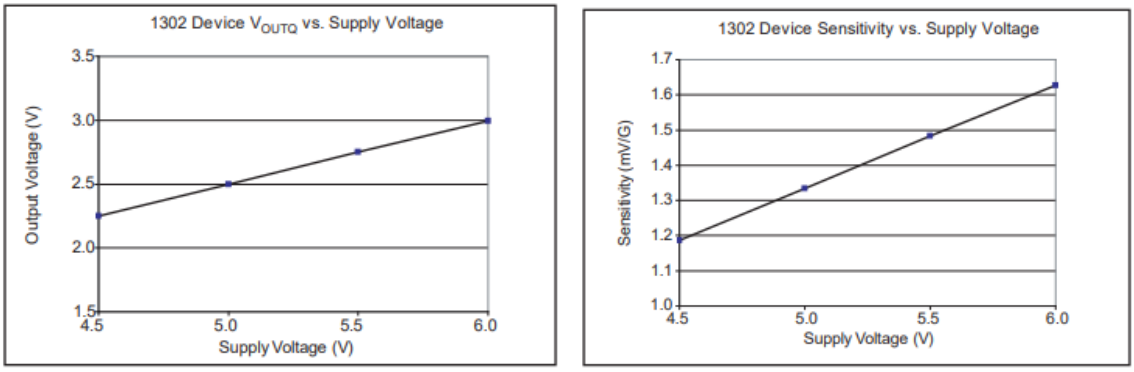


Figura 1.8: Sensore ad effetto Hall A1302 - Vout at Quiescent State and Sensitity vs. Supply Voltage

Tensione in uscita a riposo:
Quando il sensore si trova in uno stato di riposo (nessun campo magnetico significativo: $\vec{B}$=0 T), in uscita V_{outq} assume un valore pari a metà della tensione di alimentazione V_{cc}. Data la temperatura ambiente T_A ed il range della tensione di alimentazione V_{cc}, è possibile calcolare una tolleranza sulla tensione a riposo, ΔV_{outq}, in funzione di ΔV_{cc} e ΔT_A, viene definita come:

$$\Delta V_{outq}(\Delta T_A) = \frac{\Delta V_{outq}(T_A) - \Delta V_{outq}(25°)}{Sens(25°)} \tag{1.13}$$

dove "$Sens$" si misura in $\frac{\text{mV}}{\text{G}}$ ed indica la sensitività del dispositivo.

Razionometrico:
Il sensore A1301 è caratterizzato da un'uscita razionometrica, ciò vuol dire sia la V_{outq}, che la sensibilità magnetica, $Sens$, sono proporzionali alla tensione di alimentazione V_{cc}. La percentuale di cambiamento (%) della tensione in uscita a riposo è definita come:

$$\Delta V_{outq}(\Delta V) = \frac{V_{outq}(V_{cc})/V_{outq}(5V)}{V_{cc}/5V} \times 100\% \tag{1.14}$$

Mentre il *ratiometric change* (%) per la Sensitività è definito come:

$$\Delta Sens(\Delta V) = \frac{Sens(V_{cc})/Sens(5V)}{V_{cc}/5V} \times 100\% \tag{1.15}$$

Sensitività:
La presenza di un campo magnetico a polarità positiva ($+B$), perpendicolare alla parte frontale del dispositivo, aumenta la tensione di uscita V_{out}, in proporzione al campo magnetico applicato, da V_{outq} fino ad un massimo pari a V_{cc}.
Al contrario se si ha di fronte la polarità negativa di un campo magnetico ($-B$), nella stessa orientazione, ridurrà proporzionalmente V_{out} da V_{outq} fino ad un minimo pari a 0 V.
La proporzionalità di queste variazioni è definita come la sensibilità magnetica del dispositivo:

$$Sens = \frac{V_{out}(-B) + V_{out}(+B)}{2B} \tag{1.16}$$

La stabilità della sensibilità magnetica del dispositivo è una funzione della temperatura ambiente, $\Delta Sens(\Delta T_A)$ in percentuale (%) è definita come:

$$\Delta Sens(\Delta T_A) = \frac{Sens(T_A) - Sens(25°)}{Sens(25°)} \times 100\% \tag{1.17}$$

Simmetria e Linearità:
Lo stadio di uscita montato sul chip è costruito per fornire un'uscita lineare, se fornita una tensione di alimentazione pari a 5 V.
Anche se l'applicazione di forti campi magnetici non danneggiano il dispositivo, forzano però l'uscita in una zona di non-linearità.
La linearità in percentuale è misurata come segue:

$$Lin+ = \frac{V_{out}(+B) - V_{outq}}{2(V_{out}(+B\frac{1}{2}) - V_{outq})} \times 100\% \tag{1.18}$$

$$Lin- = \frac{V_{out}(-B) - V_{outq}}{2(V_{out}(-B\frac{1}{2}) - V_{outq})} \times 100\% \tag{1.19}$$

e la simmetria di uscita come:

$$Sym = \frac{V_{out}(+B) - V_{outq}}{V_{outq}) - V_{out}(-B)} \times 100\% \tag{1.20}$$

Riportate in questo paragrafo i grafici relativi alla tensione a riposo, ed alla sensibilità del dispositivo in rispetto alla tensione di alimentazione, in Figura 1.8, che saranno poi utili nell'utilizzo pratico del sensore[7].

1.4 Edge Impulse

Edge Impulse[1], ovvero TinyML come servizio, consente l'utilizzo dell'apprendimento automatico a tutti gli sviluppatori con un SDK (Software Development Kit) per dispositivi open source, sfruttando principalmente due librerie: "TensorFlow" per l'addestramento della rete neurale, e "TensorFlow Lite" per l'utilizzo su sistemi embedded.

Dalla pagina web è possibile: raccogliere dati da vari sensori, elaborare segnali in tempo reale, effettuare test su qualsiasi dispositivo [8].
In particolare viene offerta la possibilità di ottenere il modello di machine learning "TensorFlow Lite", dopo averlo addestrato, sia in un file di formato ".tflite", come libreria Arduino (che vedremo in particolare nel prossimo Capitolo), oppure come libreria C++, Cube.MX CMSIS-PACK, WebAssembly o TensorRT.

Viene inoltre offerta la possibilità di inviare raccolte di dati tramite file con estensione:

- ".CBOR"
- ".JSON"
- ".CSV"
- ".WAV" (per file audio)
- ".JPG" (per immagini)
- ".PNG" (per immagini)

Edge Impulse è una piattaforma di sviluppo, gratuita per i singoli sviluppatori, che gli utenti possono contribuire ed estendere, aggiungendo e modificando algoritmi di machine learning, o implementando supporti per nuovi dispositivi. Il software del dispositivo, inclusi SDK, client, e codice generato, viene fornito come open source con una licenza Apache 2.0 [8].

Capitolo 2

Implementazione

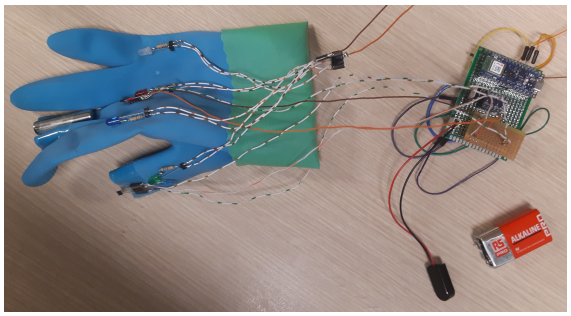


Figura 2.1: Immagine Finale del Guanto per il Riconoscimento di uno Schiocco di Dita

Utilizzando due sensori di Hall, posizionati su un guanto di gomma, uno sul retro del pollice e l'altro sulla nocca del dito medio, una calamita attaccata al retro del medio, e l'Arduino Nano 33 BLE Sense sul quale è caricato lo sketch con il modello di machine learning, sono stato in grado di rivelare quando vengono fatte schioccare le dita, e quando accade 4 led incollati sul guanto si accendono.

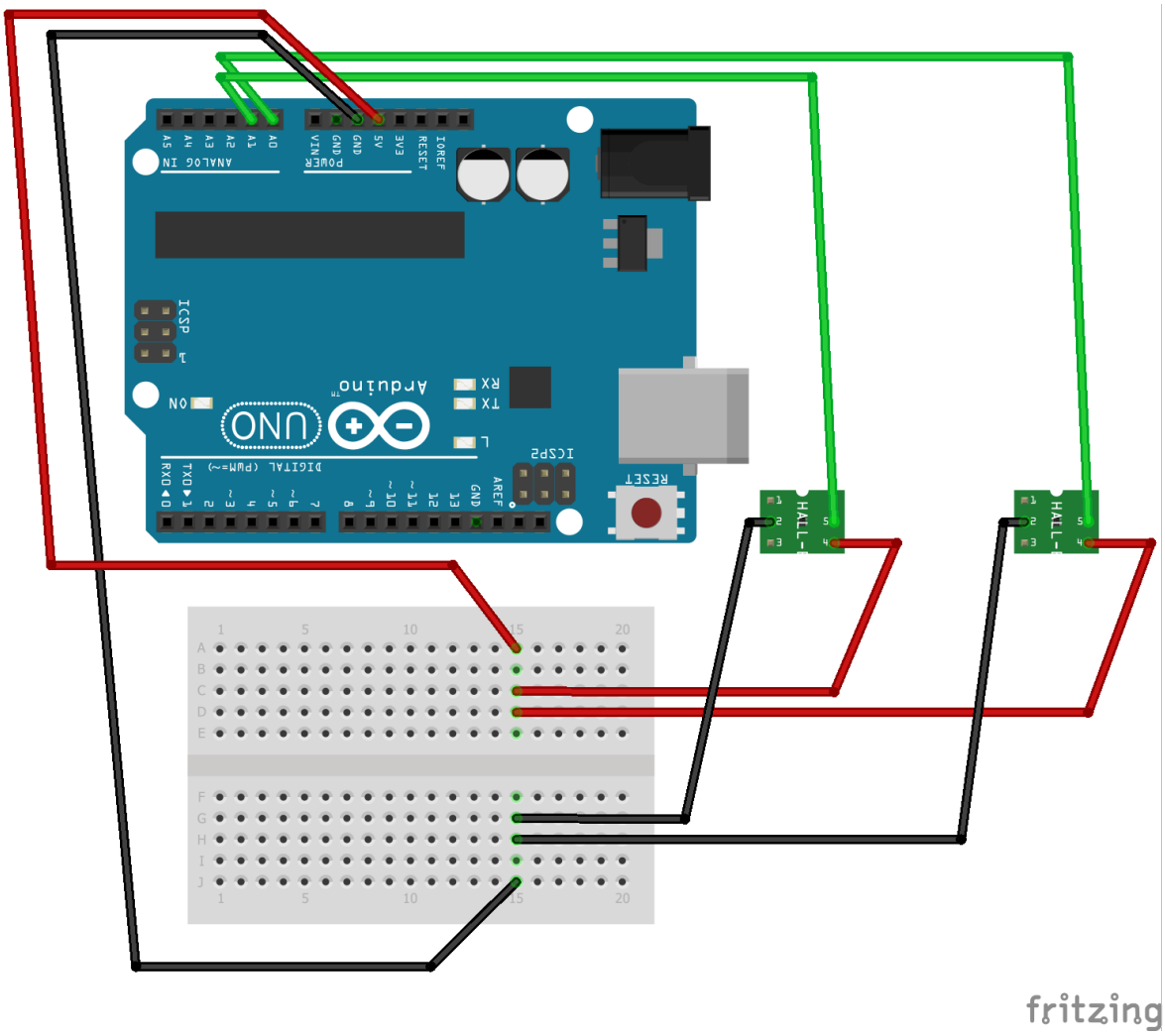
Il flusso di lavoro che ho adottato è il seguente:

- Raccolta dei dati dai sensori tramite sketch Arduino, che dovranno poi essere scritti sulla porta seriale, sempre dallo stesso sketch.
- Collezione dei dati scritti sul seriale, e salvataggio degli stessi su fogli con estensione ".csv", tramite script python.
- Inoltro dei file ".csv" appena creati sulla piattaforma Edge Impulse, con seguente creazione ed allenamento della rete neurale, che verrà poi inserita in una libreria Arduino per il dislocamento sulla scheda Nano.
- Infine, creazione di un nuovo sketch Arduino per l'utilizzo della libreria fornita da Edge.

2.1 Raccolta Dati tramite Sketch Arduino

Per prima cosa ho creato un semplice prototipo su breadboard in cui ho collegato una scheda Arduino UNO come mostrato in Figura 2.2. Ho usato questa scheda in una prima prova, per la possibilità dell'Arduino UNO di inviare e ricevere segnali a 5 V, visto che i sensori devono essere alimentati a 5 V, ed i valori in uscita dal sensore hanno range tra
0 V - 5 V, valori che normalmente **non** possono essere accettati, come scritto in precedenza, dall'Arduino Nano. Questo problema è facilmente risolvibile con convertitori di tensione generici per
5 V→ 3.3 V, convertitori che sono stati infatti adottati in seguito.

Figura 2.2: Esempio di hardware per la lettura dei sensori Hall



Una volta visto come raccogliere dati, e scelto un magnete tra quelli disponibili in laboratorio, che dia una variazione consistente se mosso vicino ai sensori, ho portato il progetto sull'Arduino Nano, tramite vari shifter di tensione, come accenato prima, ed alimentando il tutto a batteria (9 V). Lo schema del progetto è mostrato nella Figura 2.3:

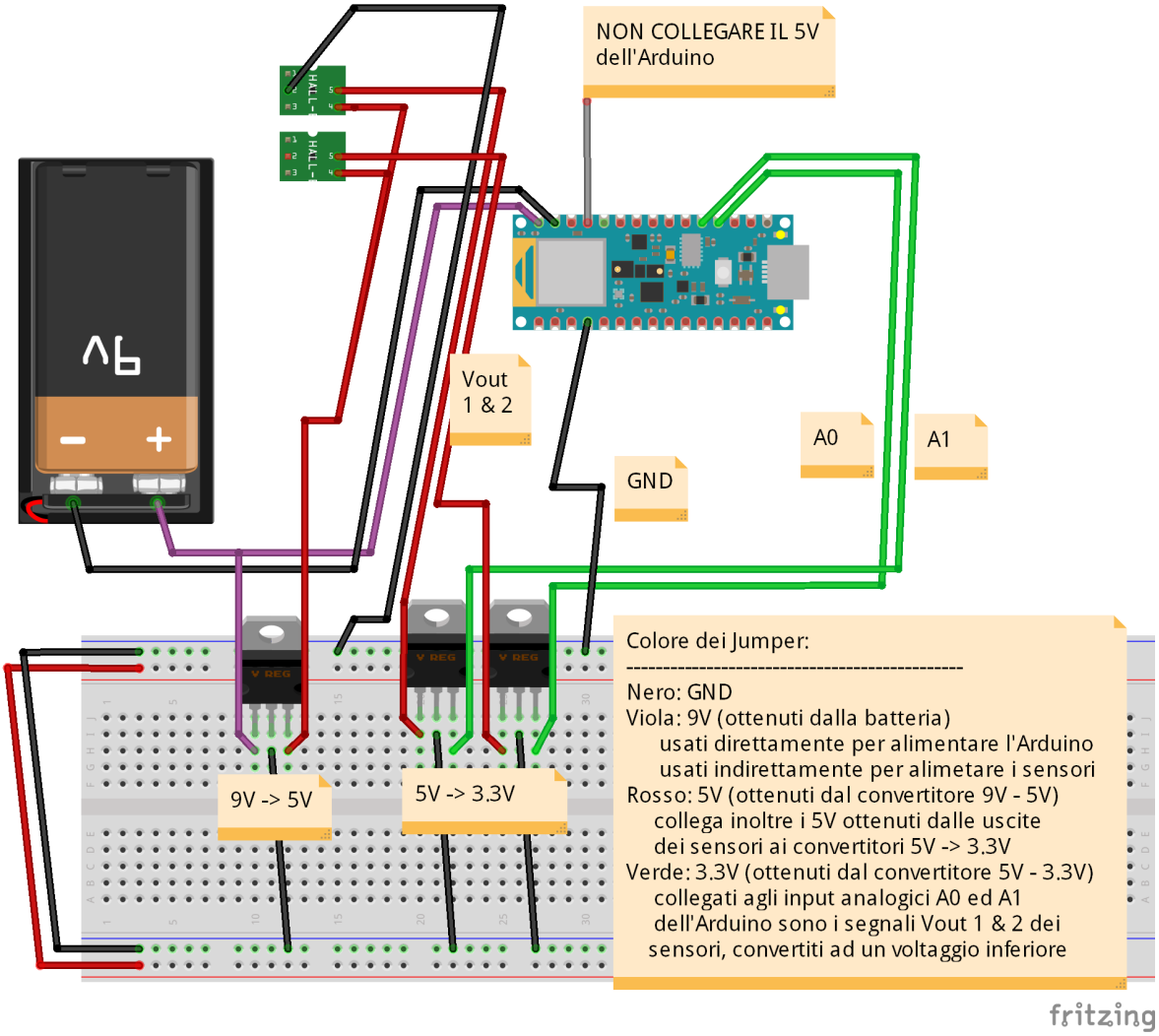


Figura 2.3: Esempio di hardware per la lettura dei sensori Hall su scheda Arduino Nano 33 BLE Sense

Tramite questa configurazione è possibile collegare l'Arduino Nano al PC per la raccolta dati tramite scrittura su seriale, utilizzando nel codice principalmente due funzioni: "AnalogRead(...)" e "Serial.print(...)", di seguito la parte principale del codice utilizzato per la lettura dai sensori:

```
void loop(){ // Lettura di due sensori
    String value_on_pollice = String(AnalogRead(A0));
    String value_on_medio = String(AnalogRead(A1));
    String timestamp = String(millis());
    Serial.println(timestamp + "," + value_on_pollice + "," +
        value_on_medio)
}
```

2.1.1 Protocollo Edge Impulse

Il formato per la scrittura dei dati sul seriale:

```
timestamp + "," + value_on_pollice + "," + value_on_medio
```

Non è stato scelto a caso, di fatti il servizio Edge Impulse richiede che i fogli ".csv" abbiano il seguente protocollo o formato, per la successiva lettura dei dati:

```
timestamp, name_value_1, name_value_2, name_value_3, name_value_4
0, 23.34, 2131.23, 321.23, 232.4
12, 23.34, 2131.23, 321.23, 232.4
23, 22.36, 2003.10, 321.23, 567.8
45, 26.24, 2056.40, 321.23, 567.9
54, 25.38, 1233.23, 321.23, 567.4
...
```

Questo formato sarà poi creato per intero dallo script python di cui parlerò nella prossima sezione, in ogni caso, è necessario che i fogli ".csv" abbiano tutti come prima colonna il "timestamp", così che Edge Impulse possa dare ad ogni dato fornitogli un marchio temporale, ecco il perché dell'omonima variabile durante la scrittura sul seriale (scritta appunto per prima).
Successivamente il sito si aspetta tutti i dati raccolti dai vari sensori, che vengono appunto scritti sul seriale, dopo il timestamp.

2.1.2 Primo Prototipo del Guanto per la Raccolta Dati

A questo punto è necessario creare un prototipo per la raccolta dati, in quanto ho necessità di registrare i vari dati durante il movimento prestabilito, uno schiocco di dita. Il prototipo con i sensori letteralmente incollati sopra è fondamentale per questo passaggio in quanto ho bisogno di avere i sensori nella stessa posizione, sia per l'allenamento, che per il test della rete neurale, se i sensori si muovessero ogni volta, addestrerei la rete su valori "traslati" rispetto alla fase di test, questo porterebbe sicuramente ad errori, non facilmente risolvibili.

Di seguito alcune foto su come ho incollato calamita e sensori, che poi collegherò alla scheda Nano come visto in precedenza in Figura 2.2.

Figura 2.4: Primo prototipo del guanto (sensore sul medio)

Figura 2.5: Sensore sul pollice

Figura 2.6: Sensore sul medio e calamita

2.2 Script Python per la Scrittura Automatica di Fogli ".csv"

Ricevendo i dati tramite seriale è possibile utilizzare una libreria python "pyserial" per la lettura di questi; il procedimento è il seguente: una volta collegata la scheda Arduino ad una porta seriale ad esempio "COM3", possiamo creare uno script python che utilizza la libreria o modulo "pyserial" in questo modo:

```
import serial

MAX_CICLES = 100
EMPTY_DATA_SRING = "b''"
INITIAL_DATA_TO_INGNORE = len("b'")
FINAL_DATA_TO_INGNORE = len("'\n\r")

arduino = serial.Serial(
    port = 'COM3',
    baudrate = 115200,
    timeout = .1
)

for _ in range(MAX_CICLES):
    data = str(arduino.readline())

    if data != EMPTY_DATA_SRING:
        return data[INITIAL_DATA_TO_INGNORE:-FINAL_DATA_TO_INGNORE]
```

Questo breve script premette solo di leggere una singola riga dal serial. Andrà ripetuto più volte per leggere tutti i dati passati dall'Arduino.

Una volta ricevuti i dati dall'Arduino, e opzionalmente salvati in una variabile, sarà possibile scriverli su un nuovo foglio ".csv".
Decidendo il percorso dove salvare il file, ad esempio
"*C:\Users\Amministratore\Cartella_Files_CSV\nuovo_file.csv*", possiamo creare il file e trascriverci i dati da una variabile utilizzando la funzione python "open(...)":

```
absolute_path = "C:\Users\Amministratore\Cartella_Files_CSV\nuovo_file
    .csv"
with open(absolute_path, "w") as file:
    file.write(
        dati_raccolti_dallArduino_tramite_la_precedente_funzione)
```

È possibile, ovviamente, ripetere tutto il procedimento quante volte vogliamo, finendo con avere a disposizione molti fogli ".csv" con all'interno i dati ricavati dai sensori tramite la scheda Arduino, dati che saranno la base dell'addestramento della rete neurale.

Per la raccolta dati nel mio specifico caso ho fatto girare lo script python sia durante uno schiocco di dita, ma anche mentre tenevo la mano ferma in varie posizioni diverse, ad esempio "mano aperta", "pugno chiuso" ecc., ed infine ho registrato anche alcune "anomalie", ovvero gesti che non devono essere classificati, questo per aiutare la rete neurale a distinguere in futuro tra uno schiocco di dita e tra un qualsiasi altro gesto, dopotutto lo scopo è far accendere dei led **solo** quando viene rivelato uno schiocco.

2.3 Utilizzo della Piattaforma Edge Impulse

Avendo a mia disposizione più di 10 minuti di dati raccolti nei file ".csv", ho creato ed addestrato la rete neurale utilizzando la piattaforma Edge Impulse:
Ho iniziato dopo la registrazione sul sito, creando un nuovo progetto, e caricando tutti i file creati in precedenza sull'apposita pagina, che si può trovare su 'Edge Impulse'→ 'Data Aquisition'→ 'Upload existing data', in questa pagina viene richiesto di caricare i file da inviare ad Edge, e scegliere un "label" che rappresenti tutti i file caricati, la label sarà poi utilizzata da Edge come classe di classificazione per la rete neurale.

Dopo viene chiesto di creare un cosiddetto "Impulso", ovvero il *workflow* che implementerà Edge per l'addestramento del modello di machine learning. Per questo progetto la struttura dell'Impulso che ho scelto è la seguente:

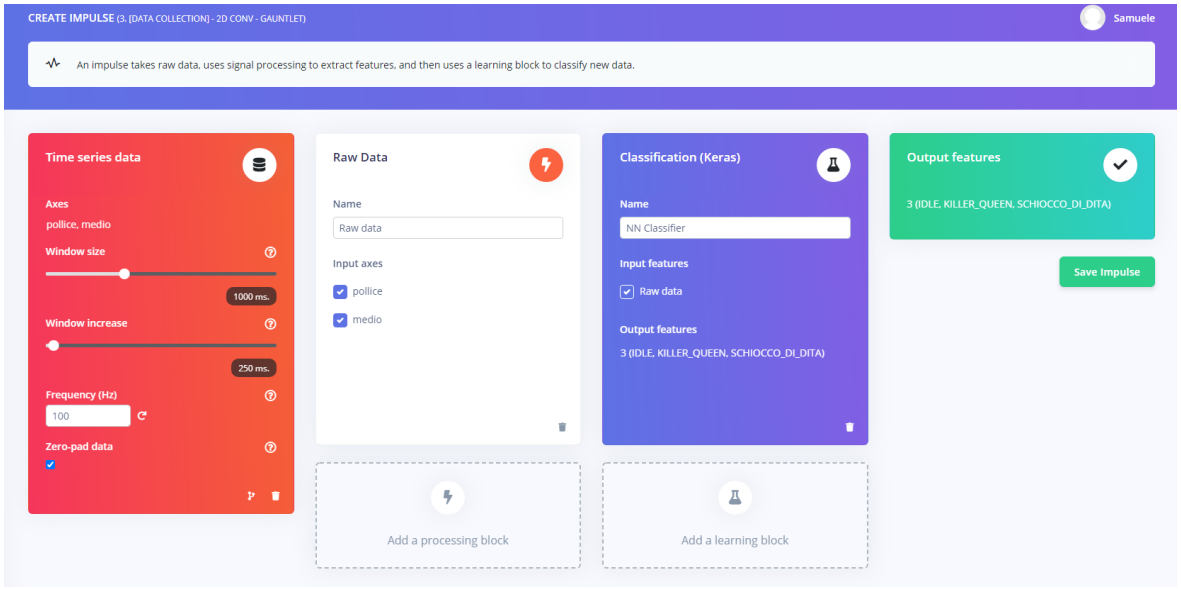


Figura 2.7: Edge Impulse - Impulso Completo

'Time Series Data', indica principalmente la grandezza del vettore in ingresso al modello, rispetto al tempo (da qui la necessità del timestamp nel Protocollo Edge Impulse).

Il 'Blocco di Processo': 'Raw Data', indica che i dati ricevuti non dovranno essere modificati prima di essere passati alla rete neurale.

Mentre il 'Blocco di Apprendimento': 'Classification (Keras)', è una rete neurale costruita in python attraverso la libreria "keras" offerta da "TenserFlow".

Le 'Output Features', come accennato poco sopra sono i label digitati durante l'upload dei file.

A questo punto per completare l'Impulso è necessario specificare lo schema della rete neurale che dovrà essere utilizzata nel modello, dopo vari tentativi per testare quale potesse essere essere il miglior approccio, sono arrivato al seguente diagramma:

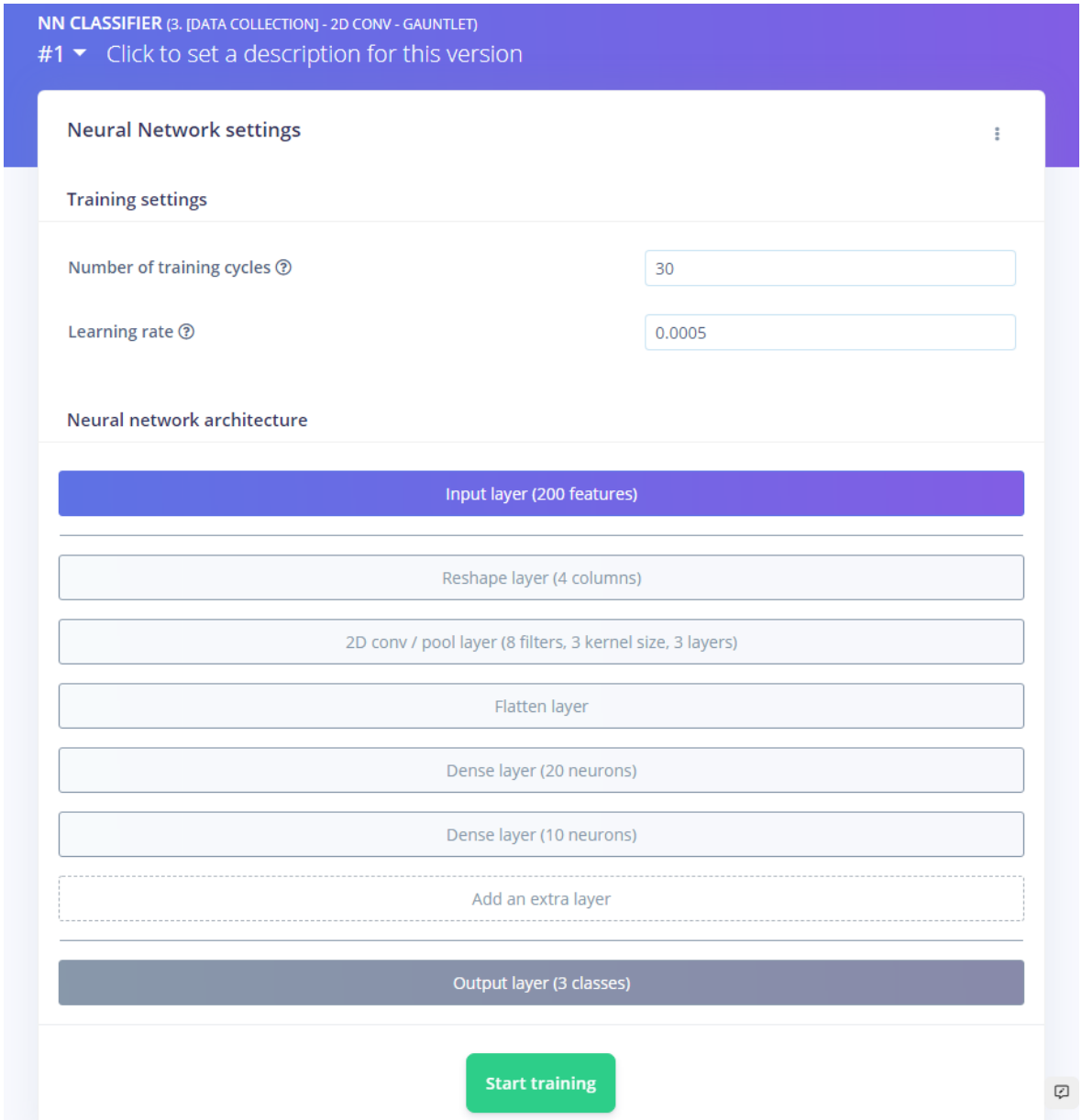


Figura 2.8: Edge Impulse - Rete Neurale

Cliccando su "Start training", dopo pochi minuti, ci viene fornita oltre all'accuratezza rispetto al set di '*cross-validation*', anche una stima delle prestazioni che avrà il modello una volta dislocato sulla piattaforma embedded scelta, ed una utile *confusion matrix*, che indica per quale/i classi il modello ha una peggiore capacità di classificazione.

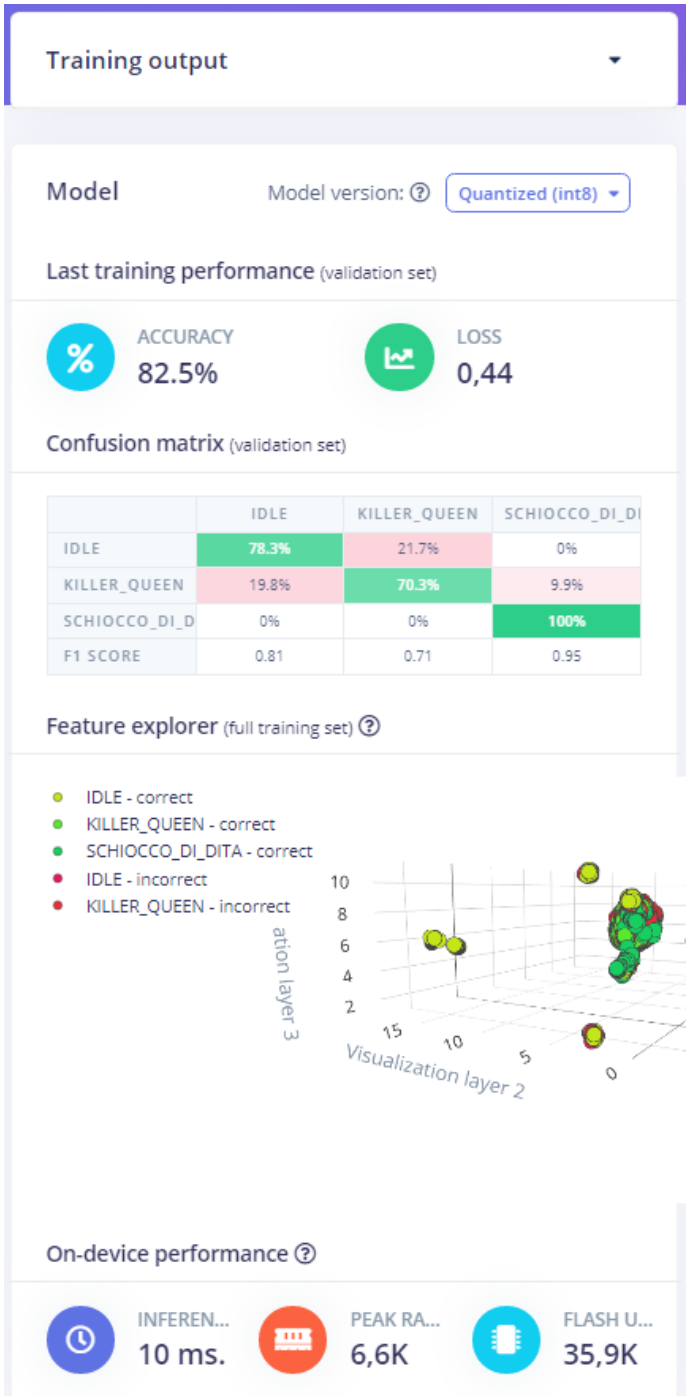


Figura 2.9: Edge Impulse - Risultati Addestramento

La rete neurale è conclusa, rimane solo da portarla sulla scheda Arduino sotto forma di modello di TensorFlow Lite.
Rechiamoci quindi, alla parte di "*Deployment*" di Edge Impulse e scegliamo l'opzione "*Arduino Library*":

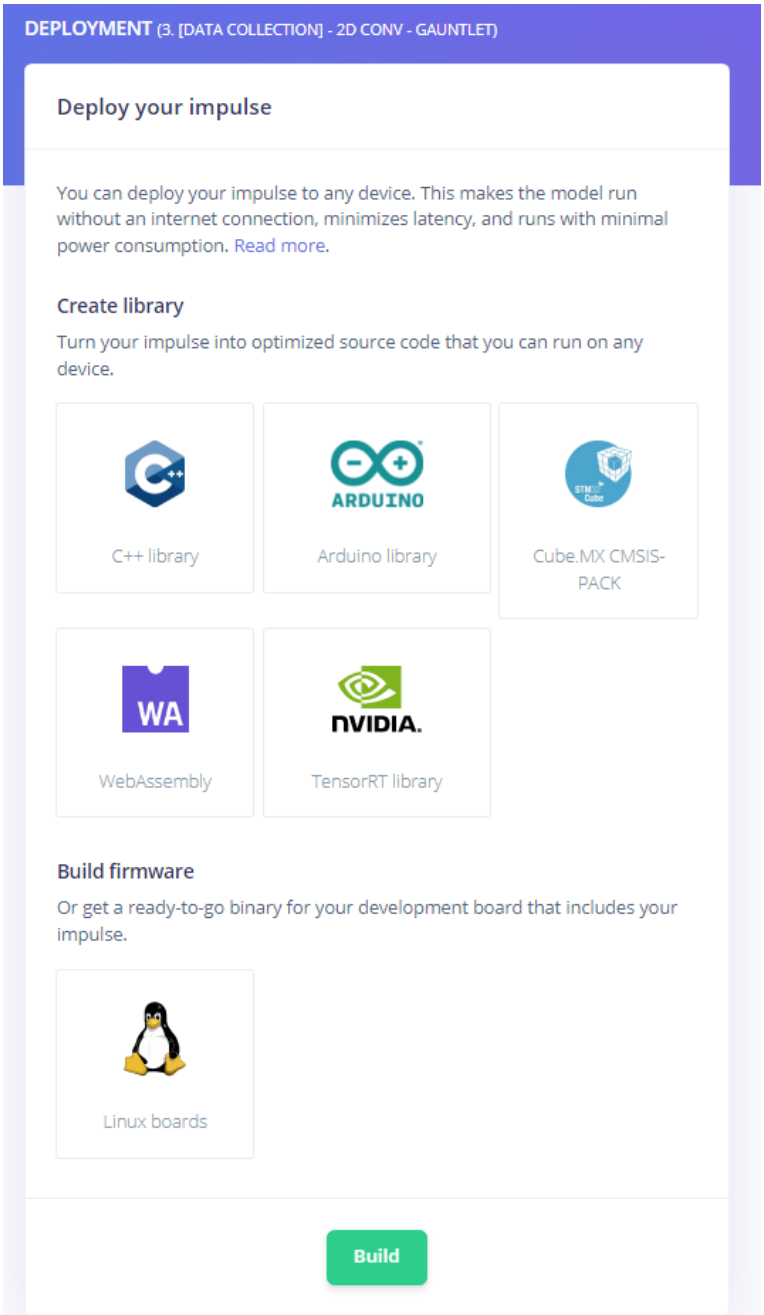


Figura 2.10: Edge Impulse - Deployment

Ci verrà fornita una libreria ".zip" che potrà essere aggiunta dall'Arduino IDE da: Sketch→ #include libreria→ Aggiungi libreria da file .ZIP...

2.4 Utilizzo della Libreria Offerta da Edge Impulse

Una volta scaricata la libreria da "Edge Impulse", la possiamo includere su un nuovo sketch Arduino, e caricare un codice esempio dalla stessa libreria, dal nome "static_buffer":

Figura 2.11: Arduino IDE - static_buffer example

```
/* Includes ---------------------------------------------------------------- */
#include <a4._Data_Collection_-_From_CSV_file_inferencing.h>

static const float features[] = {
    // copy raw features here (for example from the 'Live classification' page)
    // see https://docs.edgeimpulse.com/docs/running-your-impulse-arduino
};

/**
 * @brief      Copy raw feature data in out_ptr
 *             Function called by inference library
 *
 * @param[in]  offset   The offset
 * @param[in]  length   The length
 * @param      out_ptr  The out pointer
 *
 * @return     0
 */
int raw_feature_get_data(size_t offset, size_t length, float *out_ptr) {
    memcpy(out_ptr, features + offset, length * sizeof(float));
    return 0;
}


/**
 * @brief      Arduino setup function
 */
void setup()
{
    // put your setup code here, to run once:
    Serial.begin(115200);

    Serial.println("Edge Impulse Inferencing Demo");
}
```

In questo esempio ci è consigliato di modificare, o meglio, riempire il vettore "features[ ]" con i dati presi dai sensori, quindi utilizzando lo stesso metodo per la raccolta dati, riempiamo il vettore:

```
void loop(){
    int length_featrues = sizeof(features)/sizeof(float);
    for(int i = 1; i < length_features; i++){
        features[i-1] = features[i];
    }
    features[length_featrues-1] = new_data_just_aquired
}
```

Dopodiché nell'esempio il vettore viene passato in una serie di funzioni per poter poi essere usato dal classificatore, o rete neurale:

```
    int raw_feature_get_data(size_t offset, size_t length, float *
        out_ptr) {
        memcpy(out_ptr, features + offset, length * sizeof(float));
        return 0;
    }

    // the features are stored into flash, and we don't want
    // to load everything into RAM
    signal_t features_signal;
    features_signal.total_length = sizeof(features) / sizeof(features
        [0]);
    features_signal.get_data = &raw_feature_get_data;

    // invoke the impulse
    EI_IMPULSE_ERROR res = run_classifier(
    &features_signal, &result, false /* debug */);

    //
        ---------------------------------------------------------------------

    // TEMPI DEL CLASSIFICATORE
    // result.timing.dsp
    // result.timing.classification
    // result.timing.anomaly

    // DETTAGLI:
    // result.classification[ix].label
    // result.classification[ix].value
    // result.anomaly
```

Quindi con `result.classification[ix].label`, `result.classification[ix].value`, e `result.anomaly` è possibile implementare il resto del codice:

```
    if(result.classification[ix].label == "SCHIOCCO_DI_DITA"){
        if(result.classification[ix].value > 0.9){
            // Light the leds
            digitalWrite(A2, HIGH);

            // Empty features[] vector
            float new_blank_float[lenght_features];
            features = new_blank_float;

            delay(2000);

            // Kill the ligths
            digitalWrite(A2, LOW);
        }
    }
```

In questo caso `result.anomaly` non è stata implementata. In caso si volesse implementare è possibile, tornando alla costruzione dell'Impulso nella sezione precedente, aggiungendo un nuovo "Blocco di Apprendimento" chiamato "Anomaly Detection (K-means)" che tramite un algoritmo di K-clustering ritorna un valore, sempre tra 0 ed 1, indicativo di quanto è probabile che la classificazione trovata sia un anomalia, in caso questo se il valore fosse elevato, la classificazione va considerata errata e deve essere scartata.

Conclusioni

Il progetto, una volta finito, è in grado di rivelare la maggior parte degli schiocchi effettuati (71%): problemi sorgono tuttavia quando non si effettua uno schiocco, infatti muovendo leggermente la mano viene effettuata talvolta una lettura sbagliata, soprattutto quando viene mosso il pollice (sul quale è presente un sensore). In questo caso, il movimento non viene classificato come un anomalia ma come uno schiocco, causando l'accensione dei led. Alcune fra le possibili cause alla base di questa problematica potrebbero essere:

Il modello utilizzato è in una condizione di "overfitting", ovvero il modello si è abituato ai dati dati nel training set, e classifica in modo errato i dati effettivi; un segnale d'allarme per questo particolare caso è dato dal "Model Testing" su Edge Impulse che riporta un'accuratezza **teorica** del modello pari al 80%.

I dati sulle anomalie sono insufficienti. Semplicemente raccogliendo più dati, sopratutto su molti casi nei quali l'utente **non** schiocca le dita, è possibile allenare il modello a distinguere le due classi in maniera più accurata.

I sensori utilizzati non sono ottimali per il lavoro da eseguire; i sensori di Hall riportano una grande variazione ad un piccolo movimento se il sensore è molto vicino alla calamita (¡ 5 cm). Diversamente, ad una distanza maggiore di 5 cm il sensore rivelerà una piccolissima variazione di tensione in caso di movimento del magnete (al minimo 1.3 mV), mentre ad una distanza superiore ai 7.5 cm non verrà rilevata nessuna variazione di tensione.

Infine, i dati non stati modificati in alcun modo prima di essere passati alla rete neurale, quindi creando un nuovo "Processing Block" su Edge Impulse, oppure modificando direttamente i dati inviati dall'Arduino, è possibile facilitare l'addestramento della rete neurale.

Bibliografia

[1] "Edge Impulse - pagina iniziale", https://www.edgeimpulse.com/. URL

[2] "TensorFlow - pagina iniziale", https://www.tensorflow.org/lite. URL

[3] "Arduino Nano 33 BLE Sense", Arduino Shop,
https://store.arduino.cc/products/arduino-nano-33-ble-sense?selectedStore=eu. URL

[4] "Arduino IDE Download", Arduino, https://www.arduino.cc/en/software. URL

[5] S. Petrizzelli, "Appunti di fisica 2- effetto hall".
http://users.libero.it/sandry/Fisica2-002.pdf. URL

[6] "Effetto Hall", Wikipedia L'enciclopedia libera,
https://it.wikipedia.org/wiki/Effetto_Hall. URL

[7] Allegro Microsystems Inc. A1301 Datasheet,
A1https://docs.rs-online.com/872c/0900766b80d88057.pdf. URL

[8] "Edge impulse: un intuitivo SDK", Elettronica Open Source
https://it.emcelettronica.com/edge-impulse-un-intuitivo-sdk-per-il-tinyml-parte-i. URL

Ringraziamenti

Vorrei ringraziare tutte le persone che mi hanno aiutato a svolgere il progetto, e compilare questa tesi:

Alessandro Pozzebon.

Stefano Parrino.

Stefano Frigerio.

Garbiele Di Renzone.

Giacomo Peruzzi.