



UNIVERSIDAD MAYOR DE SAN SIMÓN  
POSGRADO FACULTAD DE CIENCIAS Y TECNOLOGÍA



## **RECONOCIMIENTO DE VOZ CON MACHINE LEARNING**

Presentado por:

Eguivar Vilca Shirley Carminia

COCHABAMBA - BOLIVIA

Cochabamba 17 de Enero de 2019

## DEDICATORIA

A Dios por el soplo de vida, la voluntad y la oportunidad de seguir estudiando.

A mis padres quienes me dieron la vida, educación y apoyo. Con su ejemplo de empeño, constancia y coraje forjaron en mi la persona que hoy soy , no conozco a nadie en este mundo a quienes les deba mas amor y agradecimiento.

A la *vizcacha* que siempre esta ahí brindando el apoyo necesario en cada decisión tomada.

.

## INDICE DE CONTENIDO

<b>1 CAPITULO I - INTRODUCCION .....</b>	<b>4</b>
<b>1.1. ANTECEDENTES .....</b>	<b>5</b>
<b>1.2. DEFINICION DEL PROBLEMA .....</b>	<b>5</b>
<b>1.3 OBJETIVOS.....</b>	<b>6</b>
1.3.1    Objetivo General.....	6
1.3.2    Objetivos Específicos .....	6
<b>1.4. AREA DE CONOCIMIENTO: Machine Learning .....</b>	<b>6</b>
<b>1.5. ALCANCE.....</b>	<b>6</b>
<b>1.6. JUSTIFICACION .....</b>	<b>6</b>
<b>2 CAPITULO II - MARCO TEORICO .....</b>	<b>8</b>
<b>2.1. INTELIGENCIA ARTIFICIAL .....</b>	<b>8</b>
<b>2.2. MACHINE LEARNING .....</b>	<b>8</b>
2.2.1. Tipos de aprendizaje.....	8
2.2.1.1. Aprendizaje Supervisado .....	8
2.2.1.2. Aprendizaje No Supervisado.....	9
<b>2.3. DEEP LEARNING.....</b>	<b>10</b>
<b>2.4. REDES NEURONALES .....</b>	<b>11</b>
2.4.1.Redes Monocapa.....	11
2.4.1.Redes Multicapa .....	12
<b>3 CAPITULO III.....</b>	<b>13</b>
<b>RECONOCIMIENTO DE VOZ.....</b>	<b>13</b>
<b>3.1. MACHINE LEARNIG CON PYTHON .....</b>	<b>13</b>
3.1.1.Scikit-Learn.....	13
3.1.2. TensorFlow .....	13
3.1.3. Keras .....	13
3.1.4. PyTorch .....	14

3.1.5. Theano .....	14
<b>3.2. ANTECEDENTES - HISTORIA RECONOCIMIENTO DE VOZ .....</b>	<b>14</b>
<b>3.3.MECANISMO VOCAL.....</b>	<b>16</b>
<b>La voz consiste en el sonido producido por un ser humano haciendo uso de sus cuerdas vocales para hablar, cantar, reírse, gritar, chillar, etc. Su frecuencia oscila entre alrededor de 60 a 7000 Hz. ....</b>	<b>16</b>
<b>3.4. PROCESAMIENTO DE AUDIO CON PYTHON .....</b>	<b>17</b>
3.4.1. Librerías de Audio .....	17
3.4.1.1.Librosa .....	17
3.4.1.2. IPython.display.Audio.....	17
3.4.2. Cargando un archivo de audio con Python.....	18
3.4.3. Reproduciendo audio .....	18
3.4.4. Vizualizando Audio .....	18
3.4.4.1.Waveform/ Ondas .....	18
3.4.4.2.Espectograma.....	19
3.4.5. Extracción de Características.....	20
3.4.5.1. Zero Crossing Rate .....	20
3.4.5.2. Spectral Centroid .....	21
3.4.5.3. Spectral Rolloff.....	22
3.4.5.4. Mel-Frequency Cepstral Coefficients .....	23
3.4.5.5. Chroma Frequencies.....	25
<b>4 CAPITULO IV .....</b>	<b>26</b>
<b>RECONOCIMIENTO DE COMANDOS DE VOZ .....</b>	<b>26</b>
<b>4.1. INTRODUCCION .....</b>	<b>26</b>
<b>4.2. DATASET .....</b>	<b>27</b>
<b>4.3. PRE-PROCESAMIENTO DE ARCHIVOS DE AUDIO.....</b>	<b>28</b>
4.3.1. Extracción de espectograma por cada audio .....	28

4.3.1. Extraccion de Caracteristicas desde los espectogramas.....	29
<b>4.4. ENSAMBLADO DEL DATASET .....</b>	<b>30</b>
<b>4.5. ANALIZANDO LOS DATOS EN PANDAS .....</b>	<b>32</b>
4.5.1. Limpieza de datos .....	33
<b>4.6. DIVIDIENDO EL DATASET EN TRAINING AND TEST DATA.....</b>	<b>33</b>
<b>4.7. CLASIFICACION CON KERAS- CONSTRUCCION DE LA REDE NEURONAL .....</b>	<b>34</b>
4.7.1. Ajustes al modelo.....	34
<b>4.8. ENTRENAMIENTO DE LA RED .....</b>	<b>35</b>
4.8.1. Interpretacion de los resultados del entrenamiento .....	35
4.8.2. Evaluacion .....	36
<b>4.9. PREDICCIONES .....</b>	<b>36</b>
4.8.2. Analisis de Resultados.....	37
<b>5 CONCLUSIONES Y RECOMENDACIONES .....</b>	<b>38</b>
<b>6 BIBLIOGRAFIA .....</b>	<b>39</b>

## INDICE DE FIGURAS

Figura 1 Aprendizaje Supervisa y Aprendizaje no supervisado.....	10
Figura 2 Machine learning vs Deep Learning.....	10
Figura 3 Red Neuronal Monocapa .....	12
Figura 4 Red Neuronal Multicapa.....	12
Figura 5 Mecanisvo vocal .....	16
Figura 6 Cargando audio Python .....	18
Figura 7 Reproduciendo audio Python .....	18
Figura 8 Visualizando audio con Python .....	19
Figura 9 Espectograma de un audio .....	19
Figura 10 Zero Crossing Rate.....	20
Figura 11 Spectral Centroid .....	22
Figura 12 Spectral Rolloff .....	23
Figura 13 MFCC de un audio.....	24
Figura 14 Chroma Frequency .....	26
Figura 15 Proceso de reconocimiento de voz .....	26
Figura 16 Fuente del audioset .....	27
Figura 17 Estructura de carpetas del audioset.....	28
Figura 18 Extraccion de espectogramas.....	29
Figura 19 Ejemplo de los espectogramas generados en el proyecto .....	29
Figura 20 Extraccion de características a partir de un mfcc .....	30
Figura 21 Ensamblado del dataset.....	31
Figura 22 Ejemplo de dataset generado en el proyecto .....	31
Figura 23 Dataset desde pandas .....	32
Figura 24 Analisis del dataset.....	32
Figura 25 Dividiendo el dataset para el entrenamiento y test.....	33
Figura 26 Resumen del modelo construido.....	34
Figura 27 Compilando el modelo .....	35
Figura 28 Entrenando la red neuronal.....	35
Figura 29 Epoch numero 1.....	35
Figura 30 Epoch numero 50.....	36
Figura 31 Epoch numero 500.....	36
Figura 32 Evaluacion del modelo .....	36
Figura 33 Predicciones con los datos de test.....	37
Figura 34 Analisis de resultados .....	37

## 1 CAPITULO I - INTRODUCCION

*"Machine learning is a core, transformative way by which we're rethinking everything we're doing". – Sundar Pichai*

### 1.1. ANTECEDENTES

En los últimos años el término de machine learning está adquiriendo más notoriedad en el ámbito tecnológico. Se ha producido un fuerte crecimiento de la inversión en este campo, el auge de los smartphones ha permitido que inicien a desarrollarse soluciones comerciales basadas en machine learning que son usadas por una gran cantidad de usuarios como ejemplo tenemos los llamados asistentes virtuales.

Grandes empresas de la industria tecnológica han incorporado los asistentes virtuales dentro de sus dispositivos tales como Apple (Siri), Google (Google Assistant y Google Now), Microsoft (Cortana), Amazon (Alexa), Samsung (Bixby) estas aplicaciones permiten que el usuario realice acciones mediante comandos de voz.

Las aplicaciones de reconocimiento de voz incluyen interfaces de usuario de voz como la marcación por voz, enrutamiento de llamadas, control doméstico de dispositivos, búsqueda, etc.

Las interfaces de usuario de voz, son las tecnologías que permiten a las personas interactuar con computadoras y dispositivos a través de la entrada de voz haciendo que los dispositivos actúen de forma inteligente entendiendo y respondiendo a las diferentes instrucciones.

Las tecnologías que están detrás de estas interfaces de usuario de voz utilizan machine learning que combinados con otras tecnologías permiten la comunicación de los usuarios con las computadoras.

### 1.2. DEFINICION DEL PROBLEMA

Con el avance de la tecnología hoy en día, el usuario puede abrir el navegador, consultar la hora, cerrar una ventana en particular, escribir un texto, realizar cálculos todo esto por medio de comandos de voz. El reconocimiento de voz no cambia lo que la computadora solía hacer, pero sí cambia la forma en que el usuario lo hace.

El presente proyecto plantea como problema el reconocimiento de voz utilizando un algoritmo de machine learning.

Como podría reconocer comandos de voz para ejecutar tareas rutinarias a través del uso de la tecnología?

### **1.3 OBJETIVOS**

#### **1.3.1 Objetivo General**

Implementar un modelo de machine learning para reconocimiento de voz para ejecutar tareas rutinarias a través del uso de la tecnología.

#### **1.3.2 Objetivos Específicos**

- Definir el dataset a utilizar por el modelo
- Preparación y limpieza de datos
- Desarrollar un modelo machine learning utilizando redes neuronales.
- Interpretar los resultados finales

### **1.4. AREA DE CONOCIMIENTO: Machine Learning**

### **1.5. ALCANCE**

La detección de comandos disparadores Keyword Spotting, consiste en la detección de determinados comandos en una grabación de audio, y al reconocer dichos comandos de forma automática el sistema pueda ejecutar las acciones correspondientes.

Se definen como alcances del proyecto los siguientes puntos:

- Si bien en un entorno real el sistema recibe un stream de audio de forma continua, en el presente proyecto, se considera archivos de audio como entrada con una duración acotada.
- Además mencionar que el objetivo principal del proyecto es el reconocimiento de voz aplicando los conocimientos adquiridos de machine learning, por lo que la ejecución de comandos luego del reconocimiento de voz no será parte de este proyecto.

### **1.6. JUSTIFICACION**

Hoy en día las interfaces de usuario de voz están adquiriendo gran importancia en la vida cotidiana, un ejemplo reciente fue lo ocurrido en diciembre 2019, un hombre fue rescatado después de gritar "Siri, llama al 911" cuando su automóvil había caído en un río congelado.(CNN)

Estas aplicaciones de reconocimiento de voz cumplen un papel muy importante dentro del área tecnológica y cada vez es más común encontrarlas o escuchar de ellas en situaciones del vivir



diario como en teléfonos, televisores, hogares inteligentes y otros productos. Los dispositivos controlados por voz, como Apple HomePod, Google Home y Amazon Echo, están adquiriendo una fuerte demanda en el mercado.

Por estas razones existe la necesidad de crear nuevos recursos y herramientas que faciliten el manejo de estas tecnologías.

## 2 CAPITULO II - MARCO TEORICO

### 2.1. INTELIGENCIA ARTIFICIAL

La inteligencia artificial (IA) hace posible que las máquinas aprendan de la experiencia, se ajusten a nuevas aportaciones y realicen tareas como hacen los humanos. La mayoría de los ejemplos de inteligencia artificial de los que usted escucha hoy día desde computadoras que juegan ajedrez hasta automóviles que se conducen por sí solos, se sustentan mayormente en aprendizaje a fondo (deep learning) y procesamiento del lenguaje natural. Mediante el uso de estas tecnologías, las computadoras pueden ser entrenadas para realizar tareas específicas procesando grandes cantidades de datos y reconociendo patrones en los datos.

### 2.2. MACHINE LEARNING

El Aprendizaje Automático de las máquinas (Machine Learning) tiene como objetivo desarrollar algoritmos que permitan aprender sobre un conjunto de observaciones, de tal manera que sea posible establecer hipótesis generales o predicciones sobre nuevas observaciones, y de esta forma, tomar decisiones automáticamente, dotando así a un sistema de Inteligencia Artificial.

El Aprendizaje Automático es utilizado en una gran variedad de ámbitos, desde el reconocimiento de voz, hasta la exploración o minería de datos, existiendo una gran variedad de algoritmos. Este tipo de algoritmos tienen dos fases fundamentales: la fase de entrenamiento y la fase de evaluación.

#### 2.2.1. Tipos de aprendizaje

##### 2.2.1.1. Aprendizaje Supervisado

En el aprendizaje supervisado, los algoritmos trabajan con datos “etiquetados” (*labeled data*), intentando encontrar una función que, dadas las variables de entrada (*input data*), les asigne la etiqueta de salida adecuada. El algoritmo se entrena con un “histórico” de datos y así “aprende” a asignar la etiqueta de salida adecuada a un nuevo valor, es decir, predice el valor de salida. Ver figura 1.

El aprendizaje supervisado se suele usar en:

- Problemas de *clasificación* (identificación de dígitos, diagnósticos, o detección de fraude de identidad).
- Problemas de *regresión* (predicciones meteorológicas, de expectativa de vida, de crecimiento etc).

Estos dos tipos principales de aprendizaje supervisado, clasificación y regresión, se distinguen por el tipo de variable objetivo. En los casos de clasificación, *es de tipo categórico*, mientras que, en los casos de regresión, la variable objetivo *es de tipo numérico*.

Los algoritmos más habituales que aplican para el aprendizaje supervisado son:

- Árboles de decisión.
- Clasificación de Naïve Bayes.
- Regresión por mínimos cuadrados.
- Regresión Logística.
- Support Vector Machines (SVM).

Métodos “Ensemble” (Conjuntos de clasificadores).

### **2.2.1.2. Aprendizaje No Supervisado**

El aprendizaje no supervisado tiene lugar cuando no se dispone de datos “etiquetados” para el entrenamiento. Sólo conocemos los datos de entrada, pero no existen datos de salida que correspondan a un determinado *input*. Por tanto, sólo podemos describir la estructura de los datos, para intentar encontrar algún tipo de organización que simplifique el análisis. Por ello, tienen un carácter exploratorio. Ver figura 1.

El aprendizaje no supervisado se suele usar en:

- Problemas de *clustering*.
- Agrupamientos de co-ocurrencias.
- Perfilado o *profiling*.

Si embargo, los problemas que implican tareas de encontrar similitud, predicción de enlaces o reducción de datos, pueden ser supervisados o no.

Los tipos de algoritmo más habituales en aprendizaje no supervisado son:

- Algoritmos de clustering.
- Análisis de componentes principales.
- Descomposición en valores singulares (singular valuedecomposition).
- Análisis de componentes principales (IndependentComponentAnalysis).[3]

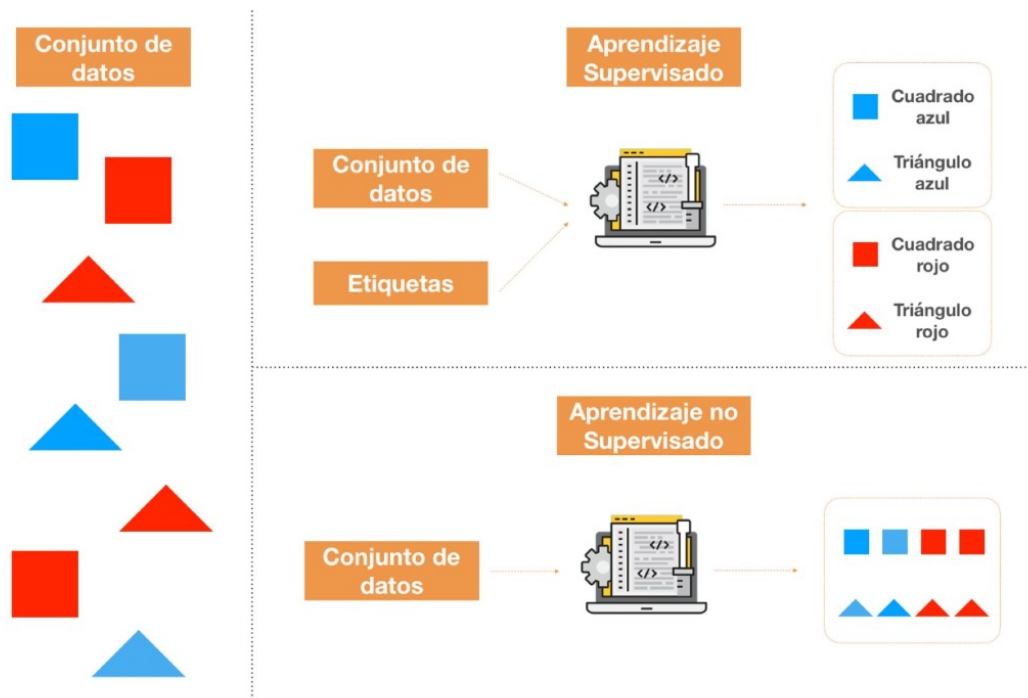


Figura 1 Aprendizaje Supervisa y Aprendizaje no supervisado

## 2.3. DEEP LEARNING

El deeplearning es un tipo de machine learning que entrena a una computadora para que realice tareas como las hacemos los seres humanos, como el reconocimiento del habla, la identificación de imágenes o hacer predicciones. En lugar de organizar datos para que se ejecuten a través de ecuaciones predefinidas, el deeplearning configura parámetros básicos acerca de los datos y entrena a la computadora para que aprenda por cuenta propia reconociendo patrones mediante el uso de muchas capas de procesamiento.[4]

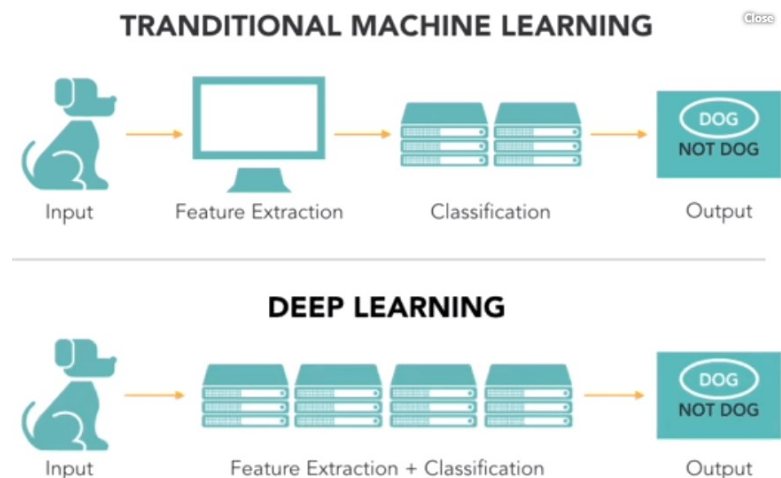


Figura 2 Machine learning vs Deep Learning.

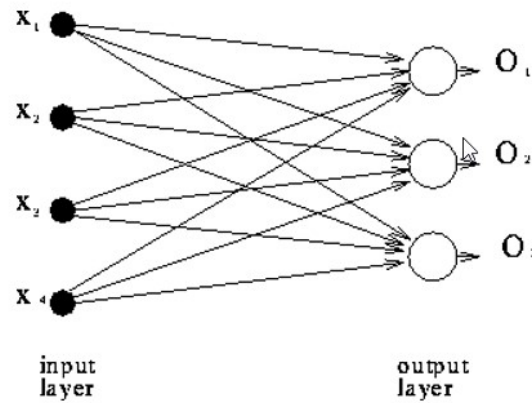
## 2.4. REDES NEURONALES

Una red neuronal consiste en un conjunto de neuronas separadas por capas y comunicadas entre ellas mediante conexiones. Cada neurona recibe un valor inicial, y esta lo transforma y reenvía a todas las neuronas con las que esté conectada en adelante. Estos valores iniciales pueden proceder o de la entrada inicial a la red, o de incluso otra neurona. Cada neurona cuenta con una función de activación y cada conexión con un peso. Las funciones de activación son funciones limitadoras, que o bien modifican el valor que emite la neurona o impone un mínimo para que este valor sea propagado. En cada conexión que comunica una neurona A con una neurona B, se multiplica el valor de salida de la neurona A por el valor del peso de la conexión, quedando como resultado que la entrada que recibe la neurona B es igual a la suma de todas las salidas multiplicadas por todos los pesos conectados con ella. Así, la entrada se va procesando hasta que alcanza la última capa o capa de salida, de la que se extraen los resultados del cómputo. La fortaleza de las redes neuronales reside en que los pesos de las conexiones pueden ajustarse según si los resultados obtenidos en la capa de salida son correctos o no, en un proceso conocido como entrenamiento. De esta manera, si el resultado obtenido es acertado, se refuerza el comportamiento de la red ante ese tipo de entradas, mientras que, si el resultado es erróneo, se ajustan los pesos para tratar de “acercar” los futuros resultados a una solución satisfactoria. A cada uno de los pasos en el entrenamiento de una red neuronal se le conoce como “época” (o Epoch en inglés)

### 2.4.1. Redes Monocapa

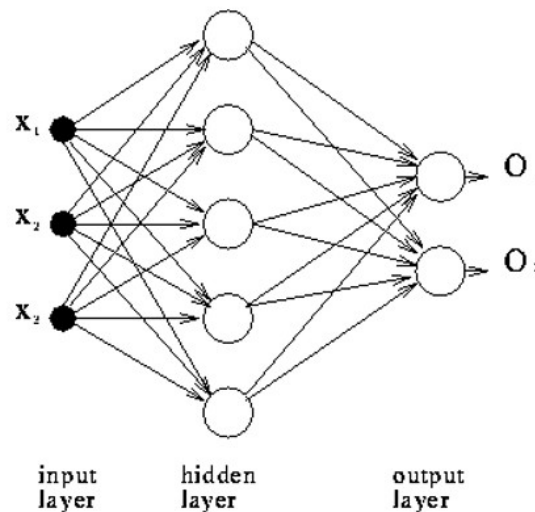
Esta es la arquitectura de red neuronal más simple. En ella, sólo existe una capa de neuronas que recibe las entradas y directamente las emite como una o varias salidas calculadas. Debido a su simpleza, este es un tipo de red neuronal muy adecuado para problemas simples de clasificación, con pocas entradas y pocas clases o etiquetas como salidas, ya que son también redes que obtienen resultados muy rápidos.

Incluso en su simpleza, contando con solo una capa se pueden alcanzar sistemas ligeramente más sofisticados, como una red de Hopfield, en la cual las neuronas tienen a su vez conexiones con neuronas de la misma capa. Este tipo de redes también se conoce como redes de neuronas recurrentes.

*Figura 3 Red Neuronal Monocapa*

### 2.4.1.Redes Multicapa

Añadiendo capas de neuronas, se alcanza una mayor capacidad de cómputo y unos resultados más precisos. La desventaja de estos tipos de redes es que, debido a su mayor tamaño, también acarrearán más necesidad de tiempo y recursos para llegar a la solución y ser entrenadas. La diferencia de estas redes con las redes monocapa es que estas incluyen una o más capas intermedias llamadas capas ocultas, que continúan el proceso de emitir resultados a otras neuronas.

*Figura 4 Red Neuronal Multicapa*

### 3 CAPITULO III

#### RECONOCIMIENTO DE VOZ

##### 3.1. MACHINE LEARNIG CON PYTHON

La mayoría de la documentación encontrada relativa al diseño e implementación de redes de neuronas estaba orientada a Phyton. Todas las librerías de uso extendido en el sector, ya sean de manipulación de datos o de *MachineLearning*, son compatibles con este lenguaje, o directamente han sido desarrollados en el.

En adición a la sencillez del propio lenguaje y a su sintaxis simple, que facilita el despliegue y mantenimiento del código, desde un principio se valoro como la opción mas adecuada para el desarrollo del proyecto.

###### 3.1.1.Scikit-Learn

Scikit-Learn es un paquete basado en SciPy, haciendo uso intenso de sus operaciones matemáticas. Proporciona una interfaz concisa y consistente para los algoritmos de *Machine Learning* mas comunes, facilitando su incorporación en sistemas en producción.

La librería combina un código de calidad y una buena documentación, así como una sencillez de uso y un alto rendimiento. Todo esto la ha convertido en la librería estándar para desarrollar modelos de *Machine Learning* en Python.

###### 3.1.2. TensorFlow

Desarrollada inicialmente por Google, desde la liberación de su código ha ganado popularidad rápidamente, eclipsando en gran medida a las otras alternativas.

Se basa en grafos de computación de flujo de datos, y destaca por su sistema de nodos multicapa que permite un rápido entrenamiento de redes de neuronas en grandes *dataset*. Además, se ha usado para una variedad de aplicaciones del mundo real, como los servicios de Google para reconocimiento de voz (Google VoiceRecognition) y de identificación de objetos en imágenes (Google ImageRecognition).

###### 3.1.3. Keras

Keras es una librería que permite construir redes de neuronas a alto nivel, de forma sencilla y minimalista. Esta escrita en Python y permite trabajar sobre Theano, TensorFlow y Microsoft Toolking (CNTK).

Keras es altamente modular y escalable, además de permitir el prototipado rápido de redes de neuronas por su sencillez. Los datos se preparan en tensores y sus modelos se basan en capas, con una capa para los tensores de entrada, otra para los de salida y un número de indeterminado de capas ocultas.

#### **3.1.4. PyTorch**

PyTorch es un *framework* de *Deep Learning* para una experimentación rápida y flexible. Permite la computación con tensores con una potente aceleración por GPU.

Usualmente, se utiliza como sustituta a NumPy para aprovechar la técnica de cómputo de las GPUs, proporcionando integración con librerías de aceleración de GPU como Intel MKL y NVIDIA CuDNN.

#### **3.1.5. Theano**

Theano es una librería que define arrays multidimensionales de una forma similar a NumPy, así como operaciones y expresiones matemáticas. Se proporciona compilada, de forma que puede ejecutarse de forma eficiente en todas las arquitecturas.

Destaca por su estrecha integración con NumPy a bajo nivel para sus operaciones, así como su optimización de uso de GPU y CPU, incrementando el rendimiento para la computación intensiva de datos. Además, sus ajustes de eficiencia y estabilidad permiten obtener resultados muchísimos más precisos, incluso con valores muy pequeños.

### **3.2. ANTECEDENTES - HISTORIA RECONOCIMIENTO DE VOZ**

El reconocimiento de voz tal y como lo conocemos empezó en 1952, ese año Bell Labs desarrolló un sistema capaz de identificar los dígitos de forma automática, a este sistema lo denominaron.

Después, los años 60, aparecieron numerosos algoritmos y técnicas. Por un lado, los soviéticos inventaron el algoritmo de “Dynamic Time Warping”(DTW) capaz de identificar 200 palabras segmentando el audio en fragmentos de 10ms. En paralelo, Leonard Baun desarrolló las matemáticas de las cadenas de Markov, siendo la base de los modelos probabilísticos desarrollados por primera vez por James & Janet Baker poco tiempo después.

A finales de los 70, principios de los 80, IBM mejoró las técnicas existentes presentando “Tangora”, un sistema de reconocimiento de voz basado en HMM que permitía reconocer 20.000 palabras, desplazando a las técnicas existentes que empleaban DTW; este sistema contaba con un elaborado modelo de lenguaje que incrementó notablemente la precisión de los algoritmos y técnicas de la época.



A finales de los años 80, Slava M. Katz desarrollo los modelos de lenguaje N-gram, que modelan la probabilidad condicionada de una palabra basandose en N-1 palabras de la secuencia; estos modelos de lenguaje mejoraron en gran medida los sistemas de reconocimiento de voz existentes hasta la fecha.

A principios de los años 90, aparecio en el mercado Dragon Dictation, siendo una de las primeras empresas en comercializar a nivel general un sistema de reconocimiento de voz, a mediados de los 90, Dragon Dictation fue comprada por Lernout&Hauspie, dando pie al desarrollo posterior de uno de los algoritmos de reconocimientos mas extendidos de la historia, el algoritmo de reconocimiento de voz integrado con Windows XP que, debido a su masificacion internacional, convirtio a Lernout&Hauspie en los lideres de la industria en tecnologias de reconocimiento de voz.

Mas adelante, Lernout&Hauspie desaparecio debido a un escandalo contable en 2001, vendiendo su tecnologia de reconocimiento de voz a ScanSoft, quienes pasaron a desarrollar y mantener el algoritmo de reconocimiento de voz hasta su cambio de nombre en 2005 por Nuance.

En 2005, ScanSoft, ahora Nuance, se alio con Apple para ofrecer las tecnologias de reconocimiento de voz en los productos de la empresa de la manzana, fruto de esta union comercial, Apple obtuvo la licencia del software para proporcionar la capacidad de reconocimiento de voz a su asistente de voz virtual Siri.

En 2007, Google empezo a contratar investigadores de Nuance, la finalidad de ello fue desarrollar un servicio de directorio telefonico que permitio a Google recopilar cientos de horas de voz que posteriormente emplearia para desarrollar "Google VoiceSearch", con su propio sistema de reconocimiento de voz; posteriormente, Google integro su tecnologia en la mayoria de sus productos: Maps, Earth, Google Translateapp y muchos mas.

### 3.3.MECANISMO VOCAL

La voz consiste en el sonido producido por un ser humano haciendo uso de sus cuerdas vocales para hablar, cantar, reírse, gritar, chillar, etc. Su frecuencia oscila entre alrededor de 60 a 7000 Hz.

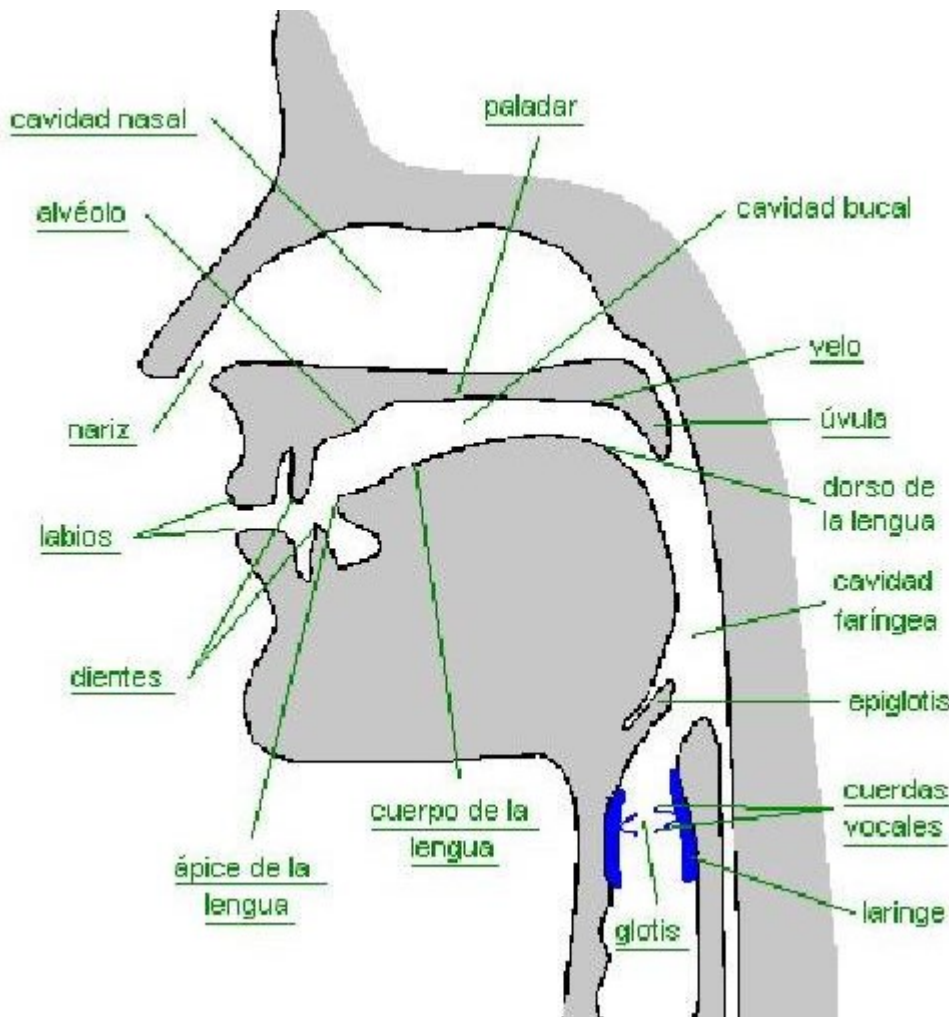


Figura 5 Mecanismo vocal

La voz comienza con la inspiración (toma de aire), el aire llena los pulmones y luego sale a través de la espiración. Cuando la espiración ocurre, el aire pasa de los pulmones a la tráquea y de ésta a la laringe y ahí, pasa por en medio de las cuerdas vocales; dos músculos pequeños que se tensan para producir sonidos agudos y se relajan para los sonidos graves. De este modo la voz adquiere su tono.

El flujo de aire continúa su camino hacia la boca, en donde la lengua, mandíbula, paladar y labios se encargan de darle forma al sonido y se producen los fonemas

(vocales y consonantes). Ahí es donde la voz adquiere su forma, lo que llamamos dicción o articulación.

Al mismo tiempo el sonido producido por la laringe y la boca adquiere su timbre característico al resonar dentro de las cajas de resonancia ubicados en los huesos, pecho, laringe, boca, nariz y cráneo. A través de este mecanismo, la voz también se amplifica naturalmente.

### **3.4. PROCESAMIENTO DE AUDIO CON PYTHON**

El sonido se representa en forma de una señal de audio que tiene características como frecuencia, ancho de banda, decibeles, función de la amplitud , tiempo , etc.

Los sonidos están disponibles en muchos formatos digitales lo que permite que sean leídos y analizados por la computadora. Ejemplo de formatos: mp3 ,wma , wav.En el presente documento analizaremos archivos en formato wav.

Python tiene excelentes librerías para el procesamiento de audio como Librosa, PyAudio.

A continuación se detallan las utilizadas en el presente proyecto.

#### **3.4.1. Librerías de Audio**

##### **3.4.1.1.Librosa**

Es un módulo de Python para analizar señales de audio en general,

[`pip install librosa`](#)

Para aumentar más potencia de decodificación/ análisis de audio, también se puede instalar *ffmpeg*<sup>1</sup> que incluye muchos decodificadores de audio.

##### **3.4.1.2. IPython.display.Audio**

IPython.display.Audio le permite reproducir audio directamente en un jupyter notebook.

---

<sup>1</sup>FFmpeg es framework capaz de decodificar, codificar, transcodificar, filtrar y reproducir un sinfín de formatos.

### 3.4.2. Cargando un archivo de audio con Python

```
In [45]: import librosa
audio_path = 'house_0001.wav'
x, sr = librosa.load(audio_path)
print(type(x), type(sr))
print(x.shape, sr)

<class 'numpy.ndarray'> <class 'int'>
(22050,) 22050
```

Figura 6 Cargando audio Python

Las anteriores instrucciones devuelven una serie temporal de audio como una matriz numpy con una frecuencia de muestreo.

### 3.4.3. Reproduciendo audio

Se usa `IPython.display.Audio` para reproducir audio.

```
In [46]: import IPython.display as ipd
         ipd.Audio(audio_path)

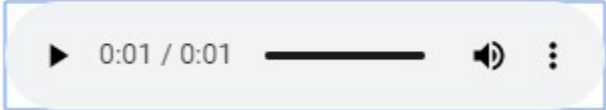
Out[46]: 
```

Figura 7 Reproduciendo audio Python

La instrucción `ipd.Audio(audio_path)` devuelve un widget de audio en el notebookjupyter como se vio en la imagen anterior.

### 3.4.4. Vizualizando Audio

#### 3.4.4.1. Waveform/ Ondas

Se puede trazar la matriz de audio usando `librosa.display.waveplot`

```
In [48]: %matplotlib inline
import matplotlib.pyplot as plt
import librosa.display
plt.figure(figsize=(14, 5))
librosa.display.waveplot(x, sr=sr)
```

```
Out[48]: <matplotlib.collections.PolyCollection at 0x24840e23a58>
```

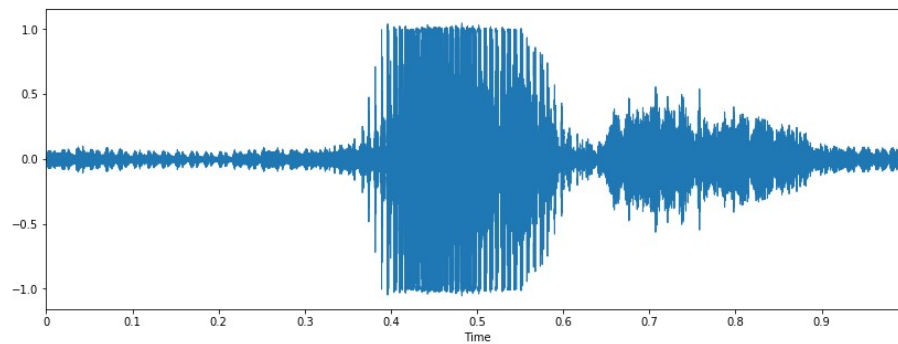


Figura 8 Visualizando audio con Python

### 3.4.4.2. Espectrograma

Un espectrograma es una representación visual del espectro de frecuencias de sonido que varían con el tiempo. Un espectrograma se utiliza para el análisis de las señales eléctricas, de comunicaciones, y cualquier señal audiovisual en su contenido frecuencial. Es una representación en tres dimensiones, temporal, frecuencial y amplitud de la distribución de energía de una señal en este caso del sonido. Los espectrogramas a veces se llaman ecografías, o diagramas de voz.

Se puede mostrar un espectrograma usando `librosa.display.specshow`.

```
In [50]: X = librosa.stft(x)
Xdb = librosa.amplitude_to_db(abs(X))
plt.figure(figsize=(14, 5))
librosa.display.specshow(Xdb, sr=sr, x_axis='time', y_axis='hz')
plt.colorbar()
```

```
Out[50]: <matplotlib.colorbar.Colorbar at 0x2483d8b9438>
```

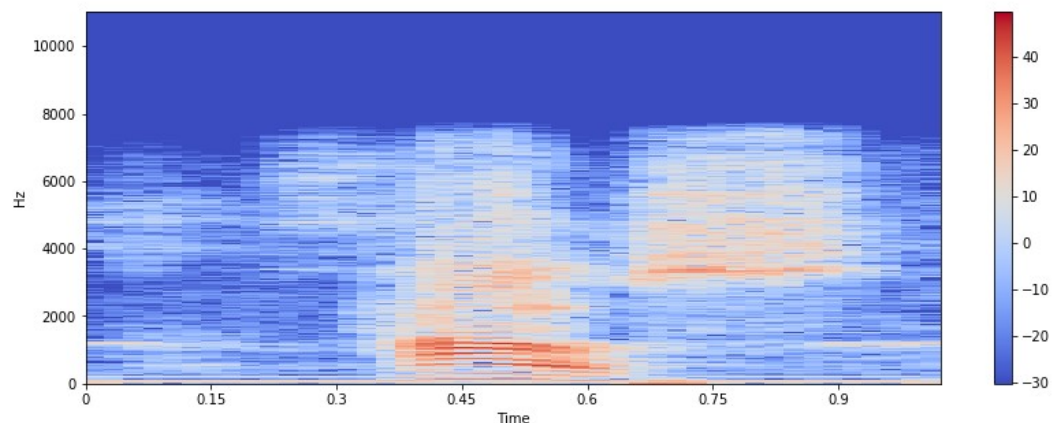


Figura 9 Espectrograma de un audio

El eje vertical muestra frecuencias (de 0 a 10 kHz), y el eje horizontal muestra el tiempo del clip.

### 3.4.5. Extracción de Características

Cada señal de audio consta de muchas características o parámetros. Se debe extraer las características que sean relevantes al objetivo del proyecto. A continuación se mencionan algunas de ellas.

#### 3.4.5.1. Zero Crossing Rate

El “zero crossing rate” es la tasa de cambios de signos a lo largo de una señal de audio, es decir, la tasa a la que la señal cambia de positivo a negativo o de regreso. Este atributo es uno de los comúnmente utilizados para el reconocimiento de voz pero también en muchos casos es utilizado en la recuperación de información musical. En las imágenes siguientes se puede ver el resultado aplicado al clip de audio *stop.wav*

```
In [9]: # Load the audio
x, sr = librosa.load('stop.wav')
#Plot the audio:
plt.figure(figsize=(14, 5))
librosa.display.waveplot(x, sr=sr)

Out[9]: <matplotlib.collections.PolyCollection at 0x13e1e2053c8>
```

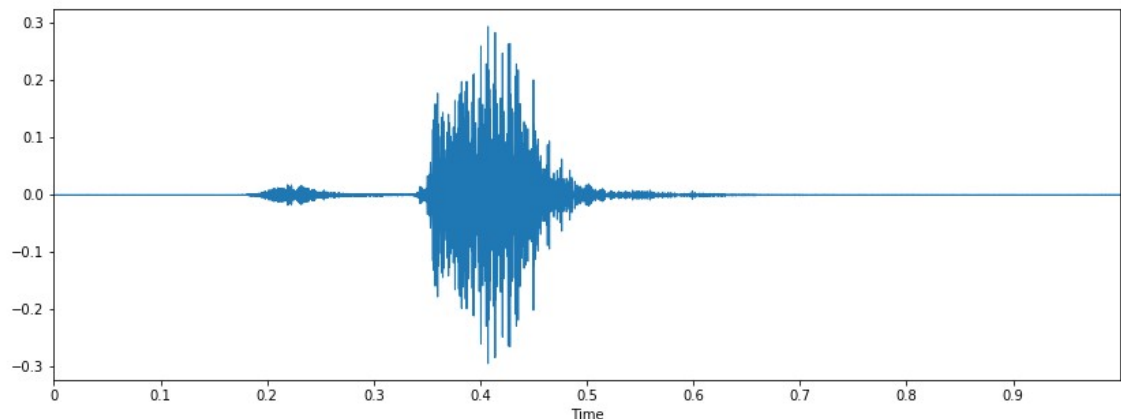
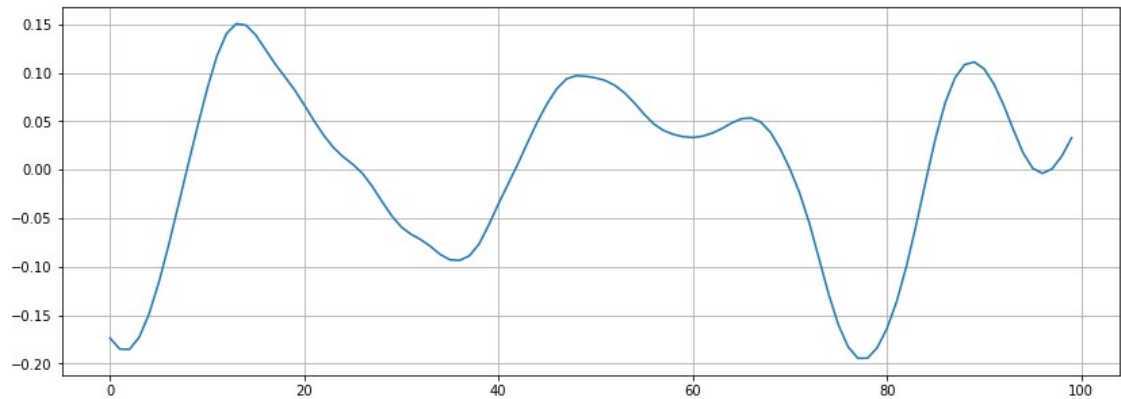


Figura 10 Zero Crossing Rate

```
In [10]: # Zooming in
n0 = 9000
n1 = 9100
plt.figure(figsize=(14, 5))
plt.plot(x[n0:n1])
plt.grid()
```



Tenemos 7 zero crossing para el audio ejemplo

```
In [13]: zero_crossings = librosa.zero_crossings(x[n0:n1], pad=False)
print(sum(zero_crossings))
```

7

### 3.4.5.2. Spectral Centroid

Indica dónde se encuentra el centro del espectro para un sonido y se calcula como la media ponderada de las frecuencias presentes en el sonido.

```
In [16]: import sklearn
spectral_centroids = librosa.feature.spectral_centroid(x, sr=sr)[0]
spectral_centroids.shape
```

```
Out[16]: (44,)
```

```
In [18]: # Computing the time variable for visualization
frames = range(len(spectral_centroids))
t = librosa.frames_to_time(frames)

# Normalising the spectral centroid for visualisation

def normalize(x, axis=0):
    return sklearn.preprocessing.minmax_scale(x, axis=axis)

#Plotting the Spectral Centroid along the waveform
librosa.display.waveplot(x, sr=sr, alpha=0.4)
plt.plot(t, normalize(spectral_centroids), color='r')
```

```
Out[18]: [<matplotlib.lines.Line2D at 0x13e1f939be0>]
```

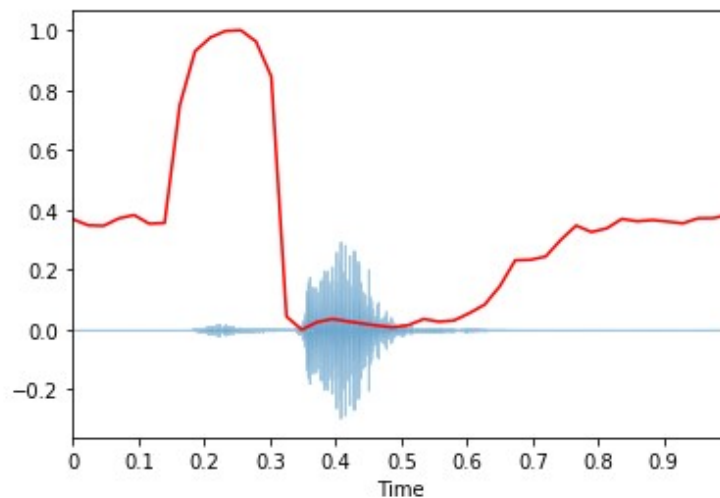


Figura 11 Spectral Centroid

### 3.4.5.3. Spectral Rolloff

Es una medida de la forma de la señal. Representa la frecuencia por debajo de la cual un porcentaje especificado de la energía espectral total.



```
In [23]: spectral_rolloff = librosa.feature.spectral_rolloff(x+0.01, sr=sr)[0]
librosa.display.waveplot(x, sr=sr, alpha=0.4)
plt.plot(t, normalize(spectral_rolloff), color='g')
```

```
Out[23]: [<matplotlib.lines.Line2D at 0x13e20a4b978>]
```

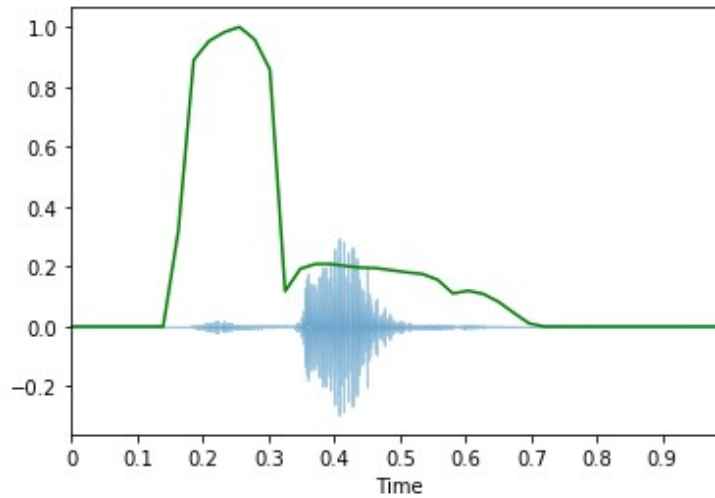


Figura 12 Spectral Rolloff

#### 3.4.5.4. Mel-Frequency Ceptral Coefficients

Más conocidos como los MFCCs, los coeficientes cepstrales de frecuencia Mel de una señal son un pequeño conjunto de características alrededor de 10 a 20 que describen de manera concisa la forma general de una envoltura espectral. Modela las características de la voz humana.

```
In [24]: x, fs = librosa.load('stop.wav')
librosa.display.waveplot(x, sr=sr)
```

Out[24]: <matplotlib.collections.PolyCollection at 0x13e21abe0b8>

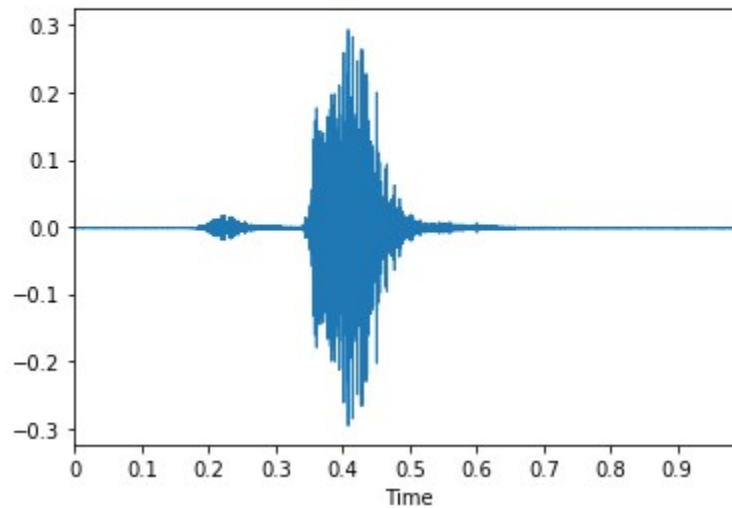


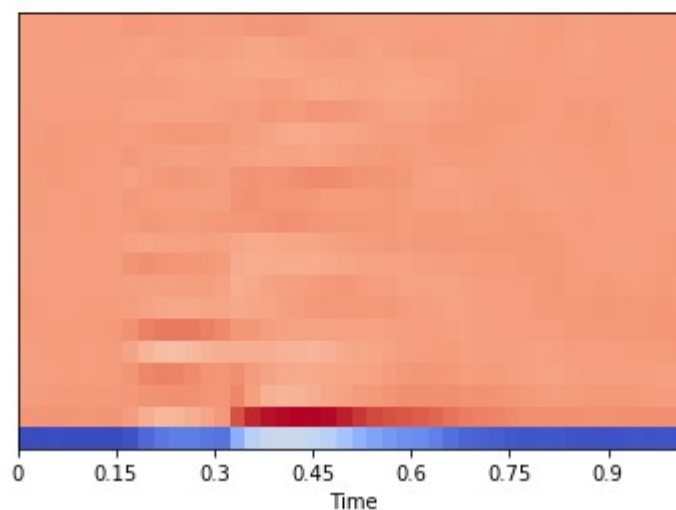
Figura 13 MFCC de un audio

```
In [28]: mfccs = librosa.feature.mfcc(x, sr=fs)
print (mfccs.shape)

#Displaying the MFCCs:
librosa.display.specshow(mfccs, sr=sr, x_axis='time')
```

(20, 44)

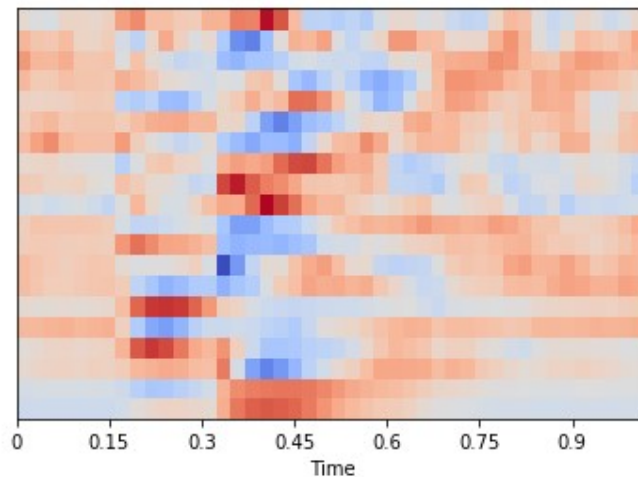
Out[28]: <matplotlib.axes.\_subplots.AxesSubplot at 0x13e21e41da0>



```
In [29]: import sklearn
mfccs = sklearn.preprocessing.scale(mfccs, axis=1)
print(mfccs.mean(axis=1))
print(mfccs.var(axis=1))
librosa.display.specshow(mfccs, sr=sr, x_axis='time')

[ 3.79302278e-08 -4.06395317e-09  1.35465106e-09 -4.19941806e-08
 -2.16744169e-08 -7.58604557e-08  1.62558127e-08  4.06395317e-09
  1.08372085e-08 -6.77325529e-09 -1.21918591e-08  8.12790635e-09
 -5.41860423e-09 -2.70930212e-08 -5.41860423e-09  1.35465106e-08
 -2.53997068e-09  3.92848811e-08  1.35465106e-08 -3.21729621e-09]
[0.99999976 0.9999998  1.0000002  0.99999994 1.          1.
 1.0000001  1.          1.0000001  0.99999994 0.9999998  0.99999964
 1.0000004  1.0000001  1.0000002  1.0000001  1.          1.0000001
 0.9999998  1.          ]
```

```
Out[29]: <matplotlib.axes._subplots.AxesSubplot at 0x13e21e9c240>
```



### 3.4.5.5. Chroma Frequencies

Los atributos de croma son una representación interesante para el audio especialmente para la música en la que todo el espectro se proyecta en 12 contenedores que representan los 12 semitonos (o croma) distintos de la octava musical.

```
Out[30]: <matplotlib.axes._subplots.AxesSubplot at 0x13e21eec128>
```

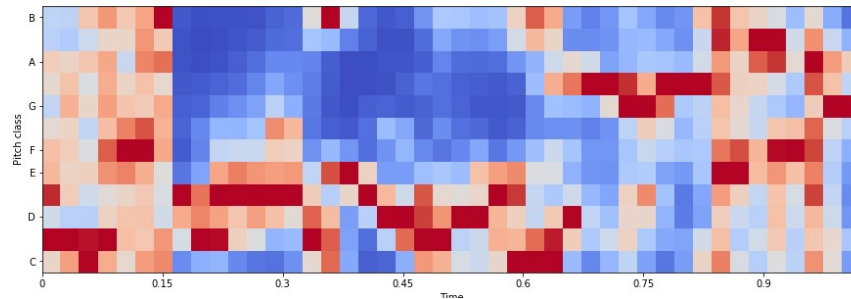


Figura 14 Chroma Frequency

## 4 CAPITULO IV

### RECONOCIMINETO DE COMANDOS DE VOZ

#### 4.1. INTRODUCCION



Figura 15 Proceso de reconocimiento de voz

Luego de tener una visión general sobre las señales acústicas y el proceso de extracción de características, en esta sección se utilizará los conocimientos adquiridos para poder resolver el problema de reconocimiento de voz/comandos con machine learning.

La propuesta es desarrollar un modelo de machine learning usando redes neuronales, antes de que el dataset sea procesado por la red neuronal, este debe pasar por una fase pre-procesado, dado que la red no puede trabajar con los archivos de audio como entradas.

Al momento de preprocesar los archivos de audio se pueden optar por dos enfoques

- Obtener los espectrogramas (ya descritos anteriormente) y diseñar una red que trabaje con imágenes como entradas.
- Obtener los características y tratarlos como vectores de números y usar esos como entradas de la red neuronal.

En el presente proyecto se optó por el segundo enfoque. El procesamiento de audio requiere una serie de operaciones a realizar de forma secuencial en cada uno de los archivos de audio que

permita extraer los atributos o características relevantes que en su conjunto compondrán el dataset.

## 4.2. DATASET

Durante el diseño de un modelo de Machine Learning es fundamental elegir correctamente el dataset a utilizar ya que esto influye directamente en los resultados obtenidos.

En el caso de procesamiento de lenguaje natural NLP los dataset son llamados corpus por la relación que tienen con la lingüística.

En función del objetivo del proyecto el tipo de dataset a escoger ha sido diferente de los normalmente trabajados, en este caso tenemos un conjunto de archivos de audio/corpus en idioma inglés en formato WAV

El audioset ha sido descargado de la página *Open Speech and Language Resources*<sup>2</sup> el identificador del audioset es el SLR45<sup>3</sup>



*Figura 16 Fuente del audioset*

El audioset seleccionado contiene 30 carpetas, cada una de ellas representa una palabra y dentro se encuentran los archivos de audios que corresponden a dicha palabra.

Cada carpeta en promedio contiene 2300 archivos de audio por lo que se tendrá un total de 69000 archivos en promedio para trabajar.

<sup>2</sup> Open Speech and Languages Resources o más conocido como OpenSLR es una página web que contiene un sin fin de audiosets para libre descarga.

<sup>3</sup> Dentro la página OpenSLR cada audio set tiene un identificador SLR45 es el que corresponde al utilizado en el presente proyecto.

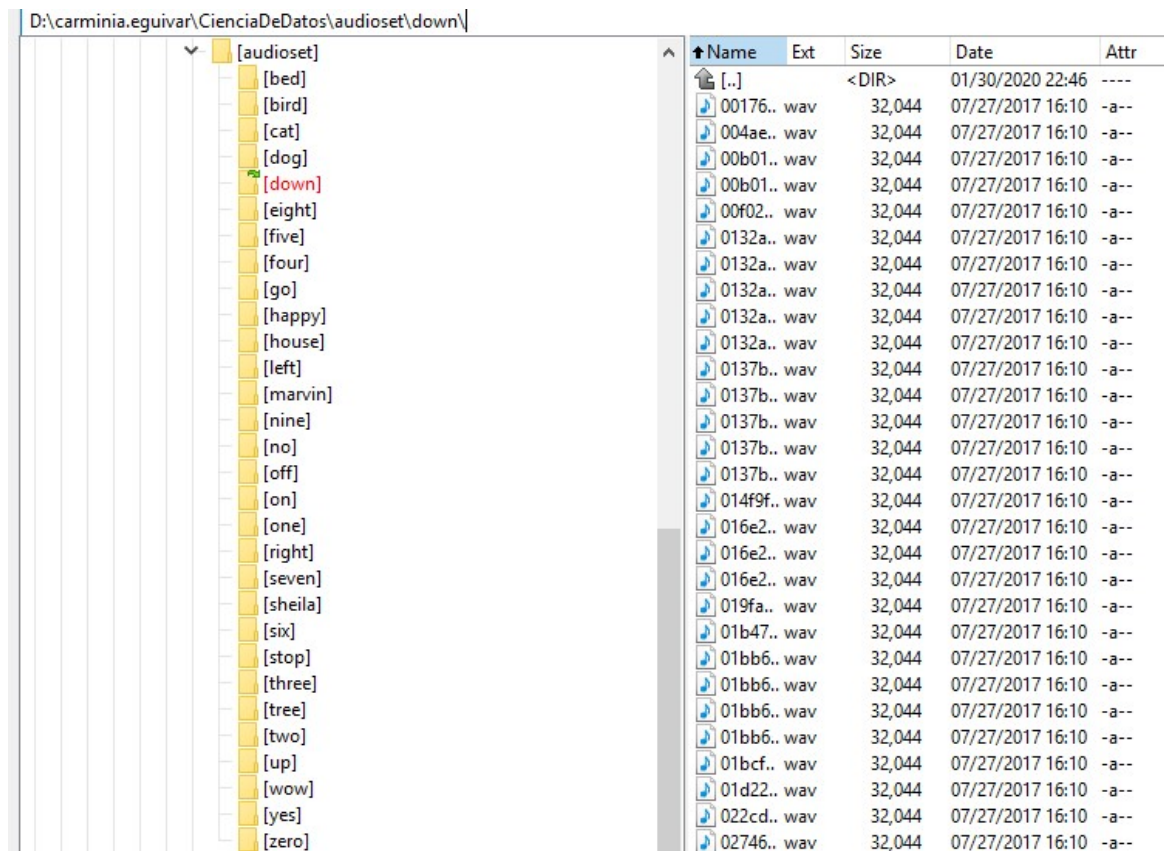


Figura 17 Estructura de carpetas del audioset

### 4.3. PRE-PROCESAMIENTO DE ARCHIVOS DE AUDIO

Antes de entrenar el modelo de clasificación, se transformaron las muestras de audio en representaciones más significativas/relevantes y de interés para el objetivo planteado. Los archivos de audio necesitan un procesamiento previo para que puedan ser usados como entradas de la red neuronal. Se realizó este pre-procesamiento con librerías python para leer archivos de audio. (Librerías ya mencionadas en detalle en el capítulo anterior)

Se realizó la extracción de los atributos más significativos de los archivos de audio. Se han elegido 5 atributos: coeficientes cepstrales de frecuencia de mel, centroide espectral, velocidad de cruce cero, frecuencias de croma, roll-off espectral.

#### 4.3.1. Extracción de espectrograma por cada audio

Como ya se había mencionado en el capítulo anterior un espectrograma es la representación visual acerca de qué frecuencias integran un sonido y cuáles son las respectivas amplitudes y fases, constituye lo que se denomina espectro del sonido.



En resumen el espectrograma extraído por cada audio es una imagen con extensión .png que será guardada en disco. Se crearan nuevas carpetas replicando la misma estructura del audioset para guardar las imágenes/espectrogramas.

```
In [37]: |
plt.figure(figsize=(10,10))

commands = 'bed bird cat dog down eight five four go happy house left marvin nine'

for g in commands:
    pathlib.Path(f'img_data/{g}').mkdir(parents=True, exist_ok=True)
    for filename in os.listdir(f'audioset/{g}'):
        songname = f'audioset/{g}/{filename}'
        y, sr = librosa.load(songname, mono=True, duration=5)
        plt.specgram(y, NFFT=2048, Fs=2, Fc=0, noverlap=128, cmap=cmap, sides='d')
        plt.axis('off');
        plt.savefig(f'img_data/{g}/{filename[:-3].replace(".", "")}.png')
        plt.clf()
```

Figura 18 Extraccion de espectrogramas

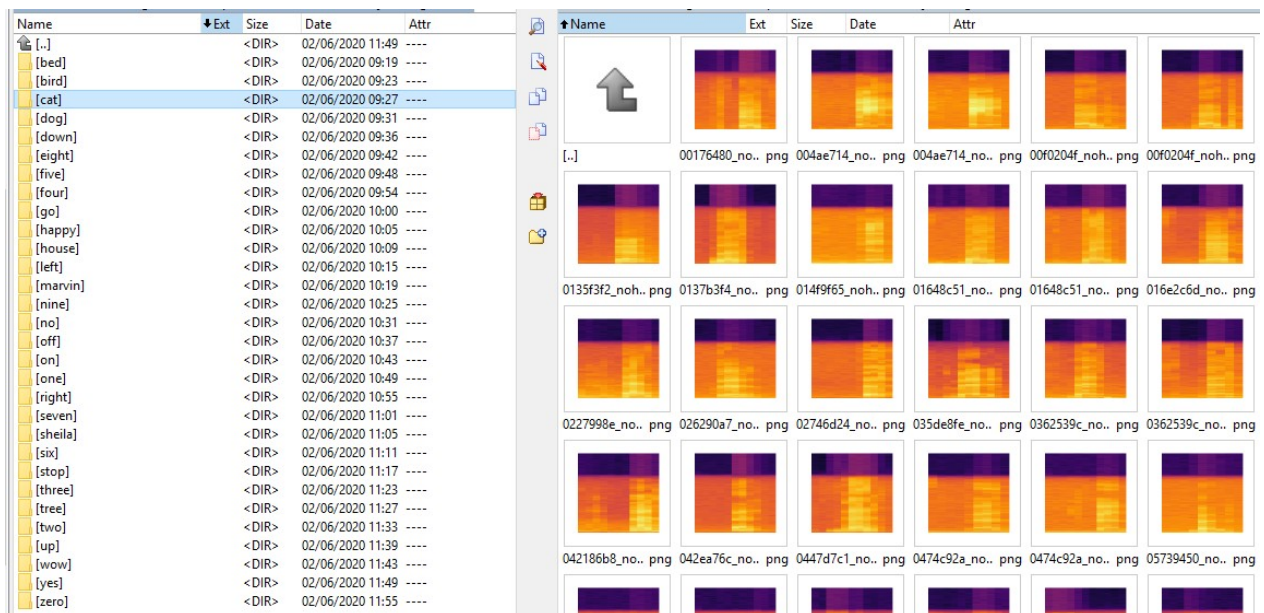


Figura 19 Ejemplo de los espectrogramas generados en el proyecto

#### 4.3.1. Extraccion de Caracteristicas desde los espectrogramas

Los atributos a extraer por cada espectrograma(imagen) serán los siguientes:

- Coeficientes cepstrales de frecuencia de mel (MFCC) (en un número de 20)

- Centroide espectral,
- Tasa de cruce cero
- Frecuencias de croma
- Desplazamiento espectral.

```
In [*]: file = open('dataset.csv', 'w', newline='')
with file:
    writer = csv.writer(file)
    writer.writerow(header)
commands = 'bed bird cat dog down eight five four go happy house left marvin nine'
for g in commands:
    for filename in os.listdir(f'audioaset/{g}'):
        songname = f'audioaset/{g}/{filename}'
        y, sr = librosa.load(songname, mono=True, duration=30)
        chroma_stft = librosa.feature.chroma_stft(y=y, sr=sr)
        spec_cent = librosa.feature.spectral_centroid(y=y, sr=sr)
        spec_bw = librosa.feature.spectral_bandwidth(y=y, sr=sr)
        rolloff = librosa.feature.spectral_rolloff(y=y, sr=sr)
        zcr = librosa.feature.zero_crossing_rate(y)
        mfcc = librosa.feature.mfcc(y=y, sr=sr)
        rmse = librosa.feature.rms(y=y)[0]
        to_append = f'{filename} {np.mean(chroma_stft)} {np.mean(rmse)} {np.mean(
        for e in mfcc:
            to_append += f' {np.mean(e)}'
        to_append += f' {g}'
        file = open('dataset.csv', 'a', newline='')
        with file:
            writer = csv.writer(file)
            writer.writerow(to_append.split())
```

Figura 20 Extracción de características a partir de un mfcc

#### 4.4. ENSAMBLADO DEL DATASET

Una vez realizadas la extracción de atributos del archivo de audio , estas características se añaden a un archivo .csv y el conjunto de todos los archivos de audio procesados componen el dataset que luego será utilizado por algoritmo de clasificación en este caso la red neuronal.



```

In [*]: file = open('dataset.csv', 'w', newline='')
with file:
    writer = csv.writer(file)
    writer.writerow(header)
commands = 'bed bird cat dog down eight five four go happy house left marvin nine'
for g in commands:
    for filename in os.listdir(f'audioaset/{g}'):
        songname = f'audioaset/{g}/{filename}'
        y, sr = librosa.load(songname, mono=True, duration=30)
        chroma_stft = librosa.feature.chroma_stft(y=y, sr=sr)
        spec_cent = librosa.feature.spectral_centroid(y=y, sr=sr)
        spec_bw = librosa.feature.spectral_bandwidth(y=y, sr=sr)
        rolloff = librosa.feature.spectral_rolloff(y=y, sr=sr)
        zcr = librosa.feature.zero_crossing_rate(y)
        mfcc = librosa.feature.mfcc(y=y, sr=sr)
        rmse = librosa.feature.rms(y=y)[0]
        to_append = f'{filename} {np.mean(chroma_stft)} {np.mean(rmse)} {np.mean'
        for e in mfcc:
            to_append += f' {np.mean(e)}'
        to_append += f' {g}'
        file = open('dataset.csv', 'a', newline='')
        with file:
            writer = csv.writer(file)
            writer.writerow(to_append.split())

```

Figura 21 Ensamblado del dataset

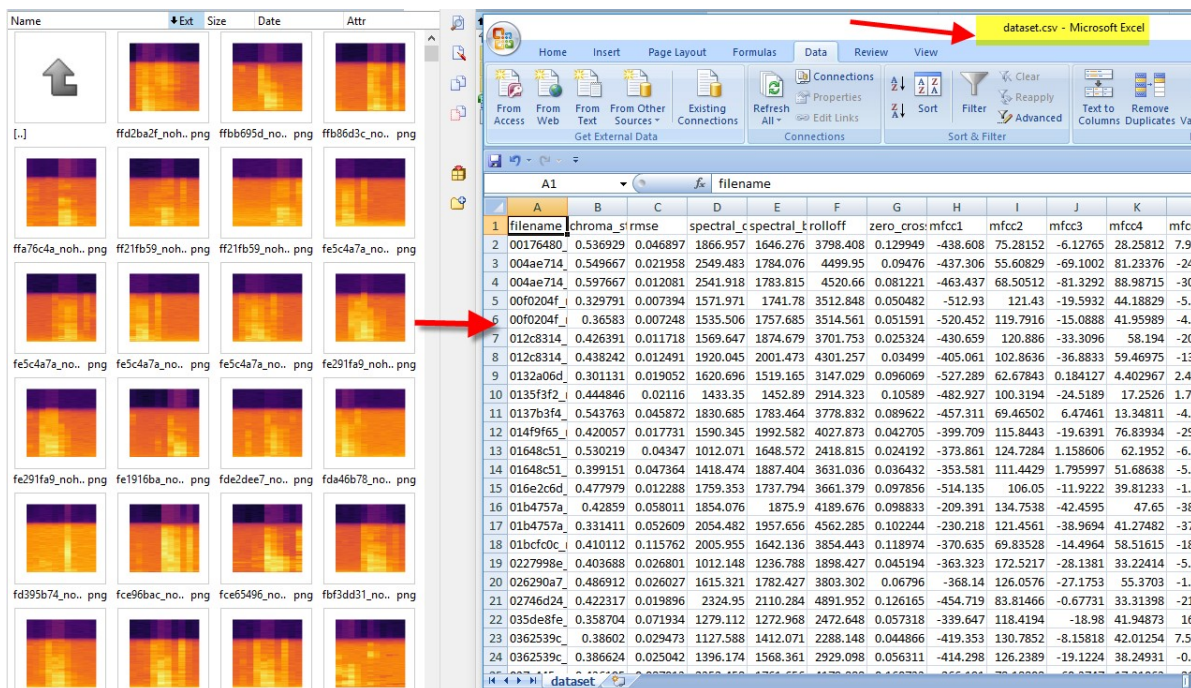


Figura 22 Ejemplo de dataset generado en el proyecto

## 4.5. ANALIZANDO LOS DATOS

```
In [40]: data = pd.read_csv('dataset.csv')
data.head()
```

Out[40]:

	filename	chroma_stft	rmse	spectral_centroid	spectral_bar
0	00176480_nohash_0.wav	0.536929	0.046897	1866.957312	1646.
1	004ae714_nohash_0.wav	0.549667	0.021958	2549.482652	1784.
2	004ae714_nohash_1.wav	0.597667	0.012081	2541.917660	1783.
3	00f0204f_nohash_0.wav	0.329791	0.007394	1571.970794	1741.
4	00f0204f_nohash_1.wav	0.365830	0.007248	1535.506265	1757.

5 rows × 28 columns



```
In [41]: data.shape
```

Out[41]: (64721, 28)

Figura 23 Dataset desde pandas

```
In [45]: fig, ax = plt.subplots()
fig.set_size_inches(14, 6)
sns.countplot('label', data=data, saturation=5)
```

Out[45]: <matplotlib.axes.\_subplots.AxesSubplot at 0x23f780369e8>

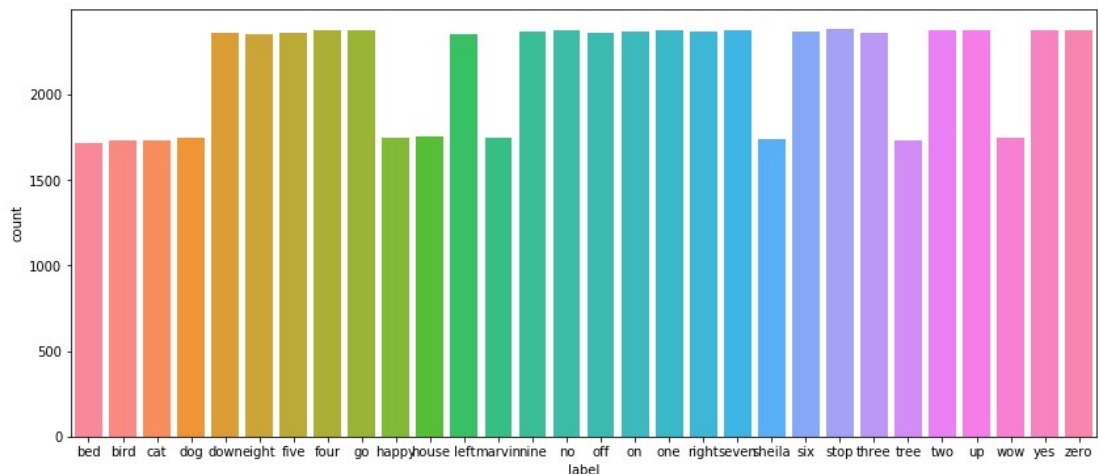


Figura 24 Analisis del dataset

### 4.5.1. Limpieza de datos

- Eliminamos columnas innecesarias

```
In [54]: data = data.drop(['filename'],axis=1)
```

- Codificación de las etiquetas

```
In [43]: genre_list = data.iloc[:, -1]
encoder = LabelEncoder()
y = encoder.fit_transform(genre_list)
```

- Escalando las columnas

```
In [44]: scaler = StandardScaler()
X = scaler.fit_transform(np.array(data.iloc[:, :-1], dtype = float))
```

### 4.6. DIVIDIENDO EL DATASET EN TRAINING<sup>4</sup> AND TEST<sup>5</sup> DATA

```
In [45]: X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)
```

```
In [46]: len(y_train)
```

```
Out[46]: 51776
```

```
In [47]: len(y_test)
```

```
Out[47]: 12945
```

```
In [48]: #X_train[10]
X_train[0]
```

```
Out[48]: array([-0.54877684, -0.16316277,  0.42369381,  0.52406989,  0.53882632,
  0.23199028,  0.55612635,  0.11477286,  0.35761167,  0.64838676,
  0.24231778,  0.78574946, -0.65810765, -0.63737408, -0.14829235,
  0.19134811, -0.61230269,  0.09347443, -0.25362202, -0.42513681,
  0.82691093,  0.14644634, -0.28383726, -0.84061739,  1.26039126,
  0.59737763])
```

Figura 25 Dividiendo el dataset para el entrenamiento y test

<sup>4</sup> Training Data: En un aprendizaje supervisado, utiliza un conjunto de datos de entrenamiento, que contiene resultados, para entrenar la máquina.

<sup>5</sup> Test Data: Luego del entrenamiento se utiliza un conjunto de datos de prueba para predecir los resultados.



## 4.7. CLASIFICACION CON KERAS- CONSTRUCCION DE LA REDE NEURONAL

Una vez que se han extraído las características y se tiene ensamblado/definido el dataset, se procede a usar los algoritmos de clasificación existentes para reconocimiento de voz en el presente proyecto se usa redes neuronales. Como anteriormente se ha mencionado, de acuerdo a las investigaciones realizadas se pueden usar las imágenes del espectrograma directamente para la clasificación o puede extraer las características y usar los modelos de clasificación en ellas. Se ha elegido usar la extracción de características y utilizar esas como entradas de la red neuronal.

Se crea el modelo tipo secuencial y las capas de neuronas secuenciales.

```
In [77]: from keras import models
        from keras import layers

        snn_model = models.Sequential()

        snn_model.add(layers.Dense(512, activation='relu', input_shape=(X_train.shape[1],)))

        snn_model.add(layers.Dense(256, activation='relu'))

        snn_model.add(layers.Dense(128, activation='relu'))

        snn_model.add(layers.Dense(64, activation='relu'))

        snn_model.add(layers.Dense(30, activation='softmax'))
```

```
In [78]: snn_model.summary()
```

Model: "sequential\_1"

Layer (type)	Output Shape	Param #
dense_1 (Dense)	(None, 512)	13824
dense_2 (Dense)	(None, 256)	131328
dense_3 (Dense)	(None, 128)	32896
dense_4 (Dense)	(None, 64)	8256
dense_5 (Dense)	(None, 30)	1950
Total params: 188,254		
Trainable params: 188,254		
Non-trainable params: 0		

Figura 26 Resumen del modelo construido

### 4.7.1. Ajustes al modelo (Compilando el modelo)

Con esto indicamos el tipo de pérdida (loss) que se utilizara ,el optimizador de los pesos de las conexiones de las neuronas y las métricas que queremos obtener

```
In [53]: model.compile(optimizer='adam',
                        loss='sparse_categorical_crossentropy',
                        metrics=['accuracy'])
```

Figura 27 Compilando el modelo

## 4.8. ENTRENAMIENTO DE LA RED

Indicamos con `model.fit()` las entradas y sus salidas y la cantidad de iteraciones de aprendizaje (epochs) de entrenamiento.

```
In [83]: tic=time.time()
snn = snn_model.fit(X_train,
                    y_train,
                    epochs= 1000, #150
                    batch_size= 500
                    )
print('Tiempo: {} secs'.format(time.time()-tic))
```

Figura 28 Entrenando la red neuronal

### 4.8.1. Interpretacion de los resultados del entrenamiento

Si vemos las salidas del entrenamiento, vemos que la primer iteración tuvo un acierto de 0.2 , es decir sólo acierta 1 de cada 5 .

```
Epoch 1/1000
45304/45304 [=====] - 1s 24us/step - loss: 2.7657 - accuracy: 0.2018
Epoch 2/1000
45304/45304 [=====] - 1s 17us/step - loss: 2.2954 - accuracy: 0.3233
Epoch 3/1000
45304/45304 [=====] - 1s 18us/step - loss: 2.1709 - accuracy: 0.3573 0s - loss: 2
Epoch 4/1000
45304/45304 [=====] - 1s 18us/step - loss: 2.0883 - accuracy: 0.3770
Epoch 5/1000
```

Figura 29 Epoch numero 1

Luego en la epoch<sup>6</sup> 50 recupera el 0.7 de acierto ,es decir 7 de cada 10.

<sup>6</sup> Este es el numero de veces que se ejecutaran los algoritmos de forwardpropagation y backpropagation. En cada ciclo (epoch) todos los datos de entrenamiento pasan por la red neuronal para que esta aprenda sobre ellos



```
In [101]: # make class predictions with the model
x_pred = snn_model.predict_classes(X_test)

x_pred = encoder.inverse_transform(x_pred)
y_pred = encoder.inverse_transform(y_test)

# summarize the first 100 cases
for i in range(100):
    print('%s => %s (expected %s)' % (X_test[i].tolist(), x_pred[i], y_pred[i]))
```

Figura 33 Predicciones con los datos de test

## 4.8.2. Analisis de Resultados

Si vemos las parte resaltadas en amarillo observamos los resultados de las predicciones :

```
[ -1.103976242733239, 1.1098193738052478, -0.49934372538350935, 0.31280913041575353, -0.10357979450683114, -0.758616886229401
9, 0.7671319835954744, -0.11235142118426675, 0.2923406556031782, -1.1397032911159553, -0.49708620787664265, 0.076504794325991
9, 2.1487276803086917, 1.1362209851592961, 0.15551366012671938, 0.4224252161080699, 0.17875673085099614, -0.8864751873407066,
-1.0514019333166464, -0.8591377199382693, -0.7744116937577388, -0.5305056791856629, -1.2471506712252158, -0.684250307713425,
-1.0971554228956046, -0.2120000263587316] => off (expected off) 1
[ -1.1348215719458112, 1.991219939208961, -0.9477373914023414, -0.15580107770657428, -0.9357766072599776, -0.7551486050365752,
0.6295887928699188, -0.10577634435262251, 2.1331127956544895, -0.09254899951001612, 1.0676941903451653, -1.0955189674451127,
-0.4672269506829913, 0.16037946171973586, -0.08258486397060165, -0.4153952086612222, -0.2927538224359316, 1.441502037690228,
-0.07628109601745442, -0.012053934277995946, -1.237791927089387, -0.10166389729901594, -0.2521407779115703, -0.69706819377649
83, 0.5673192139270342, -0.7026245382717505] => no (expected on) 2
[ 0.17269262097094626, -0.855265218899529, 1.7340619670105382, 0.9439583591873723, 1.3138660829239028, 1.0358057404400394, -0.
7734698151083526, -1.2384961376450419, -0.8702841744588082, 0.9208673004724659, -0.690824652003845, 0.5987666137318671, -1.31
5831320782436, 1.1821401782150753, 0.47183647711464255, 0.87991065177702, 0.5752418794805502, -0.22275588043475145, 0.8448759
180569445, 1.1632938173099778, 0.6473324173653652, -1.0154214149734715, 1.2169972474181785, -0.48719249067752146, 0.524355839
5812511, 0.45723716452374724] => seven (expected seven) 3
[ -0.09218435399416139, -0.7804646445493546, 2.5944796414729154, 0.5170401432121575, 1.8229057535366286, 2.4388080610619354, -
0.44007312671027515, -1.9002561712102297, -1.5620123386168867, 0.8069338758282717, -1.4568572221520424, 0.2457030578516077, -
2.0010993293333668, 0.8979352779307855, 0.10308487549390458, 0.12564040664637205, 0.4410028352500414, -0.03712755107663539, -
0.13570003045124907, -0.5692158179906909, -0.023944166382149634, -1.3179621217476356, 1.0055276691523198, -2.062527285864916
```

Figura 34 Analisis de resultados

1 => off (expected off) : El resultado esperado era la palabra *off* el prededido fue *off* lo cual es correcto.

2 => no (expected on) : El resultado esperado era la palabra *on* y el prededicon fue *no* est preddiccion fue incorrecta.

3 => seven (expected seven) : El resultado esperado era la palabra *seven* el prededido fue *seven* lo cual es correcto.

Si evaluamos el modelo con los datos de test obtenemos que la precisión es de un 42.79 , lo cual explica el por que de nuestras predicciones incorrectas. Se puedeir mejorando el modelo para subir esta precisión.

```
In [99]: results_test = snn_model.evaluate(X_test, y_test)
print(results_test)
print("%s: %.2f%%" % (snn_model.metrics_names[1], results_test[1]*100))

19417/19417 [=====] - 0s 22us/step
[6.817297667070012, 0.4278724789619446]
accuracy: 42.79%
```



## 5 CONCLUSIONES Y RECOMENDACIONES

- Haciendo uso del reconocimiento de voz se puede reducir la brecha que existe en la comunicación entre una persona y la maquina.
- A pesar que en la evaluacion para en entrenamiento se optuvo una precision del 99.9% se pudo observar que en la etapa de test la precision solo era del 42% Por tanto aun no se ha explotado al maximo el potencial de las redes neuronales, por lo que se espera mejorar esta precision en iteraciones futuras.
- Contrario a la inicialmente previsto,la fase mas compleja del presente proyecto no ha sido el diseño y entrenamiento de la red neuronal sino la elecion y el pre-proceso de audio por lo que ha sido necesaio incluir este proceso como parte de los objetivos y alcances del proyecto. Para cumplir esta etapa satisfactoriamente la fase de investigacion ha sido extensa en cuanto a la tecnologia, librerias, audioset, etc.



## 6 BIBLIOGRAFIA

- [1] SciPy (2015) Talk & Tutorial Video - Recuperado el 31-01-2020 de <https://scipy2015.scipy.org/>
- [2] FFMPEG () Recuperado el 31-01-2020 de <https://www.ffmpeg.org/about.html>
- [3] Docs google (2015) Basic Sound processing in Python - Recuperado el 29-01-2020 de [https://docs.google.com/presentation/d/1zzgNu\\_HbKL2iPkHS8-qhtDV20QfWt9IC3ZwPVZo8Rw0/pub?start=false&loop=false&delayms=3000&slide=id.p](https://docs.google.com/presentation/d/1zzgNu_HbKL2iPkHS8-qhtDV20QfWt9IC3ZwPVZo8Rw0/pub?start=false&loop=false&delayms=3000&slide=id.p)
- [4] Open SLR (2018) Recuperado el 02-02-2020 de <http://www.openslr.org/45/>
- [5] Vincent Blog (2018) Conceptos Basicos Sobre redes Neuronales - Recuperado el 02-02-2020 de <https://vincentblog.xyz/posts/conceptos-basicos-sobre-redes-neuronales>
- [6] Machine Learning Mastery(2019) Your First Deep Learning Project in Python with Keras Step-By-Step - Recuperado el 04-02-2020 de <https://machinelearningmastery.com/tutorial-first-neural-network-python-keras/>
- [7] Towards (2019) From raw images to real-time predictions with Deep Learning- Recuperado el 31-01-2020 de <https://towardsdatascience.com/from-raw-images-to-real-time-predictions-with-deep-learning-ddbda1be0e4>
- [8] Siri, Apple Inc (2019) Recuperado el 31-01-2020 de <https://www.apple.com/es/ios/siri/>
- [9] Microsoft Corp(2019) Tu asistente virtual y personal inteligente, Recuperado el 31-01-2020 de <https://www.microsoft.com/es-es/windows/cortana>
- [10] Adroid() Dragon Mobile Assistan - Recuperado el 31-01-2019 de <http://www.dragonmobileapps.com/android/>
- [11] Universidad Carlos III de Madrid(2017) Detección de estrés en señales de voz – Recupero el 07-02-2019 de <https://e-archivo.uc3m.es/handle/10016/27535>
- [12] Universidad Carlos III de Madrid(2019) Procesamiento de Audio con Técnicas de Inteligencia Artificial Recuperado el 07-02-2019 de <https://e-archivo.uc3m.es/handle/10016/29348>
- [13] Universidad Politecnica de Valencia (2017) Desarrollo y análisis de clasificadores de señales de audio Recuperado el 08-02-2019 de <https://www.semanticscholar.org/paper/Desarrollo-y-an%C3%A1lisis-de-clasificadores-de-se%C3%B1ales-Mart%C3%ADn/d29353d72309a9c30b407c167c4812f541660147>