



UNIVERSITÀ
DEGLI STUDI
DI PADOVA



DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE

CORSO DI LAUREA IN INGEGNERIA DELL'INFORMAZIONE

Tecniche indirette di monitoraggio ambientale tramite embedded machine learning

LAUREANDO

Giorgio Magalini

Numero di Matricola 1189761

RELATORE

Prof. Alessandro Pozzebon

Università di Padova

CORRELATORE

Ing. Giacomo Peruzzi

Università di Padova

ANNO ACCADEMICO: 2022/2023
DATA DI LAUREA: 13/03/2023

*Alla mia famiglia e a tutti coloro
che mi hanno sempre supportato
in questo percorso di formazione*

Abstract

Il seguente progetto si concentra sullo studio di una tecnica indiretta di monitoraggio dell'inquinamento atmosferico ambientale con l'obiettivo di ottenere il dato relativo all'inquinamento a partire dal campionamento del suono inherente al rumore del traffico veicolare. Il sistema utilizzato per perseguire questo studio prevede l'utilizzo del Machine Learning Embedded e di tecniche di TinyML. Lo scopo finale è quello di rendere il processo di raccolta di questo dato più economico ed efficiente in termini energetici rispetto al processo tradizionale.

L'idea di questo progetto di tesi nasce, infatti, con il desiderio di perseguire e raggiungere tre obiettivi:

1. Riduzione dei costi;
2. Aumento dell'efficienza energetica;
3. Monitoraggio più distribuito della qualità dell'aria nelle aree più soggette a un alto livello di inquinamento.

Per realizzare questo progetto è stato necessario eseguire una profonda ricerca all'interno dell'estensiva letteratura presente in questo ambito, selezionando esclusivamente le informazioni utili al raggiungimento dello scopo di cui sopra.

Le tecniche utilizzate sono state quelle del Machine Learning Embedded e del Tiny ML; nello specifico, per quel che riguarda il Machine Learning Embedded è stata sfruttata la regressione partendo da campioni audio per estrapolare poi i valori di inquinamento misurati in PM 2.5 e PM 10 (particulate matter), mentre l'utilizzo delle tecniche di TinyML ha permesso l'utilizzo del Machine Learning all'interno di microcontrollori di ridotte dimensioni e ridotta potenza di calcolo.

Lo sviluppo di questo progetto dipende dalla popolazione del dataset, fondamentale per ottenere un regressore corretto, perciò sono state eseguite procedure quali l'acquisizione, in un luogo appositamente scelto, dei campioni audio e dei relativi valori di particolato. Durante ed in seguito alle acquisizioni sono emerse delle problematiche che hanno determinato una bassa qualità del dataset e, conseguentemente, una minor efficacia del modello ottenuto.

Infine è convenuta la scelta di Edge Impulse come piattaforma di sviluppo di algoritmi di ML per semplicità ed efficacia nella creazione di librerie utili al caricamento del modello sul microprocessore Arduino Nano utilizzato per questo progetto, tramite tecniche di Embedded Machine Learning.

Indice

1	Introduzione	1
2	Sistema di misurazione	5
2.1	Arduino Nano 33 BLE Sense	5
2.1.1	Microfono MP34DT05	6
2.1.2	Script di Arduino Nano	7
2.2	Arduino Uno	9
2.2.1	Sensore HPMA115S0	9
2.2.2	Script di Arduino Uno	10
3	Popolazione del dataset e analisi	15
3.1	Tecniche di acquisizione dati	15
3.2	Luogo di rilevazione	17
3.3	Valutazione del dataset e problematiche riscontrate	17
3.3.1	Selezione del dataset	18
4	Rete neurale	21
4.1	Edge Impulse	21
4.2	Parametri della rete neurale	23
4.2.1	Esiti dell'addestramento del modello	27
4.3	Test della rete	27
4.4	Esportazione del modello	29
5	Conclusione e progetti futuri	31
	Bibliografia	33

Segmenti di Codice

2.1	Da buffer di interi 16 bit a buffer di singoli bytes	7
2.2	Stampa su seriale del buffer di singoli bytes	8
2.3	Controllo attivazione e spegnimento PDM	8
2.4	Controllo dati PM 2.5 e PM 10	11
2.5	Media mobile	12
2.6	Inizializzazione della media mobile	12
3.1	Da coppie di bytes ad un intero con segno	16
3.2	Funzione per la lettura dei valori di particolato	16
4.1	Upload verso Edge Impulse	22

1

Introduzione

Il progetto sviluppato all'interno della seguente tesi consiste nel realizzare un modello di tipo regressore che, tramite delle acquisizioni audio relative al rumore ambientale dovuto al traffico veicolare, restituisca valori di particolato correttamente associati. In questo studio vengono discusse le metodologie utilizzate per il raggiungimento degli scopi prefissati.

I principali argomenti discussi in questo elaborato sono:

1. La realizzazione di un sistema di misurazione apposito;
2. L'accurata scelta di un luogo in cui effettuare i rilevamenti;
3. La collezione dei campioni e la conseguente popolazione di un dataset opportuno e rappresentativo della correlazione cercata;
4. Una analisi del dataset ottenuto e un'esposizione delle principali problematiche riscontrate;
5. La creazione di una rete neurale che realizzi il regressore ideato per gli scopi prefissati.

Un progetto di questo genere, implementato con tecniche di Embedded Machine Learning di TinyML, può trovare dei risvolti interessanti e utili nel presente poiché potrebbe aiutare a ridurre i costi e migliorare l'efficienza energetica nell'effettuare questo tipo di misure.

L'argomento oggetto di questo elaborato viene affrontato e suddiviso in diversi capitoli che trattano: il sistema di misurazione utilizzato, la popolazione del dataset e le conseguenti analisi, la costruzione di una rete neurale per giungere infine alle conclusioni e a uno sguardo su progetti futuri.

Partendo dal presupposto che la creazione di un sistema di misurazione efficace è fondamentale per sviluppare un progetto come quello esposto all'interno di questo elaborato, si è valutato di affidarsi ai seguenti strumenti:

- Arduino Nano 33 BLE Sense
- Microfono MP34DT05 integrato in Arduino Nano
- Arduino Uno
- Sensore di particolato HPMA115S0

Arduino Nano 33 BLE Sense è il dispositivo più adatto per la realizzazione di questo progetto. Infatti è un microcontrollore che, grazie alle sue caratteristiche hardware, permette di caricare al suo interno modelli addestrati di reti neurali e inoltre fornisce un microfono MP34DT05 integrato ideale per questo tipo di applicazioni. In questo studio vengono elencate le procedure eseguite e le tecniche adottate per adempiere all'ottenimento di un dispositivo che permetta di registrare campioni audio a singolo canale (mono) a 16 kHz.

Una buona qualità del dataset è molto importante nel contesto di addestramento di una rete neurale che svolga il compito di regressore, per questo motivo si è dedicato un capitolo che oltre a presentare le metodologie utilizzate, svolge una prima analisi rispetto al dataset ottenuto grazie a queste misurazioni e presenta anticipatamente che non risulta possibile ottenere una buona regressione a causa della scarsa rappresentatività del dataset stesso sulla correlazione tra i valori di intensità di rumore dovuta al traffico veicolare e quelli di particolato.

Si può affermare che la tipologia di algoritmi più in voga nell'ambito delle materie informatiche in questi anni, è quella delle reti neurali. Queste permettono la risoluzione di problemi computazionali che, in molti casi, algoritmi di altra tipologia non riescono a risolvere. In questa testi si dimostra un esempio applicativo di questo tipo di algoritmo e, nello specifico, si presenta la tipologia chiamata convolutional neural network (CNN) che in casi come questo, dove è necessario processare ed analizzare segnali audio e la loro rappresentazione in spettrogrammi, risulta essere una delle migliori scelte tra tutte le possibili.

Infine vengono effettuate delle considerazioni conclusive riguardo a tutto ciò che è possibile migliorare nella realizzazione di questo progetto. Nello specifico

CAPITOLO 1. INTRODUZIONE

viene discusso che l'aumento del dataset è sicuramente una delle principali soluzioni. Queste migliorie hanno come primo obiettivo quello di rendere questo progetto funzionante in un futuro prossimo.

2

Sistema di misurazione

Per lo sviluppo di questo progetto è fondamentale il dataset e, di conseguenza, è importante raccoglierlo nella maniera più consona all'utilizzo che ne deve essere fatto ovvero quello di riuscire a misurare la qualità dell'aria in termini di concentrazione di particolato, acquisire il rumore generato dal traffico veicolare e, successivamente, correlare queste due misure; per fare ciò sono stati quindi utilizzati diversi strumenti:

- Arduino Nano 33 BLE Sense
- Microfono MP34DT05 integrato in Arduino Nano
- Arduino Uno
- Sensore di particolato HPMA115S0

2.1 ARDUINO NANO 33 BLE SENSE

Il dispositivo utilizzato in misura maggiore è stato Arduino Nano 33 BLE Sense [1] e la scelta di questo specifico dispositivo è stata principalmente motivata dalle sue specifiche tecniche quali:

- Microfono MP34DT05 integrato
- Compatibilità con le librerie di Tensorflow Lite
- Compatibilità con EdgeImpulse

2.1. ARDUINO NANO 33 BLE SENSE

- Dimensioni ridotte
- Costo ridotto

2.1.1 MICROFONO MP34DT05

Sulla board Arduino è presente un microfono MP34DT05 [2]; tale dispositivo è ideale per un progetto come questo poiché rientra nella categoria dei microfoni omnidirezionali, ovvero in grado di trasdurre la pressione sonora producendo un segnale elettrico la cui intensità non dipende dalla direzione da cui proviene la fonte sonora. Questa caratteristica risulta quindi essere molto utile in quanto il dispositivo finale può essere comodamente installato indiscriminatamente rispetto alla direzione da cui proviene la principale fonte di rumore. Il microfono MP34DT05 è un microfono MEMS¹.

Un microfono MEMS converte onde di pressione sonora in un segnale elettrico ed è composto da tre parti principali: il trasduttore a membrana, uno stadio amplificatore ed infine un modulatore PDM². Grazie alla sua membrana il trasduttore è capace di convertire il segnale acustico in un segnale elettrico analogico. Il segnale elettrico analogico viene poi amplificato dallo stadio di amplificazione e successivamente modulato dal modulatore grazie ad un clock a 16 kHz, generato a bordo dell'Arduino. Dal momento in cui viene attivato il modulatore PDM i dati acquisiti vengono salvati in un buffer in tempo reale.

La modulazione di tipo PDM è a singolo bit, ad alta frequenza e integra la tecnica del *noise shaping* [4]. La libreria Arduino che gestisce la conversione da PDM in PCM³ è la libreria PDM.h. Nella PCM il segnale audio viene rappresentato da una serie di campioni, nel caso in questione si tratta di campioni a 16bit, i quali vengono convertiti in modulazione PCM vengono convertiti dalla libreria in modo tale da essere accessibili e disponibili per il trasferimento e successivamente essere salvati come file audio.

¹Micro-Electro-Mechanical Systems [3].

²Pulse-density modulation.

³Modulazione ad Impulsi Codificati

2.1.2 SCRIPT DI ARDUINO NANO

Lo script implementato su Arduino ha come scopo il trasferimento dei campioni audio registrati dal microfono dell'Arduino al computer, dove successivamente verranno associati ai valori di particolato ed infine usati per l'addestramento della rete neurale. Sulla base dell'esempio di libreria PDMSerialPlotter della libreria PDM.h di Arduino, è stato creato lo script "PDM Serial Sender.ino" [5] per trasferire i buffer audio da Arduino sulla porta seriale, permettere quindi il loro salvataggio e consecutivamente popolare il dataset. Nell'esempio di libreria i campioni vengono inviati tramite la funzione Serial.println(). Questa funzione, prima di inviare i bytes li converte in ASCII e poi li invia. Una conversione in ASCII come questa impedisce l'utilizzo delle massime potenzialità della comunicazione seriale, infatti non vengono raggiunti i 16000 campioni al secondo resi disponibili dal microfono. Differentemente dall'esempio della libreria, è stato modificato il comando per cui vengono trasmessi i buffer tramite seriale.

È stato deciso pertanto di utilizzare il comando SerialUSB.write() che non converte i dati in ASCII ed inoltre permette l'invio di array di byte e non soltanto singole coppie; è stata necessaria una conversione dell'array nel quale il microfono salvava i dati del campionamento da 2 byte ad 1 byte. Questa conversione è stata implementata semplicemente iterando queste righe di codice sul completo array di interi.

```

1 sampleBuffer_8bit[sb_index] = '\n';
2 sb_index++;
3 sampleBuffer_8bit[sb_index] = (sampleBuffer[i] >> 8) &
4 0xFF;
5 sampleBuffer_8bit[sb_index] = (sampleBuffer[i] & 0xFF)
6 ;
7 sb_index++;

```

Codice 2.1: Da buffer di interi 16 bit a buffer di singoli bytes

In questo modo l'array sampleBuffer 8bit sarà la versione a 8 bit dell'array sampleBuffer. Dove ogni elemento a 16 bit del buffer sampleBuffer è rappre-

2.1. ARDUINO NANO 33 BLE SENSE

sentato con due elementi a 8 bit nel buffer sampleBuffer 8bit. È stato aggiunto anche un carattere speciale "\n" che permettesse di separare i singoli campioni.

Si noti che l'indice "sb_index" aumenta di tre volte per ogni iterazione sull'indice "i" del buffer ad interi, perciò la dimensione del buffer a singolo byte risulta tripla rispetto a quella del buffer a interi (due byte). Per poter interpretare i dati trasmessi, una volta letti, dovranno essere riconvertiti a interi.

Per il trasferimento tramite porta USB è stato sviluppato il seguente codice:

```
1 if (sb_index >= 1536) {  
2     SerialUSB.write(sampleBuffer_8bit, sb_index);  
3     sb_index=0;  
4 }  
5
```

Codice 2.2: Stampa su seriale del buffer di singoli bytes

Se l'indice "sb_index" raggiunge il valore di 1536, viene chiamata la funzione SerialUSB.write() che invia il buffer a 8 bit nella sua integrità in una volta sola. Così facendo è stato possibile raggiungere la velocità 16000 campioni/s permettendo un salvataggio in tempo reale dell'audio del microfono.

Il codice implementato permette di inviare il buffer solamente quando è richiesto dalla seriale con cui il computer ed Arduino comunicano. Infatti solamente quando è necessario viene inviato il byte "1" e quindi attivato il microfono tramite PDM.begin(). Quando sono state letti tutti i sample viene inviato il byte "2" e il microfono viene disattivato tramite il comando PDM.end(). Questa tecnica garantisce il mantenimento della contemporaneità delle acquisizioni audio che rischierebbe di venire a meno nel caso in cui per qualsiasi motivo esterno (per esempio per il ritardo dovuto all'acquisizione dei valori di particolato), venga perso del tempo tra una registrazione e la successiva.

```
1 if (Serial.available() > 0) {  
2     pythonInput = Serial.read();  
3     if (pythonInput == '1') {  
4         if (!PDM.begin(channels, frequency)) {  
5             SerialUSB.println("Failed to start PDM!");  
6             while (1);  
7         }
```

```

8      else {
9          delay(5000);
10         }
11     }
12     if (pythonInput == '2') {
13         PDM.end();
14     }
15 }
16

```

Codice 2.3: Controllo attivazione e spegnimento PDM

2.2 ARDUINO UNO

Per effettuare le misure di particolato è stato necessario programmare anche una scheda Arduino Uno con il solo scopo di interfacciarsi con il sensore HPMA115S0 attraverso una comunicazione tradotta da un convertitore di livelli logici; la scheda Arduino Uno infatti, grazie alla sua architettura, permette l'utilizzo della libreria SoftwareSerial.h che non è disponibile nella programmazione della scheda Arduino Nano poiché incompatibile a livello hardware.

L'algoritmo implementato su questa scheda, è stato utile a permettere un dialogo seriale tra computer e sensore in contemporanea all'acquisizione dei file audio effettuate con Arduino Nano che vengono comunicate, anche queste, tramite seriale ma attraverso un'altra porta.

2.2.1 SENSORE HPMA115S0

Le misure di particolato sono state fatte con il sensore HMP il quale funziona tramite laser scattering. Un piccolo e silenzioso sistema di ventilazione aspira un flusso d'aria che attraversa le cavità interne del sensore e raggiunge il punto in cui viene effettuata la misurazione. Il fascio di un laser all'interno del sensore viene più o meno riflesso dal particolato in base alla quantità di particelle presenti nel flusso d'aria campionato. Questo fenomeno viene chiamato *scattering* [6] e la sua quantità di scattering viene associata direttamente ad un valore di PM 2.5

2.2. ARDUINO UNO

e PM 10; quando viene interrogato, il sensore, trasmette questi valori tramite protocollo seriale.

Convertitore di livelli logici Il sensore in questione è alimentato a 5 V e invia segnali seriali attraverso la sua porta rx e tx con livelli logici a 3,3 V. Per tale ragione è stato necessario usare un convertitore di livelli logici che alzasse i valori di tensione dal sensore ad Arduino e viceversa. Il convertitore in questione, che sfrutta la tecnologia TTL⁴, è un piccolo dispositivo che riduce in sicurezza segnali digitali da 5 V a 3,3 V o li aumenta da 3,3 V a 5 V. Il convertitore di livello è molto facile da predisporre all'utilizzo, infatti è sufficiente alimentarlo dalle due fonti di tensione (alta e bassa tensione) utilizzate dal sistema: alta tensione (5 V) al pin "HV", bassa tensione (3,3 V) a "LV" e poi collegandolo a terra tramite il pin "GND". Per effettuare la connessione seriale opportunamente convertita è necessario collegare la porta seriale funzionante al livello logico basso ai pin che sono etichettati come: ("LV1", "LV2", ... , "LVj"), mentre quella funzionante al livello logico basso ai pin etichettati: ("HV1", "HV2", ..., "HVj"), dove "LVj" sarà corrispondente alla versione a livello logico abbassato del segnale presente su "HVj". Il collegamento corretto prevede inoltre che il segnale emesso dal pin di trasmissione seriale tx di Arduino convertito al valore basso venga collegato all'ingresso per la ricezione seriale del sensore HPMA1150 e viceversa.

Inviando dei comandi specifici tramite protocollo seriale questo sensore restituisce i valori di PM 2.5 e PM 10. La sua risposta può essere ritenuta valida dopo 6 secondi [8]. Per la lettura dei valori forniti dal sensore è stato utilizzato un esempio fornito dalla libreria HPMA115S0.h [9] e sono state modificate le librerie affinché venissero stampati in seriale esclusivamente i dati essenziali, anziché stringhe di status dei campionamenti.

2.2.2 SCRIPT DI ARDUINO UNO

Lo script caricato su Arduino Uno ha lo scopo di comunicare i dati misurati tramite il sensore di particolato e inviarli sulla seriale una volta richiesti. Il codice, sviluppato a partire da un esempio di libreria estratto dalla libreria HPMA115S0.h, stampa in output seriale i valori di PM 2.5 e PM 10 e, inoltre,

⁴Transistor-transistor logic [7].

le informazioni riguardo la riuscita dell'acquisizione delle misure. La libreria è stata quindi modificata per ottenere solamente i valori numerici delle misure.

Per evitare letture errate sulla porta seriale, il codice implementato [10], permette di inviare le misure solo su richiesta. Infatti se Arduino riceve dal computer il byte "1" invia in risposta il valore di PM 2.5, se riceve il byte "2" invia quello di PM 10.

```

1   if (Serial.available() > 0) {
2       pythonInput = Serial.read();
3       if (pythonInput == '1') {
4           if (hpma115S0.ReadParticleMeasurement (&pm2_5, &
5               pm10)) {
6                   Serial.println(MovingAveragePm2_5(pm2_5));
7               }
8           if (pythonInput == '2') {
9               if (hpma115S0.ReadParticleMeasurement (&pm2_5, &
10                  pm10)) {
11                     Serial.println(MovingAveragePm10(pm10));
12                 }
13             }
14         if (hpma115S0.ReadParticleMeasurement (&pm2_5, &
15            pm10)) {
16             MovingAveragePm2_5(pm2_5);
17             MovingAveragePm10(pm10);
18             delay(1000);
19         }

```

Codice 2.4: Controllo dati PM 2.5 e PM 10

La misura di particolato effettuata secondo l'esempio di libreria estratto dalla libreria hpma115S0.h avviene con una frequenza di una misura al secondo. Nel caso di studio è stato considerato che la misura istantanea del valore particolato non includesse i valori passati e perciò è possibile che successivamente ad un'acquisizione audio vengano salvati dei valori poco coerenti a quelli presenti

2.2. ARDUINO UNO

nel trascorso lasso di tempo definito dalla lunghezza del file audio registrato. Per incrementare la precisione sulle misure di particolato e permettere che queste includano sufficienti valori passati di particolato, è stata implementata una media mobile. Il numero di valori inclusi nel buffer della media mobile è stato scelto affinché la finestra temporale della media mobile coprisse interamente la lunghezza del file audio. Infatti registrando file audio da 10 s e avendo una misura di particolato ogni secondo, la quantità di valori scelta è stata 10.

Per implementare la media mobile è stato sviluppato la seguente funzione (identica per i valori di PM 2.5):

```
1  unsigned int MovingAveragePm10(unsigned int pm10) {  
2      sumPm10 -= readingsPm10[index10];           // Remove  
the oldest entry from the sum  
3      readingsPm10[index10] = pm10;                // Add the  
newest reading to the window  
4      sumPm10 += pm10;                            // Add the  
newest reading to the sum  
5      index10 = (index10 + 1) % WINDOW_SIZE;       //  
Increment the index, and wrap to 0 if it exceeds the  
window size  
6      averagedPm10 = sumPm10 / WINDOW_SIZE; // Divide the  
sum of the window by the window size for the result  
7      return (unsigned int)averagedPm10;  
8  }  
9
```

Codice 2.5: Media mobile

Dal momento in cui viene inizializzato Arduino per la lettura dei valori di PM, è opportuno considerare valida la media mobile soltanto dopo il trascorriamento del lasso temporale prefissato dalla dimensione della finestra della media mobile. Pertanto, per far sì che i valori iniziali vengano correttamente calcolati, è stata implementata la seguente funzione che verrà chiamata nella fase di setup di Arduino Uno.

```
1  void MovingAverageInit() {  
2      Serial.println("M.A. init");  
3      for (int i = 0; i < WINDOW_SIZE; i++) {
```

```
4         if (hpma115S0.ReadParticleMeasurement(&pm2_5, &
5             pm10)) {
6             MovingAveragePm2_5(pm2_5);
7             MovingAveragePm10(pm10);
8             }
9             delay(1000);
10        }
11    }
12 }
```

Codice 2.6: Inizializzazione della media mobile

3

Popolazione del dataset e analisi

L'acquisizione dei campioni per questo progetto si basa sulla popolazione di un dataset descrittivo della correlazione attesa tra rumore e inquinamento; per fare ciò è stato necessario sviluppare uno script in linguaggio Python e definire un luogo dove operare le rilevazioni. In seguito alle misure sono state fatte delle valutazioni in merito al luogo e alla qualità del dataset che hanno in parte compromesso la buona riuscita del progetto per come era stato ideato.

3.1 TECNICHE DI ACQUISIZIONE DATI

Il dataset adatto a questo tipo di progetto e raccolto è una collezione di file audio della durata di 10s che integrano l'informazione relativa ai livelli di particolato. La tecnica scelta è stata quindi quella di salvare i file audio che avessero, nel nome con cui sono stati salvati, i dati di PM 2.5 e PM 10.

SCRIPT PYTHON

Per permettere l'acquisizione dei valori di particolato in contemporanea ai campioni audio da salvare, è stato sviluppato uno script [11] in linguaggio Python che:

1. Attiva il PDM dell'Arduino Nano;
2. Registra 10 s di file audio trasmessi da Arduino su seriale;
3. Misura e salva in una stringa i valori di particolato;

3.1. TECNICHE DI ACQUISIZIONE DATI

4. Salva il file audio da 10 s con timestamp e particolato come nome.

Per tradurre i bytes in un unico file audio è stato implementato il seguente codice:

```
1  for x in range(samples):
2      nano.read_until()
3      cc1 = nano.read(2)
4      value = int.from_bytes(cc1, "big", signed = True)
5      data = struct.pack("<h", value)
6      obj.writeframesraw(data)
7
```

Codice 3.1: Da coppie di bytes ad un intero con segno

Per effettuare le misure di qualità dell'aria è stato implementato il seguente codice:

```
1  def getPmValues(i):
2      uno.reset_input_buffer()
3      arduinoData = 0
4      if i == 0:
5          uno.write(b'1')
6          uno.reset_input_buffer()
7          arduinoData = uno.readline().decode("utf-8")
8      if i == 1:
9          uno.write(b'2')
10         uno.reset_input_buffer()
11         arduinoData = uno.readline().decode("utf-8")
12     return int(arduinoData)
13
```

Codice 3.2: Funzione per la lettura dei valori di particolato

In generale è importante che la misura di particolato venga effettuata solamente dopo la registrazione dei file audio poiché il livello di qualità dell'aria nel presente è causato dall'intensità di rumore nel passato e non viceversa. Considerando che vengono analizzati solamente 10s, il dettaglio sopra riportato influisce di meno poiché, in questa applicazione, la rete neurale è addestrata ignorando la quantità di ritardo di causalità.

3.2 LUOGO DI RILEVAZIONE

Inizialmente sono stati valutati diversi possibili luoghi in cui effettuare le misure, ma facendo queste prove è emerso che il sensore HPMA115SO non riesce a rispondere in maniera istantanea al passaggio di un singolo veicolo.



Figura 3.1: Luogo delle misurazioni.

È stato quindi ipotizzato che, i fenomeni aerodinamici causati dal passaggio di un singolo veicolo, sono tali da non permettere l'ottenimento di valori di particolato associati al veicolo stesso e, data questa osservazione, si è convenuto di scegliere un parcheggio a ridosso di un semaforo a Verona (VR) in cui è stato ipotizzato che la qualità dell'aria segua l'andamento delle variazioni di intensità di traffico giornaliere.

Allo stesso modo è stata fatta un'ipotesi relativa alla possibilità che, questa intensità di traffico, sia associata alla quantità di rumore ambientale.

3.3 VALUTAZIONE DEL DATASET E PROBLEMATICHE RISCONTRATE

Nel corso delle rilevazioni sono state rinvenute alcune problematiche che hanno inciso sulla valutazione del dataset.

Tali problematiche sono riassumibili in:

3.3. VALUTAZIONE DEL DATASET E PROBLEMATICHE RISCONTRATE

- Dipendenza dalla variabilità del clima;
- Ridotta dimensione del dataset;
- Ritardo di causalità.

Nello specifico il clima influenza i valori di PM presenti nell'aria e, per una più corretta e precisa raccolta dei dati, sarebbe stato preferibile un clima di costante bel tempo nei giorni di rilevazione dei dati. Così non è stato poiché nella notte tra il 24 e il 25 febbraio 2023 ha piovuto, mentre la mattina del 25 febbraio 2023 c'è stato molto vento; entrambe queste variazioni climatiche hanno inciso sulla raccolta dei dati abbassando notevolmente i valori di particolato. Per quanto riguarda la dimensione del dataset, il poco tempo a disposizione ha impedito di popolare estensivamente quest'ultimo; una più ampia dimensione del dataset avrebbe aiutato ad escludere fenomeni esterni, come ad esempio i fenomeni climatici, poiché potendo includerne una quantità più grande si sarebbe potuto macroscopicamente escluderli. Infine, ipotizzando come in precedenza che l'intensità di rumore sia direttamente associata alla quantità di traffico, non è determinato quale sia il lasso di tempo che trascorre tra un certo livello di traffico a cui si associa il relativo rumore e l'aumento di particolato che ne è causa.

Una più accurata ricerca può quindi aiutare a stimare un valore di tempo associabile al fenomeno del ritardo di causalità per fare in modo che le previsioni del modello nel presente possano essere considerate valide soltanto una volta trascorso questo tempo. Inoltre è possibile affrontare la correttezza dell'altra ipotesi precedentemente fatta, relativa alla linearità tra quantità di traffico e rumore, allo scopo di studiare quale sia la reale interazione tra queste due variabili.

3.3.1 SELEZIONE DEL DATASET

Considerate le misure ottenute e le problematiche appena discusse, il dataset relativo al giorno 27 febbraio 2023 risulta essere il più estensivo e continuativo, inoltre, in questo giorno, il particolato ha avuto un andamento maggiormente correlato all'intensità di traffico.

È opportuno però considerare che la sensibilità del sensore non garantisce una buona precisione all'interno di misure con valori così bassi.

CAPITOLO 3. POPOLAZIONE DEL DATASET E ANALISI

Si osservi i grafici scatter plot sotto riportati, relativi ai dati di particolato raccolti, per l'andamento dei dati e la dimensione del dataset.

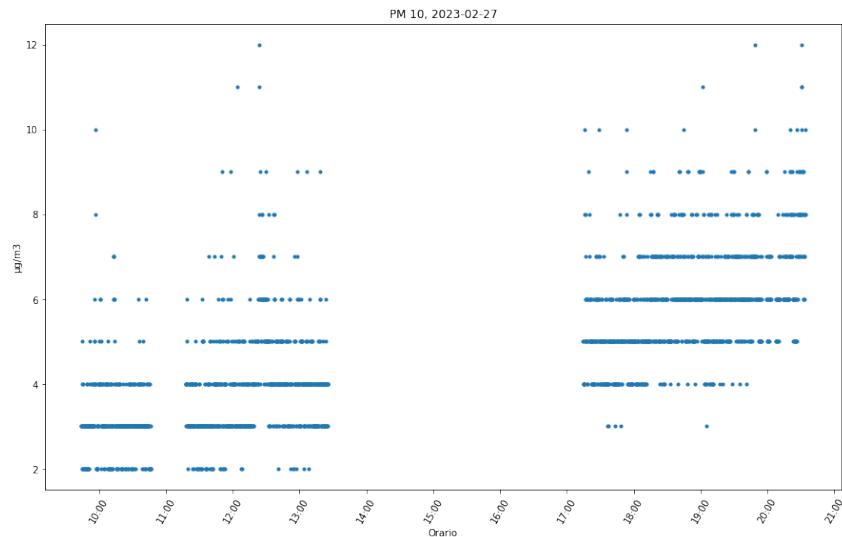


Figura 3.2: Grafico de valores dei PM 10.

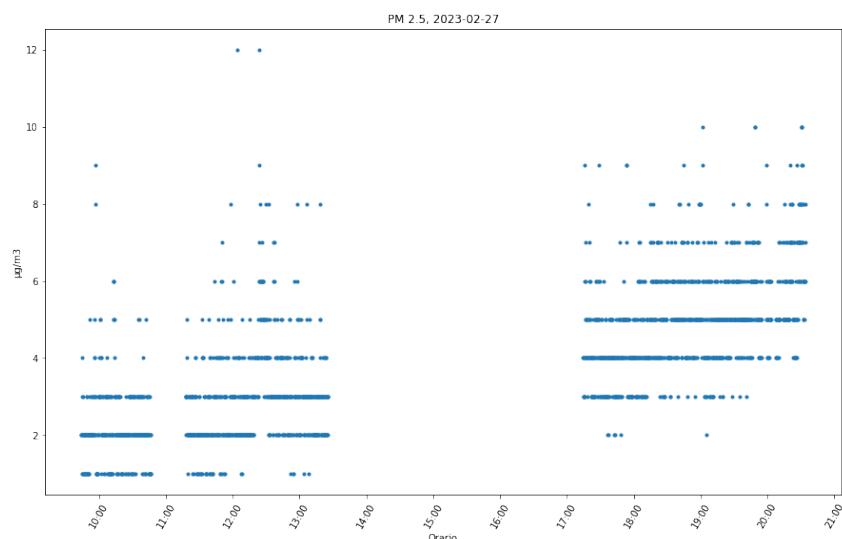


Figura 3.3: Grafico de valores dei PM 2.5.

4

Rete neurale

Nel campo dell'apprendimento automatico, una rete neurale artificiale (in inglese artificial neural network, abbreviato in ANN o anche come NN) è un modello computazionale composto di "neuroni" artificiali, ispirato vagamente dalla semplificazione di una rete neurale biologica. Questi modelli matematici sono troppo semplici per ottenere una comprensione delle reti neurali biologiche, ma sono utilizzati per tentare di risolvere problemi ingegneristici di intelligenza artificiale come quelli che si pongono in diversi ambiti tecnologici (in elettronica, informatica, simulazione, e altre discipline) [12].

Dopo aver svolto una serie di tentativi con le librerie di tensor flow in locale, è stata presa in considerazione la piattaforma online Edge Impulse con l'obiettivo di sviluppare una regressione tramite rete neurale convoluzionale (CNN¹).

4.1 EDGE IMPULSE

Edge Impulse è una piattaforma online che permette di:

- Caricare i propri dataset;
- Personalizzare i propri dataset;
- Sviluppare modelli di reti neurali e algoritmi e ottimizzarli;

¹Convolutional neural network.

4.1. EDGE IMPULSE

- Poter testare la performance del modello.

Nello specifico, per questo progetto, Edge Impulse è stato usato per creare una regressione tramite rete neurale convoluzionale. Edge Impulse fornisce la possibilità di caricare i propri campioni (in questo caso campioni audio) con la rispettiva etichetta (label).

In questo caso è stato implementato il codice UploadToEI.py [13] per caricare i campioni audio acquisiti sulla piattaforma Edge Impulse tramite il servizio di ingestion API fornito dalla stessa. È necessario eseguire i seguenti passaggi:

1. Selezionare la cartella in cui sono stati salvati i file;
2. Estrarre il valore di particolato immagazzinato nel nome del file audio per ogni file all'interno cartella;
3. Indicare la chiave (key) del progetto creato;
4. Effettuare un caricamento tramite il comando (`requests.post`) in cui viene caricata l'etichetta e la chiave all'interno del campo chiamato `headers`;
5. Aprire file tramite il comando (`open`) e inserirlo nel campo `files`.

Se il comando va a buon fine la funzione ritorna al valore 200 e i file sono stati caricati correttamente sul progetto

```
1  headers = {  
2      'x-api-key': '', # SET your Edge Impulse project API Key  
3      'x-label': label1 # insert label  
4  }  
5  
6  files = [ ('data', open(path_to_single_file, 'rb')) ] # read files  
7  
8  res = requests.post('https://ingestion.edgeimpulse.com/api/  
9    training/files', headers = headers, files = files) # upload  
10  
11  #print status  
12  if (res.status_code == 200):  
13      print('Uploaded file(s) to Edge Impulse\n', res.status_code,  
14      res.content)  
15  else:  
16      print('Failed to upload file(s) to Edge Impulse\n', res.  
17      status_code, res.content)
```

Codice 4.1: Upload verso Edge Impulse

In Edge Impulse è stato notato che l'etichetta può essere di un singolo valore e non è stato perciò possibile realizzare una rete neurale che restituisse un output bidimensionale pertanto si è deciso di costruire due reti neurali; una per il PM 2.5 e una per il PM 10 affinché possano essere elaborati entrambi i due diversi parametri, indici di particolato.

4.2 PARAMETRI DELLA RETE NEURALE

I parametri della rete neurale sono decisivi per ottenere un modello adatto ai propri scopi e spesso vengono testate varie combinazioni senza sapere a priori quali di queste siano la soluzione migliore per il proprio scopo.

Come prima fase di settaggio dei parametri c'è la creazione dell' "impulse" che non è altro che la pipeline della rete neurale. Il primo elemento della pipeline della rete neurale è la time series data che permette di creare della data argumentation spezzettando i file di ingresso in finestre temporali tramite i parametri Window Size² e Window Increase³.

Il primo blocco di data processing che viene effettuato, chiamato Audio (MFE), estrae uno spettrogramma dalle finestre di file audio di ingresso usando un banco di filtri Mel che estraggono features ottime per un contesto in cui si analizzano audio privi di contenuti simili a quelli del parlato. L'ultimo blocco, il "learning block", è quello relativo al ML che effettua una regressione che produce un singolo output avendo in input le caratteristiche (features) estratte dal blocco Audio (MFE).

Dopo numerosi tentativi effettuati per entrambe le reti, non sono stati trovati risultati migliori di quelli ottenuti tramite le seguenti impostazioni, settate sia per la rete sui valori PM 2.5 che per quella sui valori di PM 10.

²Dimensione della finestra che slitta sul file audio.

³Dimensione del passo di slittamento tra le finestre.

4.2. PARAMETRI DELLA RETE NEURALE

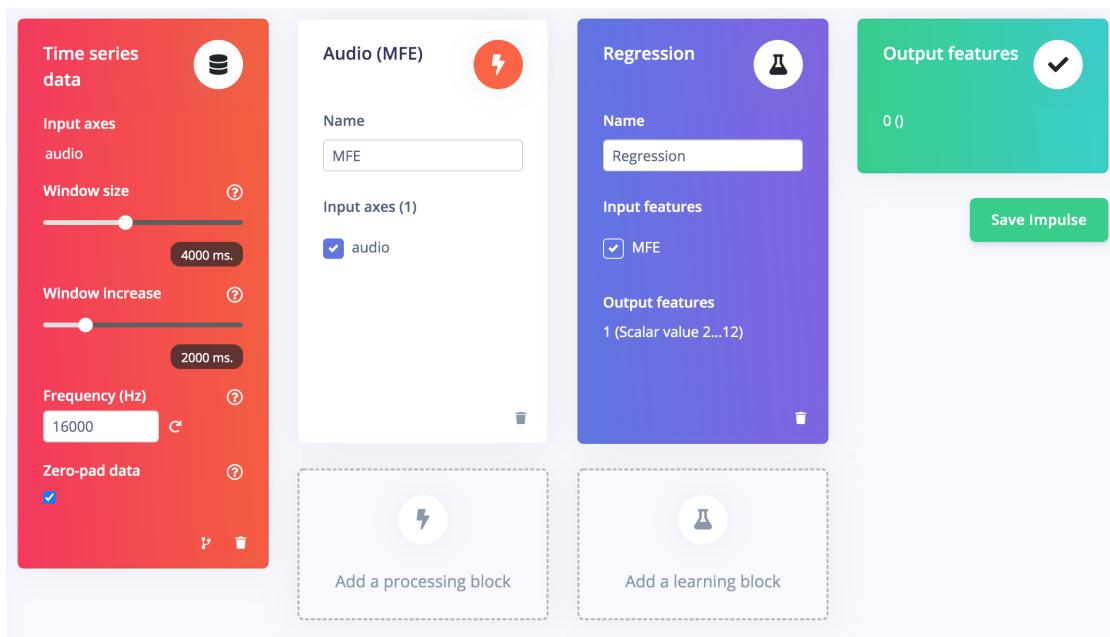


Figura 4.1: Impulse e blocco "Time Series Data".

Parameters

Mel-filterbank energy features

Frame length	0.02
Frame stride	0.01
Filter number	40
FFT length	512
Low frequency	50
High frequency	Click to set

Normalization

Noise floor (dB)	-70
------------------	-----

Figura 4.2: Blocco di processing "MFE Audio".

Neural Network settings

Training settings

Number of training cycles ⓘ 100

Learning rate ⓘ 0.005

Validation set size ⓘ 20 %

Auto-balance dataset ⓘ

Neural network architecture

Architecture presets ⓘ [1D Convolutional \(Default\)](#) [2D Convolutional](#)

Input layer (15,960 features)

Reshape layer (40 columns)

1D conv / pool layer (8 neurons, 3 kernel size, 1 layer)

Dropout (rate 0.25)

1D conv / pool layer (16 neurons, 3 kernel size, 1 layer)

Dropout (rate 0.25)

Flatten layer

Add an extra layer

Output layer (1 classes)

Figura 4.3: Blocco di regressione

4.2. PARAMETRI DELLA RETE NEURALE

Le impostazioni della regressione tramite rete neurale convoluzionale 1D hanno portato a delle quantità di parametri riassunte in questa tabella:

Model: "sequential"		
Layer (type)	Output Shape	Param #
reshape (Reshape)	(None, 399, 40)	0
conv1d (Conv1D)	(None, 399, 8)	968
max_pooling1d (MaxPooling1D)	(None, 200, 8)	0
dropout (Dropout)	(None, 200, 8)	0
conv1d_1 (Conv1D)	(None, 200, 16)	400
max_pooling1d_1 (MaxPooling1 (MaxPooling1	(None, 100, 16)	0
dropout_1 (Dropout)	(None, 100, 16)	0
flatten (Flatten)	(None, 1600)	0
y_pred (Dense)	(None, 1)	1601
<hr/>		
Total params: 2,969		
Trainable params: 2,969		
Non-trainable params: 0		

Figura 4.4: Sommario della rete neurale e numero complessivo di parametri.

4.2.1 ESITI DELL'ADDESTRAMENTO DEL MODELLO



Figura 4.5: Output dell'addestramento delle reti

4.3 TEST DELLA RETE

È stato ottenuto un RMSE⁴ che vale $1.66 \mu\text{g}/\text{m}^3$ per il modello sui PM 2.5 e vale $1.55 \mu\text{g}/\text{m}^3$ per il modello sui PM 10. La precisione viene calcolata tramite un test della rete fatto con gli input di test e una threshold (soglia) che permette di validare o meno gli output del modello. La soglia definisce un intervallo di confidenza entro il quale gli output del modello vengono considerati buoni al 100%.

Considerato che la threshold può essere impostata arbitrariamente e quindi che il livello di accuratezza varia a seconda della soglia, si può giungere alla conclusione che tale threshold è utile principalmente nella fase di training del modello. In questa fase, la parte di dataset dedicata alla "validation" viene appunto usata per validare o meno gli output che la rete genera. La severità con cui

⁴Root mean squared error, radice quadrata dell'errore quadratico medio

4.3. TEST DELLA RETE

viene categorizzato un determinato output in accettabile o meno è determinata dalla dimensione dell'intervallo di confidenza definito dal valore della soglia. Infatti, impostando questa soglia a dei valori più bassi, l'addestramento può subire una valutazione più accurata permettendo quindi di perfezionarlo in una certa misura.

Le figure esposte in seguito hanno un valore di soglia pari al 10% del range di input delle etichette: 1.1 per il modello sui PM 2.5 e 1 per quello sui PM 10. Questo valore è stato impostato automaticamente dalla piattaforma EdgeImpulse e, come previsto dalle osservazioni fatte sulle problematiche riscontrate, mostrano un'accuratezza notevolmente bassa nella previsione dei valori di particolato.

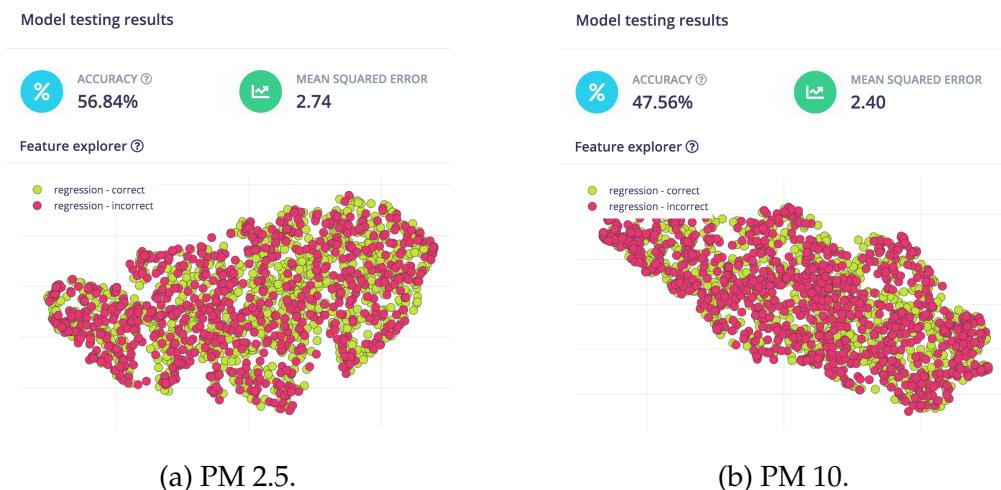


Figura 4.6: Output del test delle reti

Per mostrare il modello in funzione con tolleranza più alta, nelle figure osservabili di seguito è stato arbitrariamente impostato il valore di soglia a 30% del range di input delle etichette: 3.3 per il modello sui PM 2.5 e 3 per quello sui PM 10

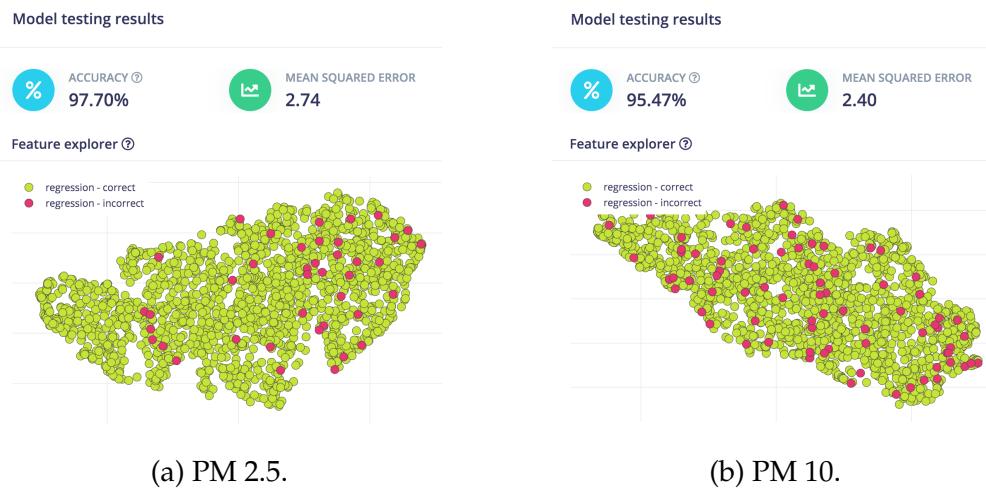


Figura 4.7: Output del test delle reti

4.4 ESPORTAZIONE DEL MODELLO

In seguito all’addestramento della rete neurale è possibile esportare una libreria Arduino che fornisce esempi che implementano il modello ottenuto utile a effettuare inferenze audio per la previsione di valori di particolato. Per poter esportare la libreria è possibile accedere all’opzione “Deployment” del menù del progetto e selezionare la “Arduino Library”.

Successivamente ci si può recare nella IDE⁵ di Arduino e installare la libreria scaricata selezionandola come file.zip; dalla stessa IDE di Arduino è inoltre possibile selezionare gli esempi forniti dalla stessa libreria e caricali su Arduino.

Considerato che Arduino Nano incorpora il microfono MP34DT05 è sufficiente aprire il proprio Monitor seriale e verificare il corretto funzionamento del Software in funzione.

⁵Integrated development environment.

5

Conclusione e progetti futuri

In conclusione, grazie a questo progetto di tesi, è stato possibile sfruttare dei sistemi di TinyML e di embedded machine learning con lo scopo di prevedere i valori di particolato basandosi sul rumore generato dal traffico veicolare. Da queste analisi sono emerse delle problematiche che, se da un lato hanno limitato la riuscita del progetto in questione, dall'altro hanno permesso di sviluppare nuove possibilità per quello che riguarda progetti futuri nel medesimo ambito. Un possibile sviluppo del progetto potrebbe prevedere l'integrazione di un dataset più ampio che permetta di raccogliere ed analizzare con più dettaglio tutta la complessità delle variabili che influiscono nel sistema di acquisizione dei dati e quindi sulla popolazione dello stesso dataset. Si auspica che tali migliorie possano essere attuate in un futuro prossimo e che i previsti risultati siano maggiormente soddisfacenti in termini di efficacia del modello per poter perseguire gli scopi inizialmente posti come obiettivi del progetto di tesi.

Bibliografia

- [1] «Arduino nano 33 ble sense Datasheet.» In: URL: <https://docs.arduino.cc/resources/datasheets/ABX00031-datasheet.pdf>.
- [2] «Datasheet microfono MP34DT05.» In: URL: https://content.arduino.cc/assets/Nano_BLE_Sense_mp34dt05-a.pdf.
- [3] Wikipedia. «Micro Electrical Mechanical Systems». In: 2022. URL: <https://it.wikipedia.org/wiki/MEMS>.
- [4] Ph.D Thomas Kite. «Understanding PDM Digital Audio». In: (2012). URL: https://users.ece.utexas.edu/~bevans/courses/rtdsp/lectures/10_Data_Conversion/AP_Understanding_PDM_Digital_Audio.pdf.
- [5] G. Magalini. «PDM_{serial}sender.ino». In: 2023. URL: https://github.com/Giorgio-Magalini/Indirect-techniques-for-environmental-monitoring-using-embedded-machine-learning/blob/main/PDM_Serial_Sender/PDM_Serial_Sender.ino.
- [6] Wikipedia. *Scattering — Wikipedia, L'enciclopedia libera.* [Online]. 2022. URL: <http://it.wikipedia.org/w/index.php?title=Scattering&oldid=127876890>.
- [7] Wikipedia. *Transistor-transistor logic — Wikipedia, L'enciclopedia libera.* [Online]. 2022. URL: http://it.wikipedia.org/w/index.php?title=Transistor-transistor_logic&oldid=127597923.
- [8] Honeywell International Inc. «HPMA Series». In: 2017. URL: <https://github.com/felixgalindo/HPMA115S0/blob/master/docs/honeywell-hpm32322550b-1099596.pdf>.
- [9] «Libreria HPMA115S0.» In: URL: <https://github.com/felixgalindo/HPMA115S0/tree/master>.

BIBLIOGRAFIA

- [10] G. Magalini. «MyHPMA_{Serial}sender.ino». In: 2023. URL: https://github.com/Giorgio-Magalini/Indirect-techniques-for-environmental-monitoring-using-embedded-machine-learning/blob/main/MyHPMA_Serial_Sender/MyHPMA_Serial_Sender.ino.
- [11] G. Magalini. «Serial_{ArduinoNANO}Serial_{ArduinoUNO}comV1.py». In: 2023. URL: https://github.com/Giorgio-Magalini/Indirect-techniques-for-environmental-monitoring-using-embedded-machine-learning/blob/main/Serial_ArduinoNANO_ArduinoUNO_com_V1.py.
- [12] Wikipedia. *Rete neurale artificiale* — Wikipedia, L'enciclopedia libera. [Online]. 2023. URL: http://it.wikipedia.org/w/index.php?title=Rete_neurale_artificiale&oldid=132087076.
- [13] G. Magalini. «UploadToEI.py». In: 2023. URL: <https://github.com/Giorgio-Magalini/Indirect-techniques-for-environmental-monitoring-using-embedded-machine-learning/blob/main/UploadToEI.py>.