Università degli Studi di Bergamo

Laboratorio di Sensori

Relazione progetto

A.A. 2023 – 2024

Studenti:

Aurora Zanenga

1054891

a.zanenga@studenti.unibg.it

Samuel Locatelli

1054674

s.locatelli30@studenti.unibg.it

Sommario

Introduzione

Per la realizzazione di questo progetto con Arduino nano 33 è stato scelto di creare un programma che date due voci sia in grado di riconoscere chi sta parlando.

L'idea di base è di realizzare un programma semplice di machine learning utilizzando Python che venga testato su un dataset di test relativamente piccolo ma di buona qualità.

Questo dataset conterrà dei valori calcolati da file audio, alcuni registrati dalla voce femminile e alcuni da una voce maschile.

Inizialmente si era pensato di realizzare il tutto utilizzando le nostre voci, questa idea iniziale si è rivelata più impegnativa dal punto di vista organizzativo per la semplice motivazione che sarebbe stato necessario essere sempre presenti entrambi per fare i test, oltretutto in uno spazio silenzioso al fine di mantenere alta la qualità delle registrazioni (fattore discriminante per la realizzazione del progetto a causa delle basse prestazioni dell'Arduino che richiedevano un file audio molto breve e a bassa frequenza di campionamento, senza queste due caratteristiche non sarebbe stato possibile completare l'elaborazione con successo).

Viste le problematiche precedentemente esposte si è deciso di realizzare il medesimo progetto sulle voci di due cantanti, scegliendo sempre una voce maschile e una voce femminile, il progetto e l'idea sono quindi rimasti i medesimi, è cambiato solamente il soggetto su cui effettuare i test.

Da Arduino IDE verrà quindi scritto un codice che dato l'input del sensore *microfono* integrato nell'Arduino verrà effettuato il riconoscimento tramite il modello per identificare chi sta cantando in quel momento (voce maschile o voce femminile).

L'output verrà comunicato tramite la stampa di 0 per la voce maschile e 1 per la voce femminile.

Lo scopo della realizzazione di questo progetto non è tanto l'utilità dello stesso quanto le applicazioni future che si potrebbero sviluppare a partire da un'applicazione di questo tipo che integra una parte di machine learning con i sensori dell'Arduino.

Strumentazione utilizzata

In questo paragrafo verrà descritta la strumentazione utilizzata per la realizzazione di questo progetto:

- Computer (sistema operativo windows 10) con installato Visual Studio Code per la stesura del codice Python (parte di machine learning) e Arduino IDE per l'integrazione con l'Arduino
- Arduino nano 33 BLE SENSE
- Cavo micro-USB (Type B)
- Dispositivo mobile esterno dal quale vengono riprodotte le due canzoni in modo alternato per effettuare i test

Predisposizione Ambiente e Programmazione Iniziale

Lo sviluppo di questo programma richiederà una parte di Python (realizzata su VS Code) e una parte realizzata tramite Arduino IDE:

Lato Python i punti dello sviluppo sono:

1. Stesura del codice
2. Creazione del dataset di Train (e dei sotto-dataset)
3. Creazione del dataset per la fase di Test
4. Train del modello
5. Test del modello
6. Porting (funzione che crea il file *my_model.h* necessario per interagire con l'Arduino)

Lato Arduino i punti di sviluppo sono:

1. Creazione del file main.cpp (file di default di Arduino IDE)
2. Inserimento nella cartella del file *my_model.h*
3. Tramite la funzione *predict* interagire con il file *my_model.h*

Progettazione

Il progetto viene realizzato in modo parallelo su due piattaforme: una parte su Arduino e una parte su VS Code (Python).

Da VS Code (lato Python) per la parte di test verranno inseriti due file in formato csv che conterranno dei campioni, uno contenente i campioni ottenuti dalla voce femminile e uno contenente i campioni ottenuti dalla voce maschile.

Questi campioni sono di fatto la trasformata di Fourier della porzione di audio, i file sono strutturati come segue: ciascuna riga conterrà la trasformata di Fourier dell'ultimo chunk di audio (la trasformata di Fourier viene fatta sull'ultimo secondo, il risultato della trasformata viene normalizzato in un range tra 0 e 1, ciascuna riga del file csv corrisponde all'elaborazione di un chunk di audio (un secondo)).

Questi dati (che vengono presi dal print del serial monitor lato Arduino) sono necessari per trainare il modello, verranno successivamente divisi in una parte di train e in una parte di test.

Una volta trainato e testato il modello, questo viene salvato in un file chiamato "**my_model.h**", questo file contiene i pesi ricavati durante il train del modello.

Il file *my_model.h* verrà inserito in una cartella assieme al codice Arduino, in modo tale che possa essere incluso in quest'ultimo con facilità.

Lo script di Arduino invece è stato strutturato in modo tale da avere due modalità, una è la modalità ***prediction*** e una è la modalità ***raccolta dati***.

Inizialmente, nella parte di loop, viene creato un vettore di buffer che mantiene il last 1 second di audio prelevato tramite il microfono dell'Arduino (si tratta di una coda che mantiene sempre l'ultimo secondo audio registrato).

Nella modalità "**prediction**" la serial monitor stampa la predizione, quindi 0 o 1 in base all'output del modello *my_model.h* su chi sta cantando.

Nella modalità "**raccolta dati**" la serial monitor stamperà riga per riga i campioni (trasformata di Fourier riga per riga del chunk di audio processato in quell'istante) che poi verranno copiati e incollati manualmente nel file csv (come riportato all'inizio del paragrafo).

Gestione dei file audio per il modello di machine learning

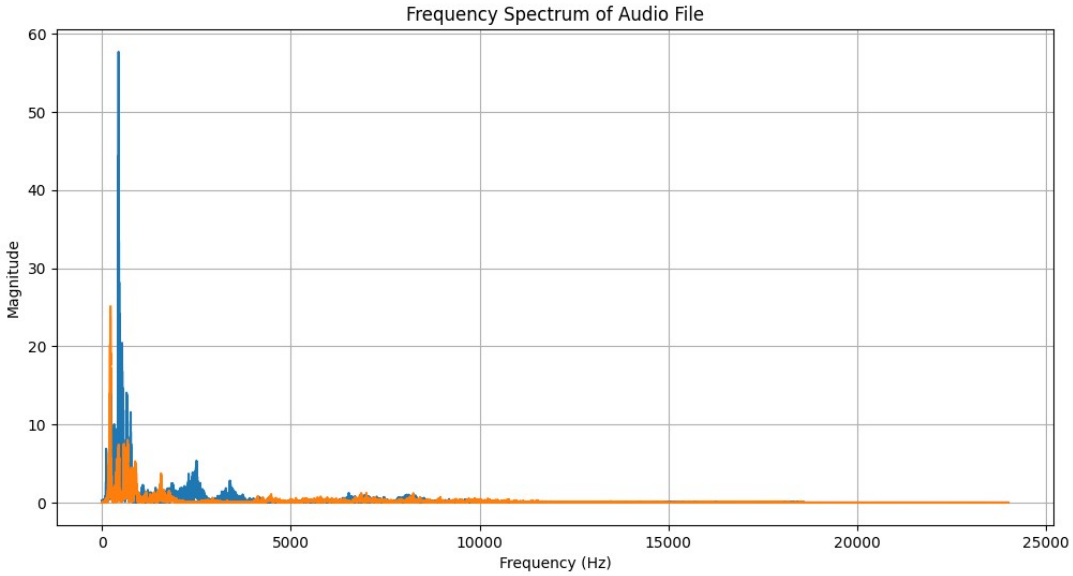
L'idea è di partire dal segnale catturato dal microfono che sarà nel dominio del tempo, la frequenza di campionamento è circa 8KHz, significa che in un secondo vengono fatte circa 8000 acquisizioni del segnale.

È possibile creare un modello che dato questo segnale (che sarà un vettore di 8000 campioni) il modello impari che i primi 8000 campioni forniti sono una voce femminile, e che dati altri 8000 campioni, far sì che impari che quei campioni devono essere associati ad una voce maschile.

Intuizione: non serve un dataset particolarmente ampio se la qualità del dato è buona, il modello viene trainato su registrazioni fatte in condizioni ottime (stanza silenziosa, buona qualità dell'audio, voce continua (essenziale visto che il buffer è di un secondo)).

Ogni segnale può essere visto come una somma di svariate sinusoidi, viene fatta la supposizione che una voce maschile abbia tonalità più basse (armoniche ad una frequenza minore), mentre una voce femminile abbia tonalità più acute (frequenza più alta).

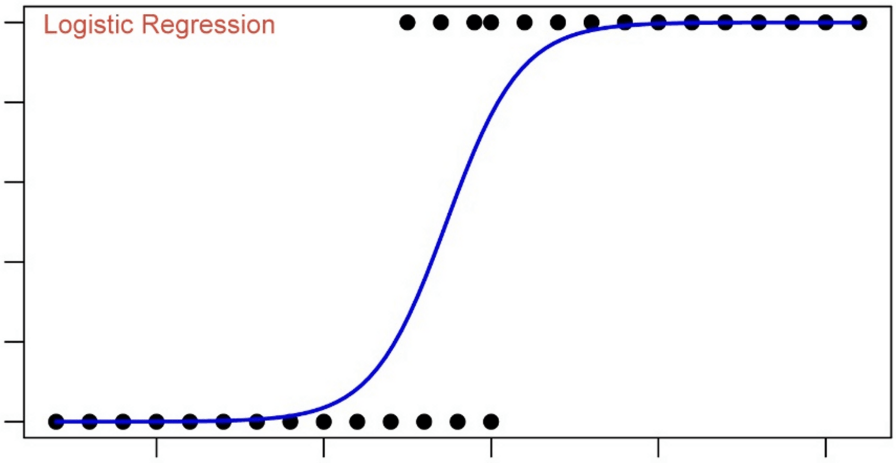
Nel momento in cui vengono trasformate le voci maschili e femminili da segnale nel dominio del tempo a segnale nello spettro delle frequenze (si passa dal dominio del tempo al dominio delle frequenze), plottando il grafico si dovrebbe vedere la differenza tra le distribuzioni delle due voci.



Modello di Machine Learning

Come modello viene utilizzata la **Logistic Regression**, è un modello molto semplice, di fatto si tratta di una retta con una Sigmoide alla fine.

Ogni valore di input viene associato e moltiplicato ad un coefficiente di quella retta.



Integrazione modello di Machine Learning con Arduino

Una volta completata la parte di machine learning e ottenuto il modello sarà necessario integrarla con la parte di Arduino:

Come prima cosa dev'essere effettuata la lettura del microfono, esiste una libreria in grado di inserire ciò che viene registrato all'interno di un vettore Buffer[], tutto quello che rileva il sensore microfono viene inserito nel vettore.

A questo punto si arriva a quella che è stata la parte più impegnativa: il training è stato fatto sul computer, il modello però è da portare sul microcontrollore.

Anche per questo è stata utilizzata una libreria che permetteva di esportare il modello creando un file *.h* in C++ contenente il metodo già implementato della funzione ***predict()***

Questo file *.h* generato dalla libreria viene messo su Arduino, una volta importato nel file *main.cpp* sarà possibile da lì chiamare la funzione *predict*.

Gestione lunghezza file audio

Problema: Lato Arduino idealmente servirebbe un vettore buffer[] (il vettore buffer[] conteneva gli ultimi *n* secondi registrati dal microfono, su questo veniva effettuato il *predict* per identificare il tipo di voce) contenente file audio abbastanza lunghi (almeno 2/3 secondi) a 16KHz al fine di identificare al meglio se si tratta di una voce femminile o maschile, l'Arduino però non è in grado di processare con successo segnali audio con queste caratteristiche, il massimo che è stato possibile ottenere è utilizzare **acquisizioni audio da 1 secondo a 8KHz**, si noti che si tratta di meno della metà della lunghezza dell'audio ottimale a metà della frequenza di campionamento.

Si è reso quindi necessario ridurre il modello, per diminuire il modello bisogna rimpicciolire l'input, come spiegato precedentemente negli audio è possibile giocare su lunghezza e frequenza di campionamento.

La scelta è stata quella di abbassare la frequenza di campionamento a 8KHz (dai 16000KHz originariamente utilizzati, che sono quelli supportati dal sensore microfono dell'Arduino).

Frequenza 8KHz * 1 secondo = 8000 campioni, questo sarà l'input della logistic regression dopo aver effettuato la trasformata di Fourier.

Commento codice Python- parte machine learning

```
import pandas
import numpy as np
import sklearn
from micromlgen import port
import matplotlib.pyplot as plt
```

All'inizio del file vengono importate le librerie necessarie per gestire i file audio, eseguire la trasformata di Fourier, fare grafici e utilizzare *micromlgen* per generare codice adatto ai microcontrollori (precedentemente importate tramite comando *py -m pip install 'nomeLibreria'*).

```
pd1 = pandas.read_csv('fem.csv')
pd2 = pandas.read_csv('masc.csv')

#convert nan to 0
pd1 = pd1.fillna(0)
pd2 = pd2.fillna(0)

#convert to numpy array
np1 = pd1.to_numpy()
np2 = pd2.to_numpy()
```

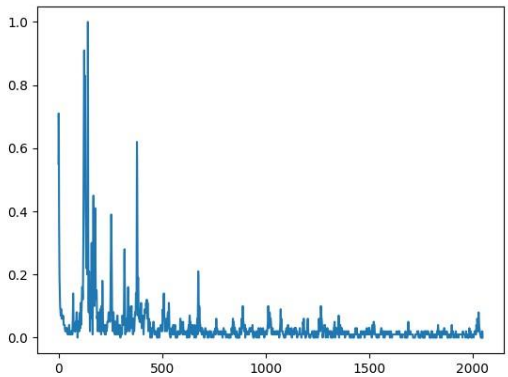
In queste righe di codice vengono importati i file csv contenenti i campioni necessari per il train del modello (ricavati da Arduino, spiegazione completa nel paragrafo successivo).

Vengono sostituiti con il valore 0 eventuali valori NaN, dopo di chè vengono convertiti in array NumPy.

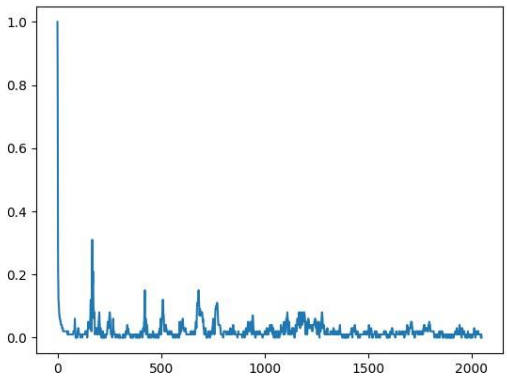
```
plt.plot(np1[0])
plt.savefig('test_np1.png')
```

Viene plottato il grafico dell'array np1 (contenente i campioni della voce femminile), questo è stato necessario per poter vedere effettivamente la diversità delle due voci:

Voce femminile:



Voce maschile:



```
train = np.concatenate((np1, np2), axis=0)
labels = np.concatenate((np.zeros(len(np1)), np.ones(len(np2))), axis=0)
```
Qui vengono concatenati gli array np1 e np2 per formare il dataset di addestramento. Viene creato un array “lables” per etichettare i dati in modo tale che il modello sappia su quali dati si sta allenando (voce maschile o voce femminile).

```
from sklearn.model_selection import train_test_split
train, test, labels, labels_test = train_test_split(train, labels, test_size=0.2,
random_state=42)
```
Viene utilizzata la funzione “train_test_split” per dividere i dati in una parte di train e una parte di test (80% di train e 20% di test).

```
from sklearn.linear_model import LogisticRegression
model = LogisticRegression( max_iter=1000)
model.fit(train, labels)
```
Crea e addestra un modello di logistic regression con i dati di addestramento.

```
import pickle
pickle.dump(model, open('model.pkl', 'wb'))
```
Salva il modello in un file chiamato “model.plk”.

```
model = pickle.load(open('model.pkl', 'rb'))

#test with dataset train
print(model.predict(train))

print("test samples")
print(model.predict(test))
print("labels test")
print(labels_test)
```
Ricarica il modello e lo testa, è interessante notare nell’immagine successiva che nel momento in cui combaciano i valori nel *test sample* con quelli nel *labels test* significa che il modello sta funzionando bene (nell’esempio riportato il modello ha sbagliato solo una volta):

```
test samples
[1. 1. 0. 1. 0. 1. 0. 1. 0. 1. 1. 0. 1.]
labels test
[1. 1. 0. 1. 0. 1. 0. 0. 0. 1. 1. 0. 1.]
finished
```

```
ported = port(model)

#save ported to txt file
with open('my_model.h', 'w') as f:
    f.write(ported)
```

Viene convertito il modello in un formato tale da poter interagire con l'Arduino, successivamente viene salvato in un file chiamato "*my_model.h*".

Questo file *my_model.h* verrà importato lato Arduino.

Commento codice Arduino

```
const int SAMPLES = 8192;
```

Per prima cosa viene definita la dimensione del buffer audio all'interno dell'Arduino, dev'essere una potenza di 2.

```
const int downsample_factor = 2;
```

Permette di ridurre i dati che poi vengono inseriti nel modello e quindi di velocizzare l'inferenza, essenzialmente prende il buffer di 8192 elementi (che rappresentano un secondo di audio) e fa il downsample (prende il primo elemento, salta il secondo, prende il terzo, salta il quarto… sostanzialmente ne prende uno sì ed uno no), in questo modo la dimensione di ciò che viene inserito nel modello si dimezza.

```
float features[SAMPLES/downsample_factor/2 + 1];
```

Questa riga di codice rappresenta effettivamente ciò che viene inserito all'interno del modello: i sample che sono stati presi, diviso per il downsample factor, diviso 2 (questo perché nel momento in cui viene fatta la trasformata di Fourier si ottiene uno spettro di potenza per ciascuna frequenza, quando viene fatta nel campo discreto si ottiene uno spettro che è assolutamente simmetrico da metà in poi, la prima metà è esattamente uguale alla seconda, questo significa che ha esattamente lo stesso contenuto informativo della prima metà, non ha quindi senso mantenerle entrambe).

```
void loop() {
  if (samplesRead) {

    for (int i = 0, j = 0; i < SAMPLES /downsample_factor; i++,j += 
downsample_factor) {
      vReal[i] = sampleBuffer[j];
      vImag[i] = 0;
    }

    FFT.Windowing(vReal, SAMPLES/downsample_factor, FFT_WIN_TYP_HAMMING, 
FFT_FORWARD);
    FFT.Compute(vReal, vImag, SAMPLES/downsample_factor , FFT_FORWARD);
    FFT.ComplexToMagnitude(vReal, vImag, SAMPLES/downsample_factor );
    minMaxNormalize(vReal, SAMPLES / downsample_factor + 1);

    for (int i = 0; i < SAMPLES/downsample_factor /2; i++) {
      features[i] = vReal[i];
      if (1 == 2){
        Serial.print(vReal[i]);
        Serial.print(',');
      }

    }
```

```
    Serial.println();

    Serial.println(model.predict(features));

    samplesRead = 0;
  }
}
```

All'interno del loop, per prima cosa viene copiato dentro ***vReal*** tutto il buffer che è stato prelevato all'istante corrente.

In parallelo a *vReal* viene realizzato un vettore ***vImg*** che serve per la parte immaginaria della trasformata di Fourier (questo viene inizializzato tutto a 0).

Vengono chiamate le tre funzioni che fanno il calcolo della trasformata di Fourier sul vettore dei sample (campioni audio) che gli è stato passato, dopo di che la trasformata viene salvata dentro *vReal*, e infine viene normalizzato il vettore tra 0 e 1.

Viene salvato il risultato normalizzato dentro "features", questo sarà il vettore che verrà poi effettivamente utilizzato all'interno del modello.

Quando ci si trova in modalità "*acquisizione dati*" (quindi se nel codice viene posto 1=1 per ottenere la condizione vera ed entrare in quella porzione di codice) allora vengono stampati i campioni (che poi diventano l'input del modello, file csv), questi vengono copiati e incollati manualmente nello script Python, tramite essi viene fatto il train e da qui torna indietro il modello.

La print rappresenta la trasformata di Fourier dell'ultimo secondo di audio.

Se invece ci si trova in modalità "prediction" allora stamperà in output 0 se la voce è maschile, 1 se la voce è femminile.