



# Instrument Recognition con CNN

Riconoscimento di strumenti musicali utilizzando una rete neurale convoluzionale

# Introduzione

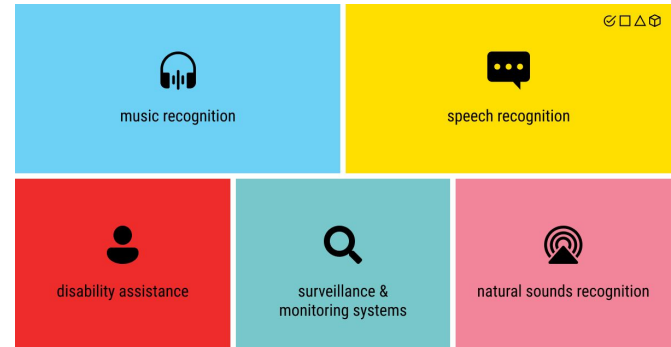
---



# Sound Recognition

La sound recognition è il processo mediante il quale un suono viene elaborato attraverso un computer per essere catalogato.

Usato soprattutto per **music recognition** e **riconoscimento vocale**.





# Music Recognition

La Music information retrieval (MIR) studia come estrapolare informazioni dalla musica.

Usata per:

- categorizzare musica
- riconoscere canzoni
- creare nuove canzoni

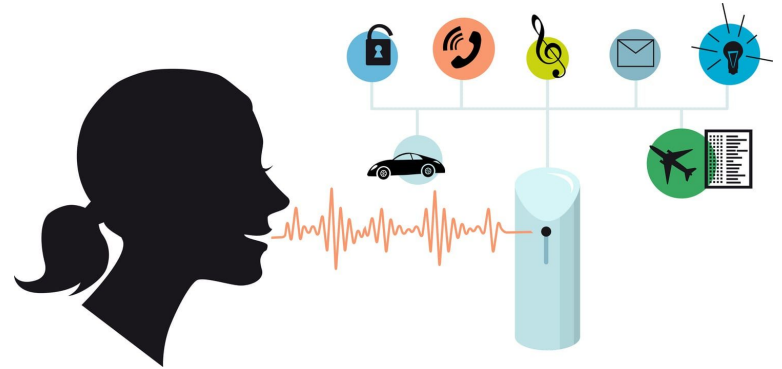
Una delle applicazioni più famose che fa music recognition è Shazam



# Riconoscimento vocale

Alcuni campi in cui il riconoscimento vocale è utilizzato:

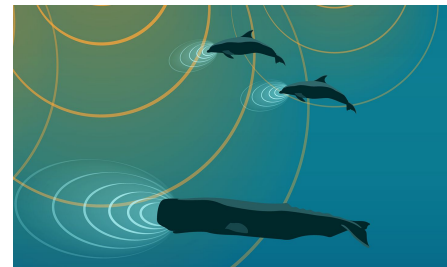
- assistenti vocali
- sicurezza con autenticazione vocale
- comandare veicoli



# Altre applicazioni della Sound Recognition

Altre applicazioni in cui è utile la sound recognition:

- sicurezza
- assistenza a persone con disabilità
- riconoscimento di specie animali

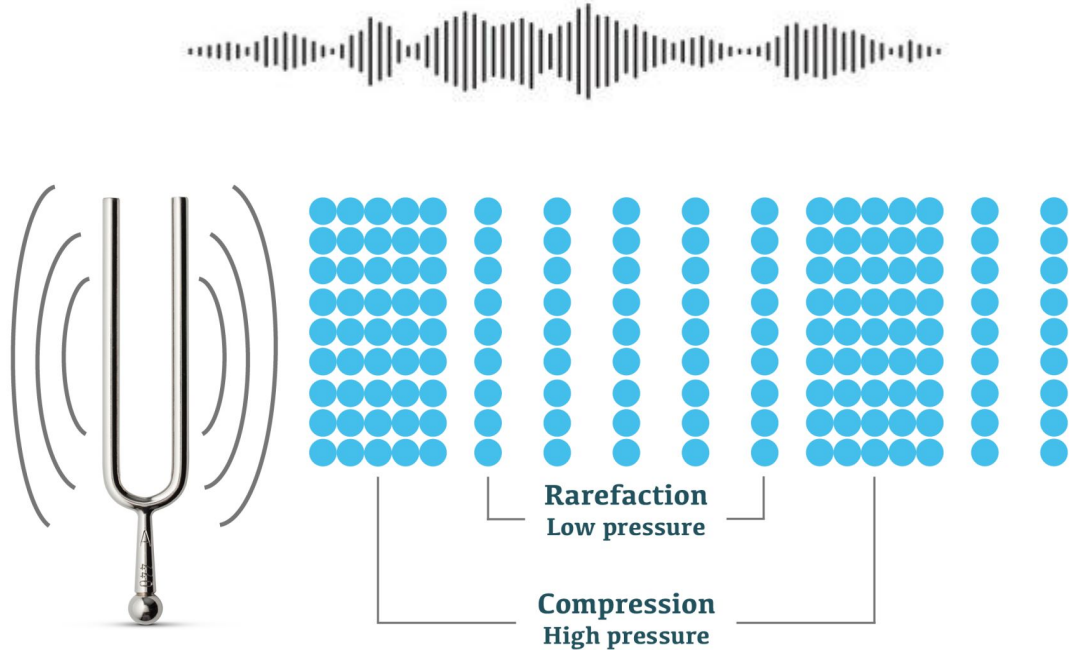


# I segnali audio

---

# Cos'è il suono?

Il suono è il risultato di un'onda meccanica creata da un oggetto che vibra. A muoversi nel mezzo sono le zone di alta e bassa pressione.

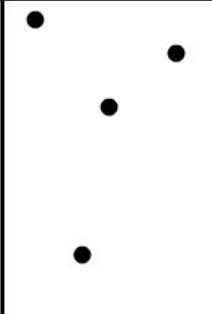
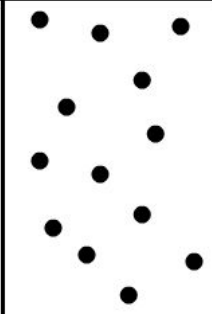
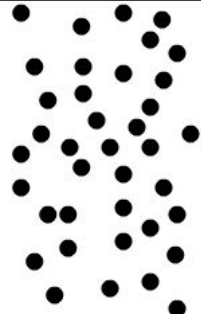




# Velocità del suono

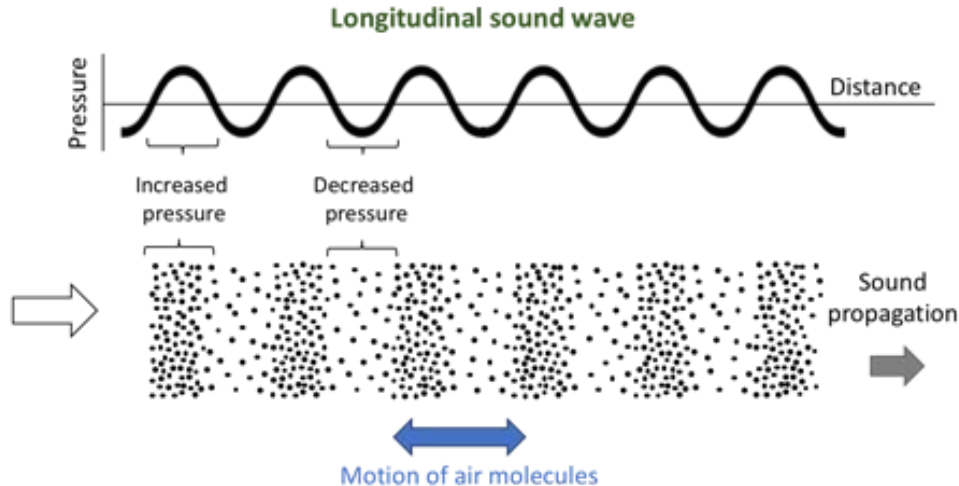
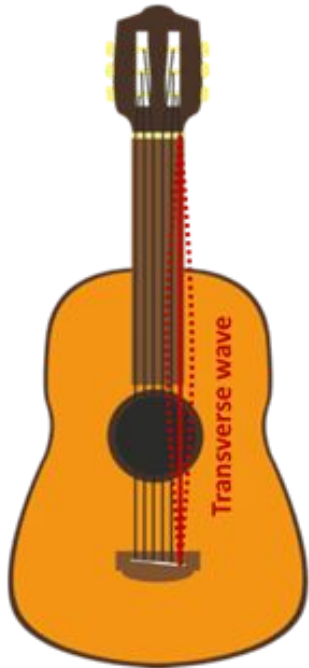
La velocità con cui il suono si propaga varia in base alla densità del mezzo.

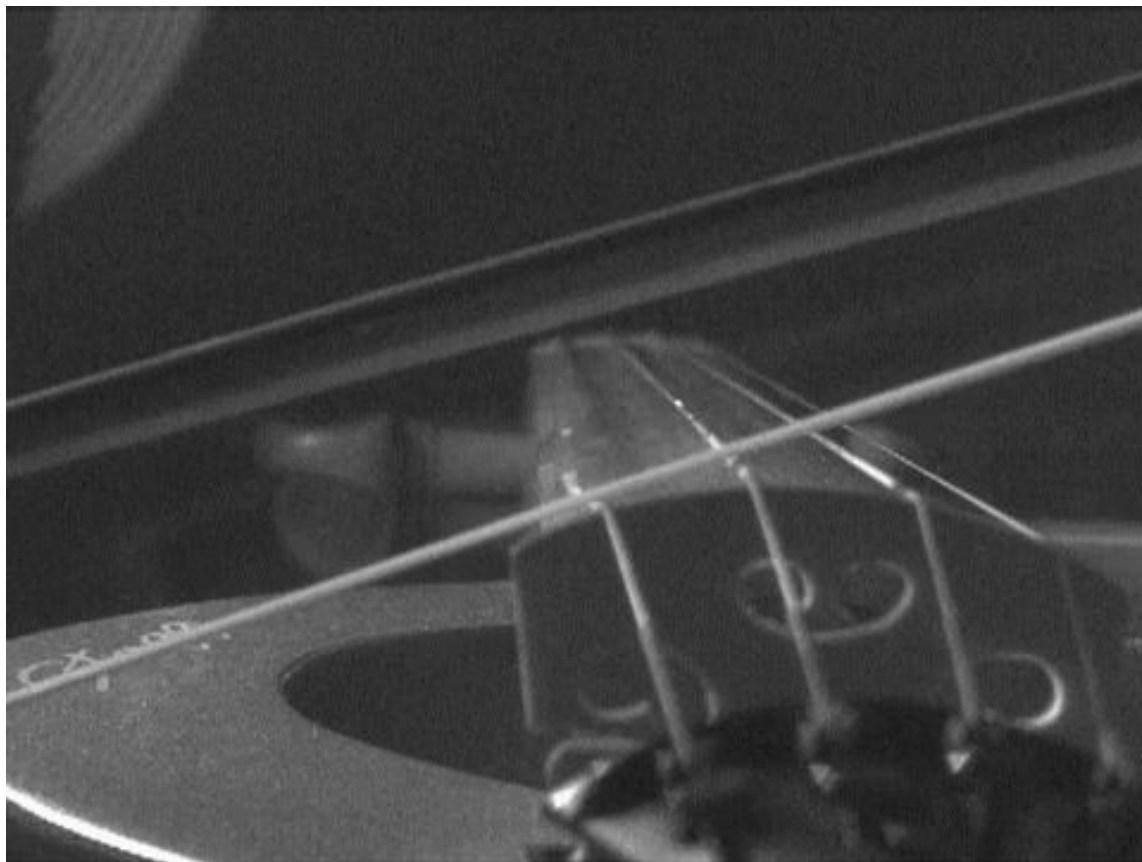
**aria:** 340 m/s  
**acqua:** 1480 m/s  
**ferro:** 5130 m/s

No Motion	Slowest Least Elastic	→	Fastest Most Elastic
Vacuum	Gas	Liquid	Solid
			

# Suono emesso da una chitarra

In una chitarra le corde, vibrando, creano zone di alta e bassa pressione alla frequenza con cui vibrano. Questo genera un'onda sonora.





Corda di violino che vibra a rallentatore

# Frequenze dell'audio



Un uomo sente nel range da 20 hz a 20000 hz. Altri animali possono udire frequenze differenti come ultrasuoni e infrasuoni.



# Strumenti musicali

Le note musicali si distinguono in base alle frequenze.

Range di frequenze fondamentali di alcuni strumenti:

- **Pianoforte:** da 27 a 4100 hz
- **Chitarra:** da 80 a 630 hz
- **Xilofono:** da 700 a 3500 hz
- **Flauto:** da 250 a 2500 hz
- **Violino:** da 200 a 3500 hz
- **Organo:** da 20 a 7000 hz



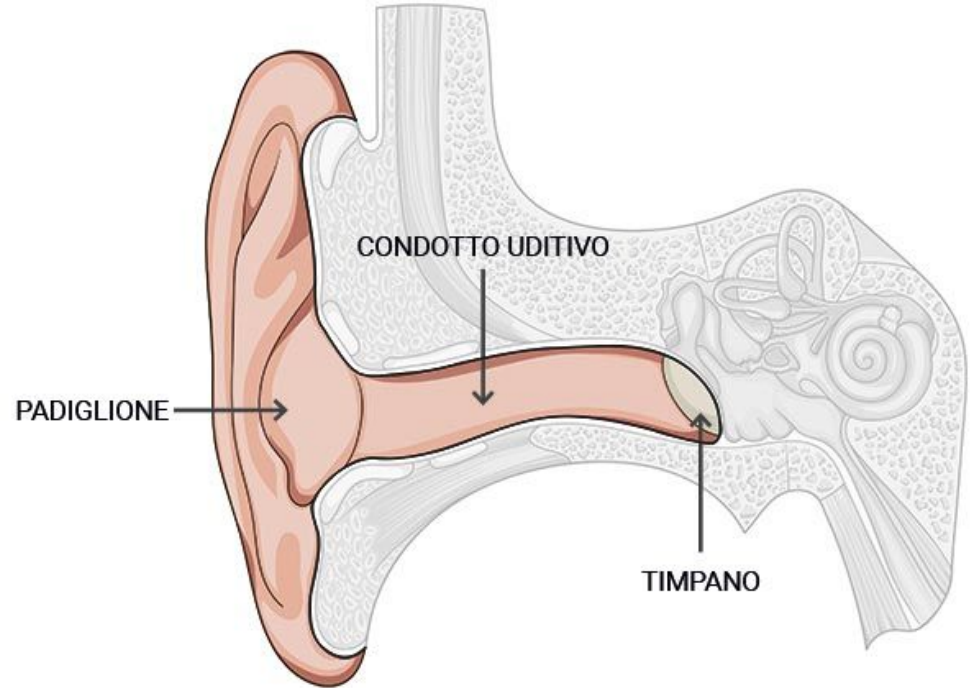
# Apparato uditivo

---

## Orecchio esterno

La vibrazione arriva al nostro orecchio.

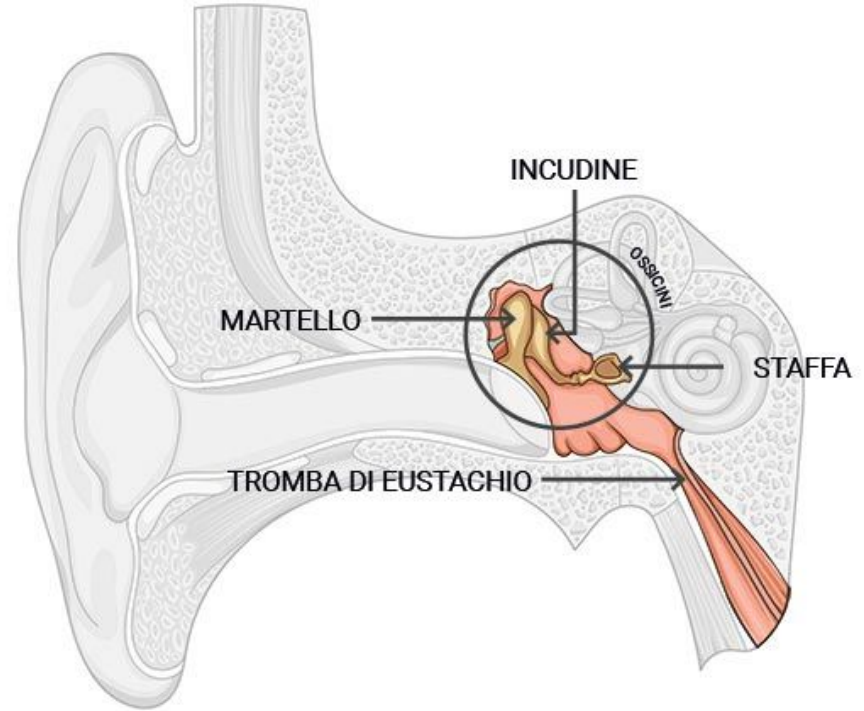
Passa dal canale uditivo che agisce come un filtro e fa vibrare il timpano.



## Orecchio medio

Il timpano è una membrana che vibra in base alle frequenze del suono.

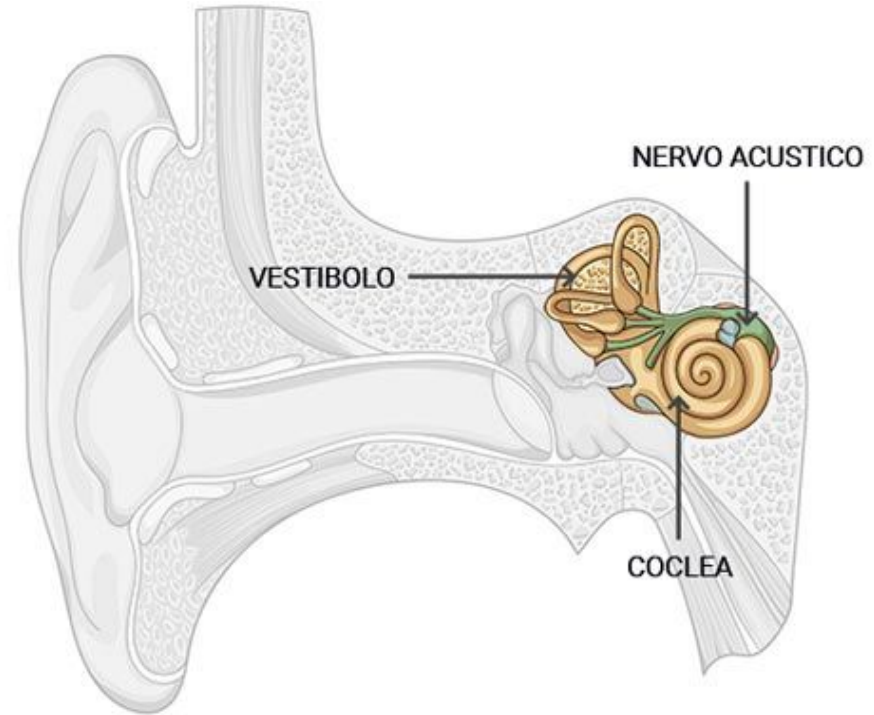
La vibrazione viene poi trasmessa tramite 3 ossicini: il martello, la staffa e l'incudine, alla finestra ovale della coclea.

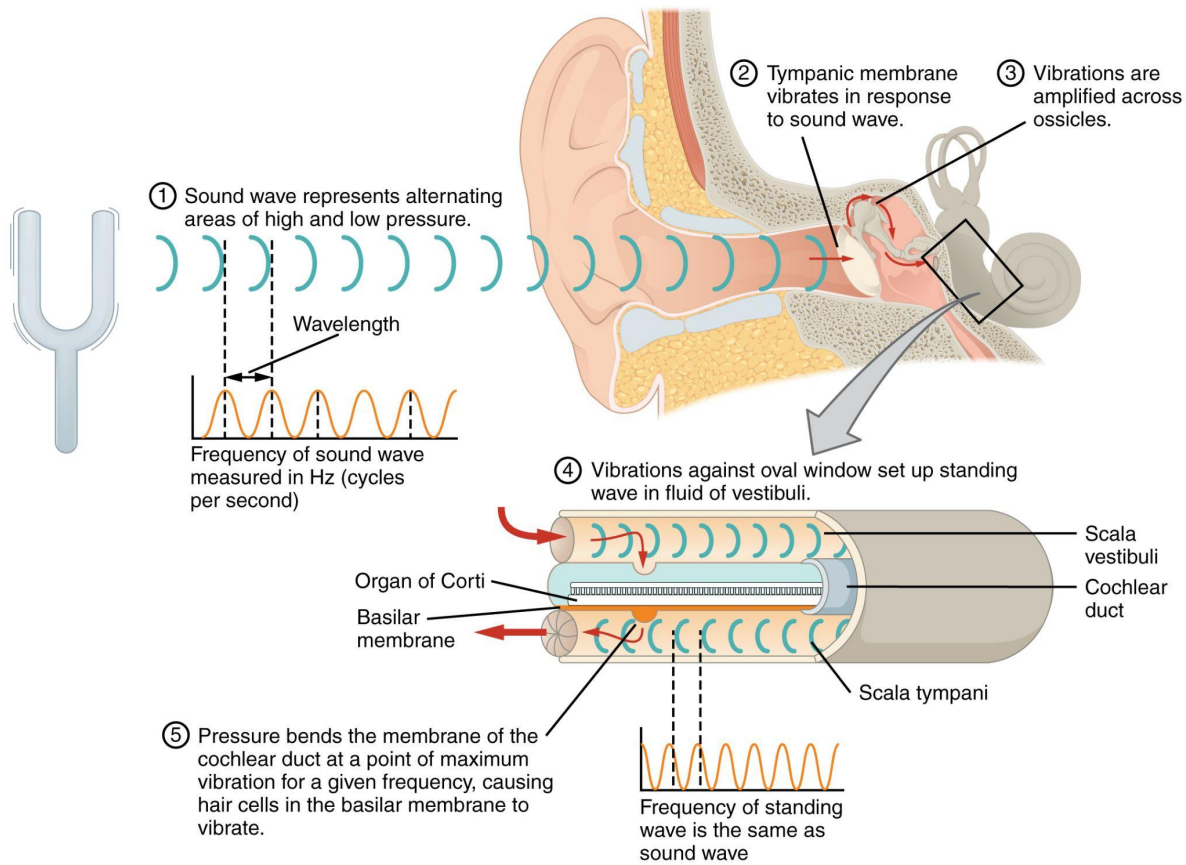




## Orecchio interno

L'orecchio interno ha il compito di tradurre la vibrazione in messaggio nervoso. All'interno della coclea la vibrazione della finestra ovale fa muovere un fluido che a sua volta farà oscillare la membrana basilare. Appoggiato su quest'ultima l'organo di corti converte lo stimolo in una risposta neurale che verrà trasmessa dal nervo uditivo al cervello.





breve riassunto

# Importazione e manipolazione del dataset

---



# Importazione del dataset

Il dataset (.csv) importato contiene due informazioni importanti:

- nome del file audio
- classe del campione

	fname	label
0	00044347.wav	Hi-hat
1	001ca53d.wav	Saxophone
2	002d256b.wav	Trumpet
3	0033e230.wav	Glockenspiel
4	00353774.wav	Cello

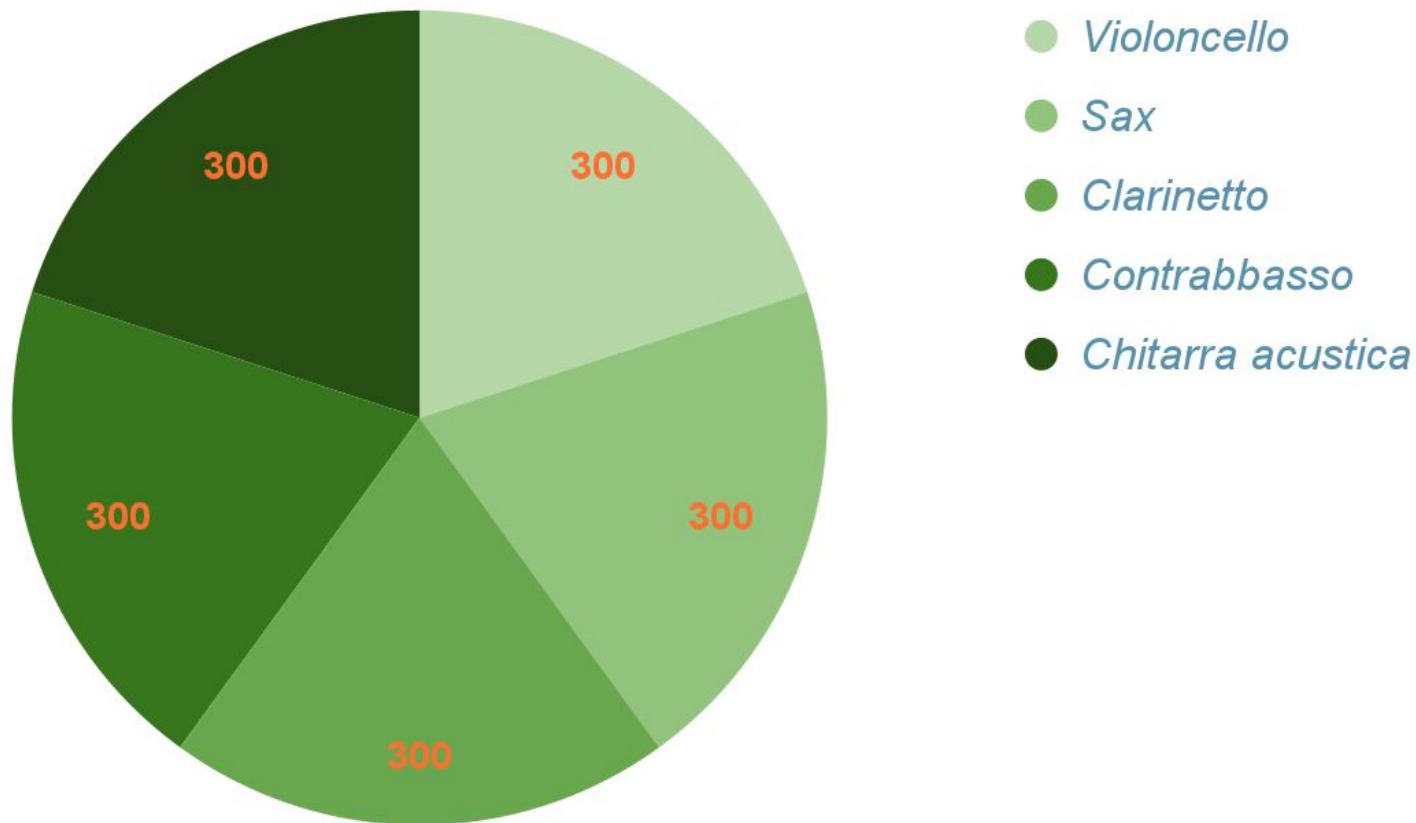


## Classi del dataset

Il numero di classi è piuttosto elevato (41) e queste sono di diversa natura: infatti, tra i file non sono presenti solamente degli strumenti musicali, ma anche suoni di altro tipo (urla, colpi di tosse, tuoni, ...).

Inoltre, il numero di file è molto grande; è dunque necessario ridurre il campione da analizzare.

## Strumenti selezionati





## Estrazione file audio

A questo punto abbiamo estratto due importanti informazioni dai file audio:

- intensità del suono campionato;
- frequenza di campionamento ( $F_s$ )

e le abbiamo inserite all'interno del dataset.

audio_waves	sample_rate
[-0.0007303721, -0.0010996412, -0.0010551845, ...	22050
[-0.003822653, -0.0053708917, -0.0044372473, -...	22050
[0.0031921505, 0.0048865937, 0.004513402, 0.00...	22050
[-0.00018609133, -0.00029739283, -0.0003075322...	22050
[0.00042805163, 0.0005860274, 0.00045838652, 0...	22050



## Lunghezza dei file audio

I dati introdotti nel passaggio precedente ci permettono di conoscere la lunghezza di un file:

- numero di campioni che compongono un segnale ( $N$ );
- in termini di tempo ( $N / F_s$ )

e, anche in questo caso, inseriamo i valori nel dataset.

bit_lengths	time_length
227556	10.32
99666	4.52
292824	13.28
154350	7
125685	5.7



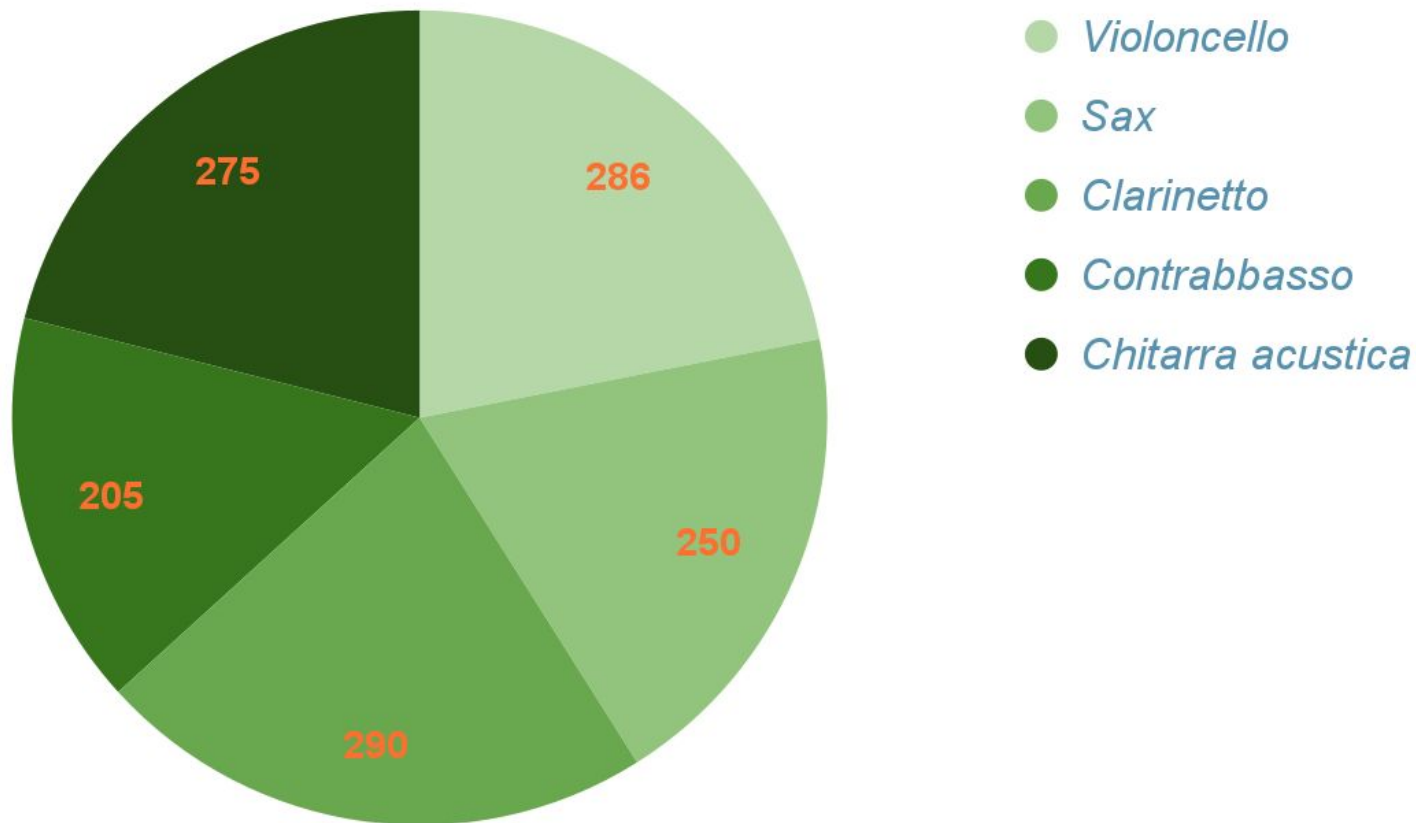


# Manipolazione del dataset

Le informazioni recuperate fin qui dai segnali sono sufficienti per il prosieguo del progetto.

Dal dataset corrente, andiamo ad eliminare tutti i file audio di lunghezza minore a 2 secondi, ossia tutti quei segnali che potrebbero contenere un numero di informazioni troppo limitato per essere classificati correttamente dalla CNN.

## Omogeneità del campione

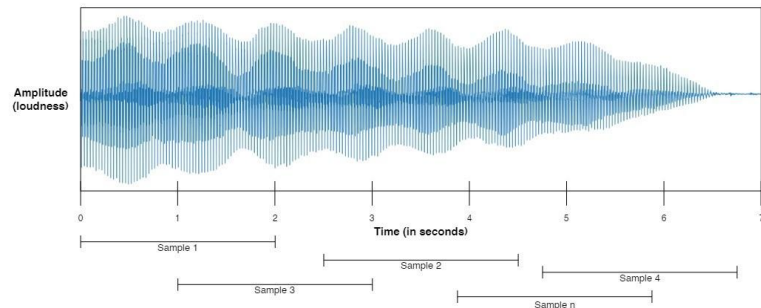


# Estrazione dei campioni per il training

La rete neurale non è in grado di adattarsi a campioni di lunghezza diversa ed è quindi necessario estrarre dai segnali restanti un set di campioni con una lunghezza ben definita.

Abbiamo scelto 2 secondi come valore, perché:

- il numero di feature estraibili permette alla CNN di classificare il segnale;
- il tempo di esecuzione della fase di training rimane contenuto, essendo un tempo piuttosto limitato.



# Pre-processing dei segnali audio

---



## Pronti per il training?

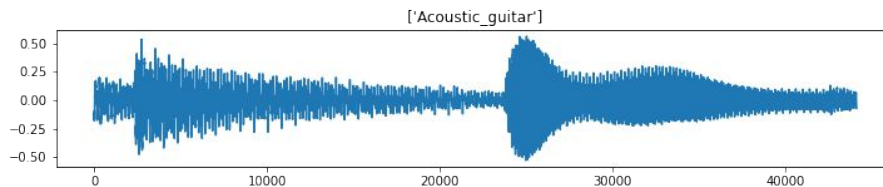
A questo punto, i campioni estratti hanno queste caratteristiche:

- sono tutti dello stesso formato;
- sono tutti della stessa lunghezza.

## Un ultimo step

Purtroppo, i valori dei segnali che abbiamo estratto non contengono informazioni che possano aiutare la CNN a catalogarli correttamente.

Infatti, al momento, l'unica informazione che ci danno i segnali è quella relativa all'intensità (potenza) di un suono, che non caratterizza affatto un suono e chi lo produce.





## Estrazione delle feature: MFCC

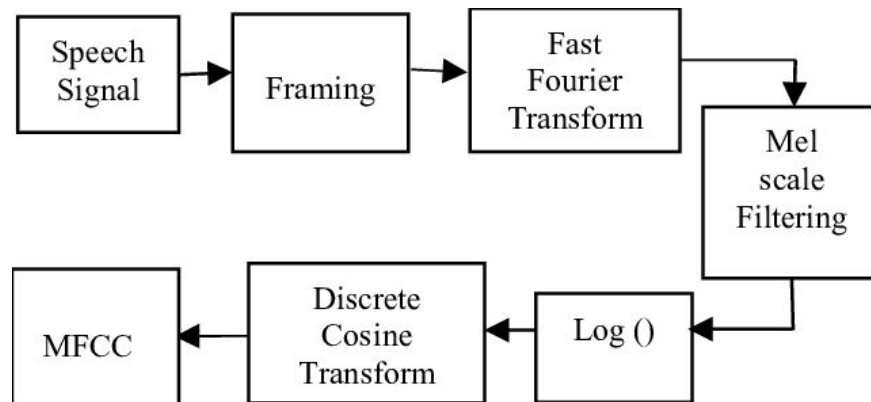
**MFCC** (*Mel-Frequency Cepstrum Coefficients*) è un insieme di feature spesso utilizzato nel mondo dell'*audio analysis* che permette di rappresentare le frequenze estratte dai file come suoni percepiti dall'orecchio umano.

Essendo uno strumento vastamente utilizzato, diverse librerie Python mettono a disposizione funzioni che calcolano rapidamente i coefficienti, che sono il risultato di complesse operazioni di elaborazione dei segnali.

## MFCC nel dettaglio

Quali sono questi passaggi matematici?

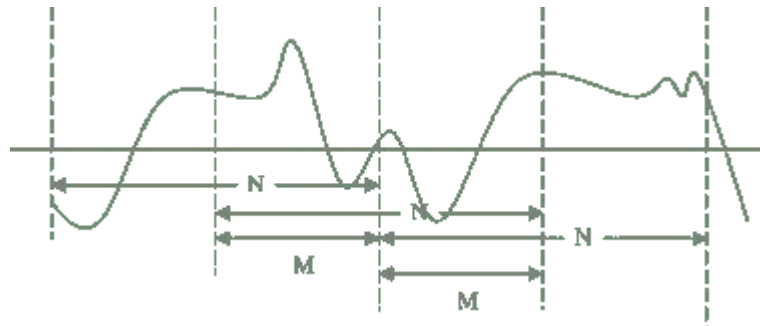
1. suddivisione del segnale;
2. calcolo della *densità spettrale di potenza*;
3. filtraggio con un *mel filterbank*;
4. logaritmo del segnale filtrato;
5. calcolo DCT;





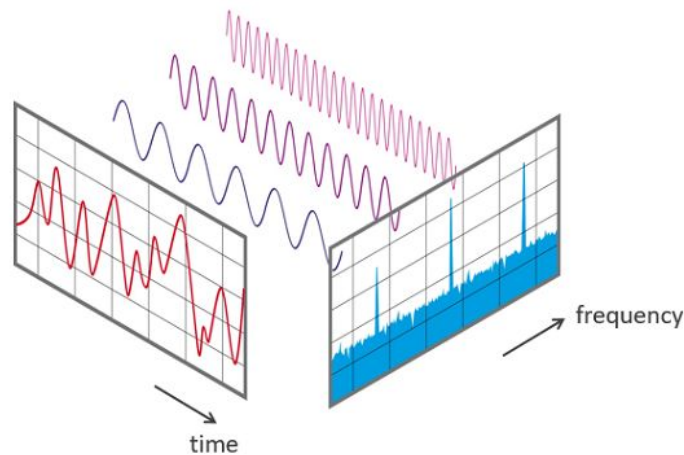
# Suddivisione del segnale

Nella prima fase, il segnale viene scomposto in tanti piccoli frame: in questo modo, avremo frame di segnali piuttosto omogenei tra loro (in termini di intensità).

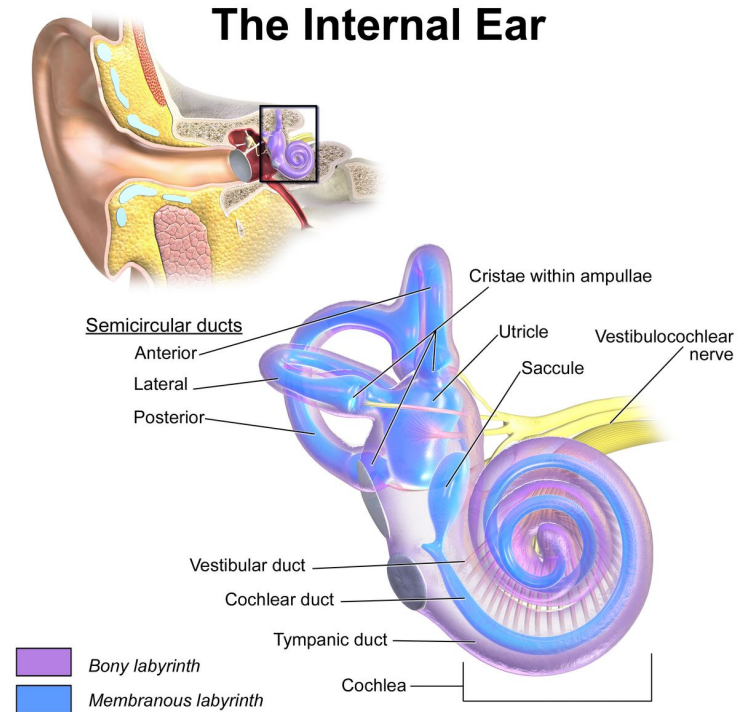


# Calcolo della densità spettrale di potenza

Calcoliamo la FFT di ogni frame creato e estraiamo la **densità spettrale di potenza** (*periodogramma*), ossia una stima dell'energia di un segnale...



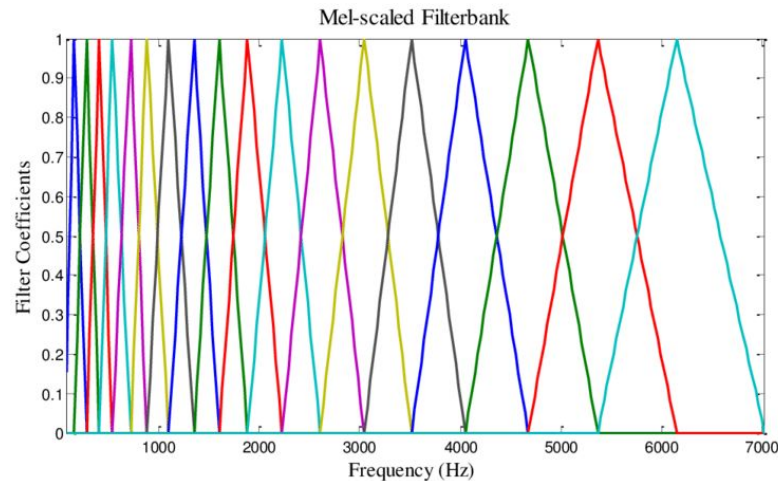
Questo calcolo viene effettuato per simulare il comportamento della *coclea*, una piccola componente dell'orecchio interno, che trasmette le vibrazioni all'organo del Corti.



## Filtraggio con un *mel filterbank*

In questa fase, filtriamo tutti i valori calcolati in precedenza utilizzando un *mel filterbank*.

In questo modo, possiamo rappresentare i segnali sulla scala di Mel, più funzionale per lo sviluppo del progetto.



## Logaritmo del segnale filtrato

Calcoliamo il logaritmo del segnale filtrato, questo perché l'orecchio umano percepisce un segnale doppiamente intenso quando la potenza del suono emesso è otto volte superiore.





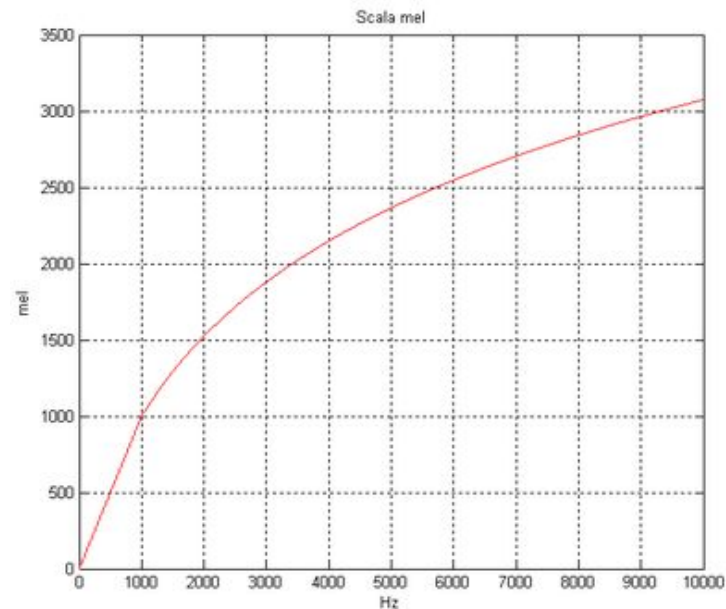
## Calcolo della DCT

Infine, calcoliamo la Discrete Cosine Transform sul segnale risultante; manteniamo dei componenti della DCT i primi 13, dato che mantenendo tutti i 26 coefficienti della DCT, la rappresentazione dei coefficienti MFCC sarebbe peggiorata.

# Perché rappresentare in scala Mel?

I coefficienti calcolati sono poi rappresentati in scala Mel, che è una scala di percezione dell'altezza del suono.

Rappresentando in questo modo il segnale, la rappresentazione delle intensità misurate sarà molto più vicina a quella percepita dall'orecchio umano.

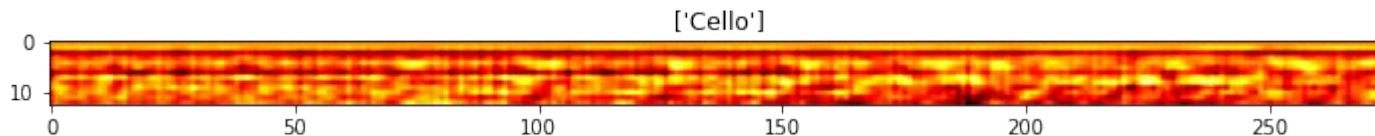


## E in Python?

Concretamente, la funzione *mfcc()* utilizzata nel codice Python produce una matrice di dimensioni:

- 275 (lunghezza del segnale);
- 13 (numero di feature utilizzate).

A questo punto, possiamo passare alla fase successiva, in cui implementiamo la rete.



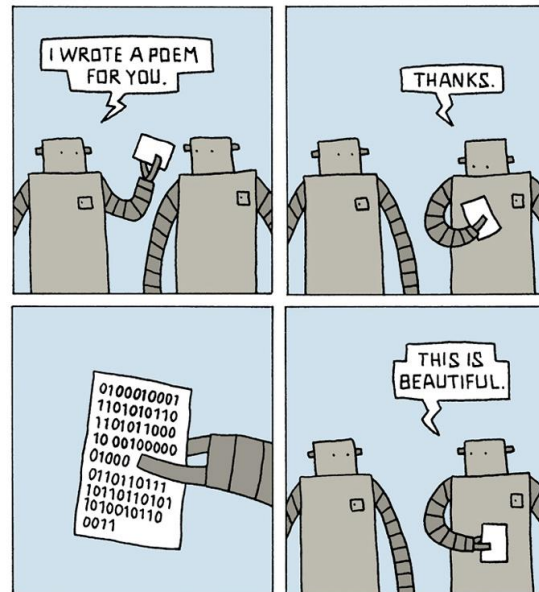


# Codifica dei label

---




# Perché?

- La nostra rete neurale lavora solo con i numeri, quindi dobbiamo codificare le etichette per renderle accettabili dalla rete.
- I label che abbiamo ora sono ancora nella forma di dati categorici non ordinali.



# Label encoding

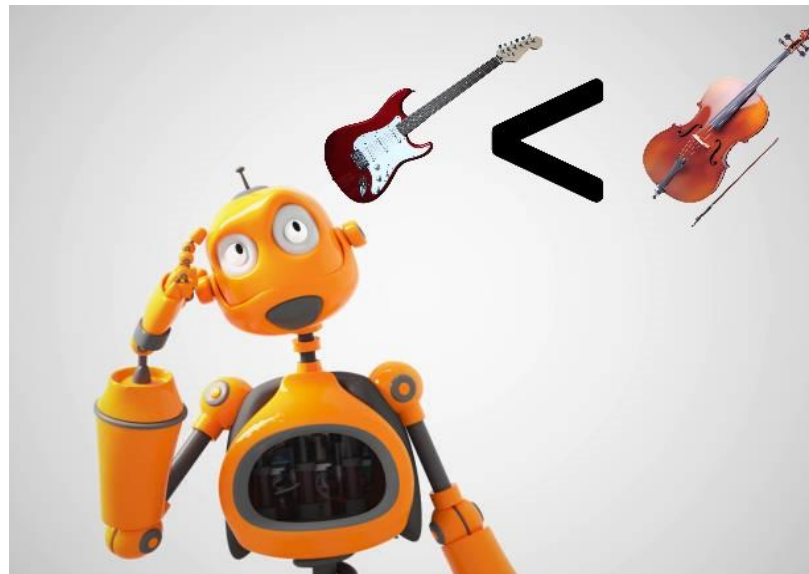
Utilizzando il label encoding ad ogni categoria di strumento musicale viene assegnato un valore intero, per esempio da “Acoustic\_guitar” si ottiene 0, da “Cello” 1 e così via (le etichette sono ordinate alfabeticamente).

	Encoding
	0
	1
	2

## Problema con il label encoding




Per alcune variabili, come le ordinali, questa codifica basterebbe.

Invece per variabili categoriche non ordinali (come quelle che abbiamo nel nostro caso) il solo label encoding potrebbe portare a risultati inaspettati.



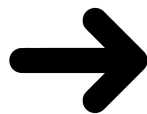
# One hot encoding

Questa codifica crea una nuova feature binaria per ogni possibile categoria (classe) e assegna il valore 1 alla feature di ogni campione che corrisponde alla sua categoria originale.

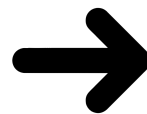
	Cat	Dog	Zebra
	1	0	0
	0	1	0
	0	0	1

## Cosa abbiamo ottenuto

```
[[ 'Saxophone']  
 [ 'Acoustic_guitar']  
 [ 'Clarinet']  
 ...  
 [ 'Saxophone']  
 [ 'Acoustic_guitar']  
 [ 'Double_bass']]
```



```
[[4],  
 [0],  
 [2],  
 ...,  
 [4],  
 [0],  
 [3]]
```



```
[[0. 0. 0. 0. 1.]  
 [1. 0. 0. 0. 0.]  
 [0. 0. 1. 0. 0.]  
 ...  
 [0. 0. 0. 0. 1.]  
 [1. 0. 0. 0. 0.]  
 [0. 0. 0. 1. 0.]]
```

# Implementazione della rete neurale

---



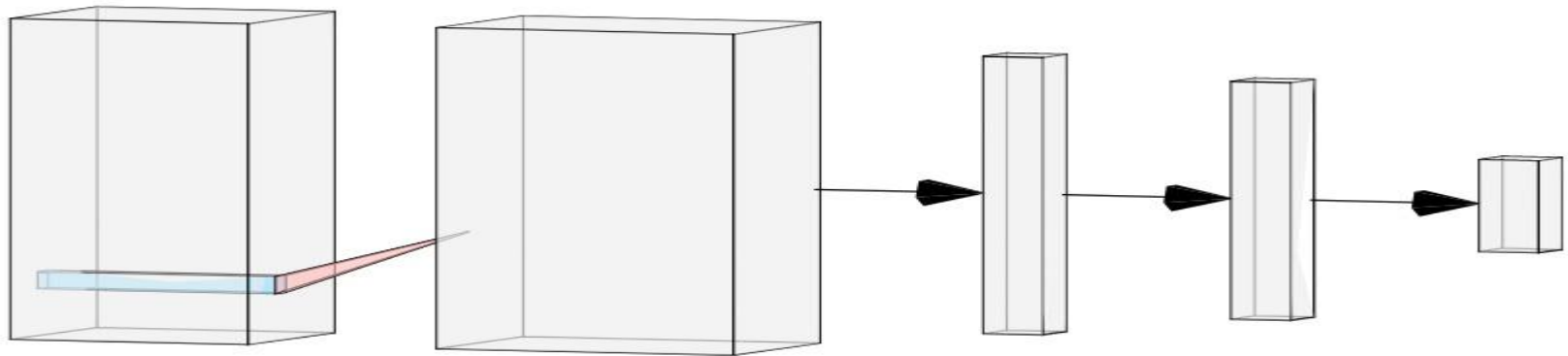
# CNN

- I dati da classificare sono immagini, ovvero le heatmap ottenute dal calcolo dei mfcc.
- Per questo motivo, utilizziamo una rete neurale convoluzionale, generalmente considerata la migliore per quanto riguarda la classificazione di immagini.
- Prima di costruire la rete però prepariamo i dati dividendoli tra train e test e cambiandone la shape per presentare alla rete le immagini con delle dimensioni che possa accettare.



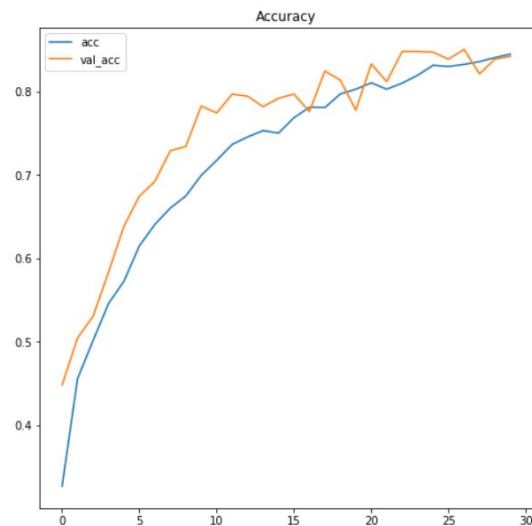
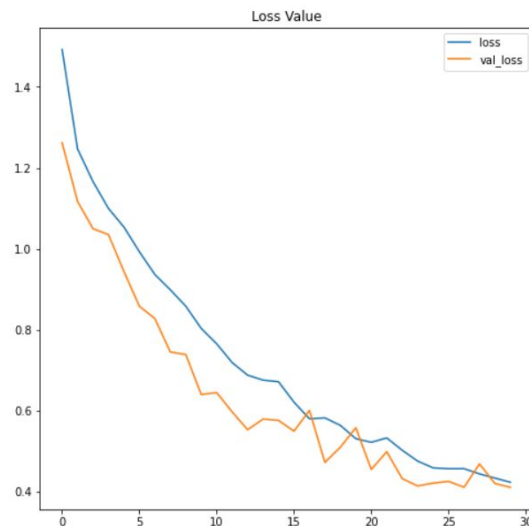
# Costruzione

Ora costruiamo la CNN: nella rete usiamo 2 layer convoluzionali e 3 densi, la funzione di attivazione è la ReLU (a parte per l'ultimo layer denso che deve dare l'output, per cui si usa una softmax).



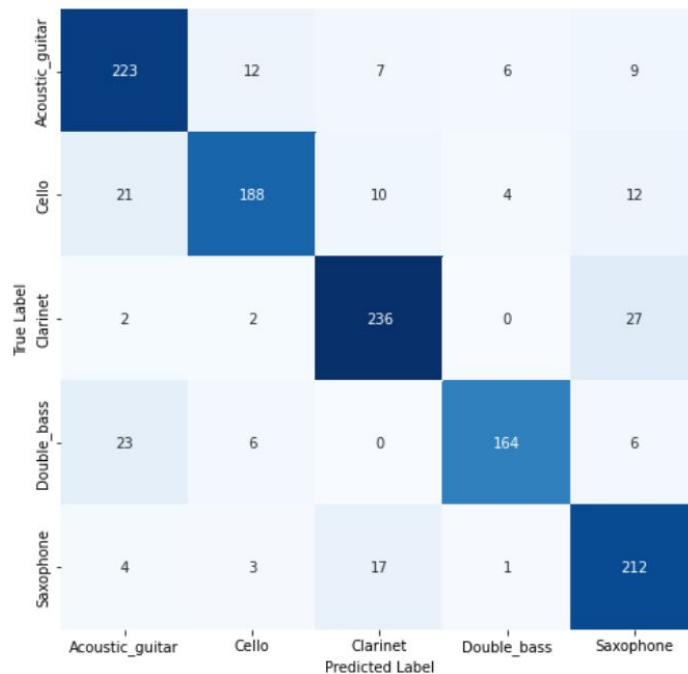
# Valutazione della rete

Facciamo il training del modello, usiamo 30 epoche e otteniamo una accuracy vicina all'85%, per vedere i miglioramenti del modello creiamo due grafici che rappresentano la loss e l'accuracy della rete al passare delle epoche.



# Matrice di confusione

Eseguiamo una predizione sui dati di test. Una volta resi comparabili i dati predetti e i label creiamo una matrice di confusione per visualizzare le performance del modello.



## Ora un pò di codice

webs

