

---

# AUDIO RECOGNIZER

---

*Sistema per il riconoscimento delle emozioni dalla voce*



DIPARTIMENTO DI INFORMATICA - SISTEMI AD AGENTI 21/22

**TEAM:**

- ANTONIO PAPEO 728625
- FRANCESCO SASSO 715742
- CHRISTIAN RIEFOLO 618687
- RUGGIERO ZAGARIA 727707

## INDICE

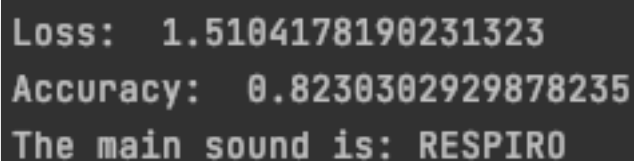
<b><u>1.</u></b>	<b><u>INTRODUZIONE .....</u></b>	<b><u>2</u></b>
<b><u>2.</u></b>	<b><u>IMPLEMENTAZIONE DEL MODELLO .....</u></b>	<b><u>2</u></b>
<b><u>3.</u></b>	<b><u>DATASET .....</u></b>	<b><u>8</u></b>
<b><u>4.</u></b>	<b><u>ORGANIZZAZIONE PROGETTO .....</u></b>	<b><u>11</u></b>
<b><u>5.</u></b>	<b><u>CONCLUSIONI FINALI .....</u></b>	<b><u>11</u></b>

## 1. Introduzione

AudioRecognizer è un sistema scritto in Python in grado riconoscere le emozioni espresse da un essere umano tramite la voce, utilizzando un modello addestrato per il riconoscimento audio.

Le emozioni rilevate dal sistema sono:

<b>Dolore</b>
<b>Sbadiglio</b>
<b>Risata</b>
<b>Respirare</b>
<b>Russare</b>
<b>Pianto di un Bambino</b>
<b>Starnuto</b>
<b>Urla</b>
<b>Tosse</b>

A terminal window with a dark background and light gray text. It displays three lines of output: 'Loss: 1.5104178190231323', 'Accuracy: 0.8230302929878235', and 'The main sound is: RESPIRO'.

```
Loss: 1.5104178190231323
Accuracy: 0.8230302929878235
The main sound is: RESPIRO
```

Figura 1: Esempio risultato riconoscimento audio

## 2. Implementazione del modello

Il modello utilizzato all'interno dell'applicativo è scritto in Python e, grazie ad un dataset appositamente suddiviso in *Training*, *Test* e *Validation*, permette di sfruttare il suo addestramento per il riconoscimento di determinate emozioni fornite in input.

Le librerie implementate sono:

- **Pandas**: utilizzata per lavorare con il csv contenente le informazioni del dataset e scaricabile con il comando `<< pip install pandas >>`.
- **Tensorflow**: utilizzata per effettuare operazioni nel modello come l'esecuzione dell'inferenza, scaricabile con `<< pip install tensorflow >>`.

- **Tensorflow\_hub**: utilizzata per caricare il modello ottenuto dal seguente link <https://tfhub.dev/google/yamnet/1>, scaricabile con `<< pip install tensorflow_hub >>`.
- **Tensorflow\_io**: utilizzata per impostare il giusto sample\_rate nell'audio caricato, scaricabile con `<< pip install tensorflow_io >>`.

```
@tf.function
def load_wav_16k_mono(filename):
    """ Load a WAV file, convert it to a float tensor, resample to 16 kHz single-channel audio. """
    file_contents = tf.io.read_file(filename)
    wav, sample_rate = tf.audio.decode_wav(
        file_contents,
        desired_channels=1)
    wav = tf.squeeze(wav, axis=-1)
    sample_rate = tf.cast(sample_rate, dtype=tf.int64)
    wav = tfio.audio.resample(wav, rate_in=sample_rate, rate_out=16000)
    return wav

testing_wav_data = load_wav_16k_mono(testing_wav_file_name)

_ = plt.plot(testing_wav_data)

display.Audio(testing_wav_data, rate=16000)
```

Figura 2: Funzione caricamento audio

La funzione, visionata nella figura 2, verrà utilizzata dopo aver definito il modello ed aver inserito l'audio di cui si vuole effettuare il riconoscimento. Essa permette di caricare il file appena immesso e di modificarne il sample rate, **ovvero il numero di cicli in cui viene catturato l'audio (campionato) in un secondo**, in modo da poterlo analizzare.

```
# Caricare la mappatura della classe

class_map_path = yamnet_model.class_map_path().numpy().decode('utf-8')
class_names = list(pd.read_csv(class_map_path)['display_name'])

# Esegue l'inferenza

scores, embeddings, spectrogram = yamnet_model(testing_wav_data)
class_scores = tf.reduce_mean(scores, axis=0)
top_class = tf.argmax(class_scores)
inferred_class = class_names[top_class]
```

Figura 3: Mappatura della classe ed Esecuzione Inferenza

Nella figura 3 mostriamo come viene mappata la classe audio secondo una decodifica UTF-8 e di come viene effettuata l'inferenza sull'audio inserito in input. Successivamente abbiamo inserito nel sistema le classi presenti nel dataset ed abbiamo associato a ciascuna di essa un id, in modo tale da poter filtrare il dataset per categoria sfruttando pandas.

E' fondamentale che si utilizzino audio con le giuste caratteristiche, ovvero con canale mono a 16 kHz, infatti al processamento di ogni file viene anteposta una fase di conversione.

Le categorie inserite corrispondono alla suddivisione esistente nel dataset:

CATEGORIA	ID CATEGORIA
DOLORE	0
PIANTO_BAMBINO	1
RESPIRO	2
RISATA	3
RUSSARE	4
SBADIGLIO	5
STARNUTO	6
TOSSE	7
URLA	8

Dopo aver definito le classi procediamo con l'incorporamento dei dati:

```
filenames = filtered_pd['filename']
targets = filtered_pd['target']
folds = filtered_pd['fold']

main_ds = tf.data.Dataset.from_tensor_slices((filenames, targets, folds))
main_ds.element_spec

def load_wav_for_map(filename, label, fold):
    return load_wav_16k_mono(filename), label, fold

main_ds = main_ds.map(load_wav_for_map)
main_ds.element_spec
```

Figura 4: Incorporamento dati

Successivamente procediamo con l'estrazione dei file audio presenti all'interno del dataset utilizzando il modello di estrazione dati presente in **Yamnet**, come possiamo vedere dalla figura 5.

```
def extract_embedding(wav_data, label, fold):
    ''' run YAMNet to extract embedding from the wav data '''
    scores, embeddings, spectrogram = yamnet_model(wav_data)
    num_embeddings = tf.shape(embeddings)[0]
    return (embeddings,
            tf.repeat(label, num_embeddings),
            tf.repeat(fold, num_embeddings))

# Estrazione

main_ds = main_ds.map(extract_embedding).unbatch()
main_ds.element_spec
```

Figura 5: Estrazione dati dal dataset

Come possiamo vedere dalla figura 6, abbiamo suddiviso i dati in Training set, Test set e Validation set, suddividendoli come:

<b>TRAINING SET</b>	<b>70%</b>
<b>TEST SET</b>	<b>20%</b>
<b>VALIDATION SET</b>	<b>10%</b>

La suddivisione dei dati, come vedremo successivamente nella Figura 9 , è stata rappresentata sia nel csv che nel codice, come possiamo notare nella figura 6, assegnando un id a ciascuna fold:

<b>TRAINING</b>	<b>1</b>
<b>TEST</b>	<b>2</b>
<b>VALIDATION</b>	<b>3</b>

```
cached_ds = main_ds.cache()
train_ds = cached_ds.filter(lambda embedding, label, fold: fold == 1)
val_ds = cached_ds.filter(lambda embedding, label, fold: fold == 2)
test_ds = cached_ds.filter(lambda embedding, label, fold: fold == 3)
```

Figura 6: Divisione dei dati nei relativi: Training, Test e Validation

```

my_model = tf.keras.Sequential([
    tf.keras.layers.Input(shape=(1024), dtype=tf.float32,
                             name='input_embedding'),
    tf.keras.layers.Dense(512, activation='relu'),
    tf.keras.layers.Dense(len(my_classes))
], name='my_model')

my_model.summary()

my_model.compile(loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
                 optimizer="adam",
                 metrics=['accuracy'])

callback = tf.keras.callbacks.EarlyStopping(monitor='loss',
                                             patience=3,
                                             restore_best_weights=True)

history = my_model.fit(train_ds,
                       epochs=20,
                       validation_data=val_ds,
                       callbacks=callback)

loss, accuracy = my_model.evaluate(test_ds)

print("Loss: ", loss)
print("Accuracy: ", accuracy)

# Prova modello

scores, embeddings, spectrogram = yamnet_model(testing_wav_data)
result = my_model(embeddings).numpy()

inferred_class = my_classes[result.mean(axis=0).argmax()]
print(f'The main sound is: {inferred_class}')

```

Figura 7: Creazione, Addestramento ed esecuzione del modello

Infine, come possiamo vedere nella figura 7, abbiamo creato il modello sfruttando la libreria di Tensorflow e chiamandolo 'my\_model' ed abbiamo verificato quanto stesse migliorando nel tempo il suo apprendimento, dopo almeno 5 avvii, l'accuratezza del sistema.

**MEDIA ACCURATEZZA → 0,72**

Comprendiamo come l'accuratezza non sia elevata e questo possa comportare, in alcuni casi, una predizione errata ma purtroppo non sono stati trovati ulteriori audio in rete utili per l'addestramento del modello.



Infine come risultato finale si ottiene la classe rispettiva all'audio inserito in input, permettendo così di comprendere la tipologia di emozione emessa dall'essere umano.

### **3. Dataset**

Tutti gli audio utilizzati per le fasi di training, validation e test derivano da ulteriori dataset presenti sulla rete. In particolar modo sono stati fondamentali i seguenti dataset:

1. ESC-50
2. Mixkit
3. Soundbank
4. Coswara
5. Mendeley
6. VIVAE
7. Non-Vocalization-Dataset

Il dataset, come possiamo vedere nella figura 8 e 9, è suddiviso in cartelle in base alla categoria e a sua volta suddiviso internamente in base al relativo utilizzo:

- **Training**
- **Validation**
- **Test**



Figura 8:Suddivisione cartelle Dataset

filename	fold	target	category
./DATASET/TRAINING/RESPIRO/breathing-shallow(33).wav	1	2	RESPIRO
./DATASET/TRAINING/STARNUTO/086N_13_6_1_25_0_0_0.wav	1	6	STARNUTO
./DATASET/TRAINING/RESPIRO/breathing-shallow(23).wav	1	2	RESPIRO
./DATASET/TRAINING/RESPIRO/breathing-shallow(68).wav	1	2	RESPIRO
./DATASET/TRAINING/URLA/IPBF_15_8_0_23_0_0_0.wav	1	8	URLA
./DATASET/TRAINING/RESPIRO/breathing-shallow(30).wav	1	2	RESPIRO
./DATASET/TRAINING/SBADIGLIO/91D1_5_13_0_25_0_0_0.wav	1	5	SBADIGLIO
./DATASET/TRAINING/URLA/GZAE_15_9_0_28_0_0_0.wav	1	8	URLA
./DATASET/TRAINING/URLA/OZDT_15_15_0_31_0_0_0.wav	1	8	URLA
./DATASET/TRAINING/SBADIGLIO/HEVI_5_13_0_39_0_0_0.wav	1	5	SBADIGLIO
./DATASET/TRAINING/RESPIRO/breathing-shallow(61).wav	1	2	RESPIRO
./DATASET/TRAINING/RESPIRO/breathing-shallow(2).wav	1	2	RESPIRO
./DATASET/TRAINING/TOSSE/BYGU_4_1_0_28_0_0_0.wav	1	7	TOSSE
./DATASET/TRAINING/SBADIGLIO/5VYV_5_7_0_19_0_0_0.wav	1	5	SBADIGLIO
./DATASET/TRAINING/PIANTO_BAMBINO/uncom_natan11.wav	1	1	PIANTO_BAMBINO
./DATASET/TRAINING/RESPIRO/breathing-shallow(45).wav	1	2	RESPIRO
./DATASET/TRAINING/PIANTO_BAMBINO/LaparAUD-20150509-WA0001.wav	1	1	PIANTO_BAMBINO
./DATASET/TRAINING/STARNUTO/LHCB_13_14_1_44_0_0_0.wav	1	6	STARNUTO
./DATASET/TRAINING/RISATA/82GZ_12_7_0_36_0_0_0.wav	1	3	RISATA
./DATASET/TRAINING/TOSSE/cough-heavy(14).wav	1	7	TOSSE
./DATASET/TRAINING/TOSSE/cough-heavy(25).wav	1	7	TOSSE
./DATASET/TRAINING/TOSSE/cough-heavy(7).wav	1	7	TOSSE
./DATASET/TRAINING/RESPIRO/breathing-shallow(46).wav	1	2	RESPIRO
./DATASET/TRAINING/TOSSE/FBVG_4_14_0_16_0_0_0.wav	1	7	TOSSE
./DATASET/TRAINING/SBADIGLIO/9U2E_5_9_0_15_0_0_0.wav	1	5	SBADIGLIO
./DATASET/TRAINING/PIANTO_BAMBINO/hung_yudi_cari_puting_15.wav	1	1	PIANTO_BAMBINO
./DATASET/TRAINING/URLA/GZAE_15_6_0_28_0_0_0.wav	1	8	URLA
./DATASET/TRAINING/TOSSE/8QH3_4_15_0_20_0_0_0.wav	1	7	TOSSE
./DATASET/TRAINING/TOSSE/LFB3_4_13_0_25_0_0_0.wav	1	7	TOSSE
./DATASET/TRAINING/RISATA/MF7V_12_8_0_28_0_0_0.wav	1	3	RISATA
./DATASET/TRAINING/TOSSE/cough-heavy(22).wav	1	7	TOSSE

Figura 9:Suddivisione csv Dataset

## **4. Organizzazione Progetto**

L'organizzazione del progetto è avvenuta tramite l'applicativo Microsoft Teams, il quale ci ha permesso di lavorare in gruppo per l'intera fase di progetto nonostante le distanze di abitazioni.

Per la condivisione del progetto abbiamo sfruttato github il quale ci ha permesso di collaborare in maniera omogenea al progetto.

Per la comunicazione giornaliera su idee di progetto e sull'avanzamento delle varie fasi è stata utilizzata la piattaforma Whatsapp mobile.

Come IDE per lo sviluppo del codice abbiamo scelto IntelliJ Idea vista la sua ottima implementazione con github il che ci permetteva un lavoro più semplice ed efficiente.

## **5. Conclusioni Finali**

Ringraziamo per l'attenzione tenutasi nei confronti di AudioRecognizer da parte di tutto il Team.

Per poter scaricare il progetto da github è possibile accedere tramite il seguente link:

<https://github.com/AntonioPapeo6/ProgSysAg>