



Desarrollo



Clasificación de sonidos respiratorios



Conjunto de datos

Composición


El modelo será entrenado y probado con dos distintos subconjuntos de datos del conjunto de datos llamado “Respiratory Sound Database”, proveído por Kaggle y conformado por novecientas veinte grabaciones de longitud variable entre un rango de diez y noventa segundos.

Información de los archivos .WAV

Dentro del conjunto de datos, hay otros elementos que también serán utilizados, por ejemplo, un archivo de texto en formato .CSV listando los diagnósticos para cada paciente

< **patient_diagnosis.csv** (1.46 KB)

Detail Compact Column

# 101	U URTI
	COPD 51%
	Healthy 21%
	Other (35) 28%
102	Healthy
103	Asthma
104	COPD
105	URTI
106	COPD

Desarrollo

Bibliotecas necesarias

```
from os import listdir, getcwd
from os.path import isfile, join

import librosa
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import seaborn as sn
from sklearn.metrics import classification_report, confusion_matrix
from sklearn.model_selection import train_test_split
from tensorflow.keras import regularizers, optimizers
from tensorflow.keras.layers import Dense, Conv1D, Flatten, Activation, MaxPooling1D, Dropout
from tensorflow.keras.models import Sequential, Model
from tensorflow.keras.utils import plot_model, to_categorical
```

Figura 1: Celda de un Jupyter Notebook con las importaciones de librerías y asignaciones de alias.

Entorno de desarrollo

Debido a los requerimientos de hardware necesarios para desarrollar y realizar las pruebas de evaluación sobre el modelo, se usará la infraestructura que Google puede proveer a través de Colaboratory. La primera tarea a realizar será subir el conjunto de datos en el Google Drive donde se ubica la instancia y después de asignar los permisos necesarios para montar los datos del Google Drive como un disco virtual.

Unidad virtual

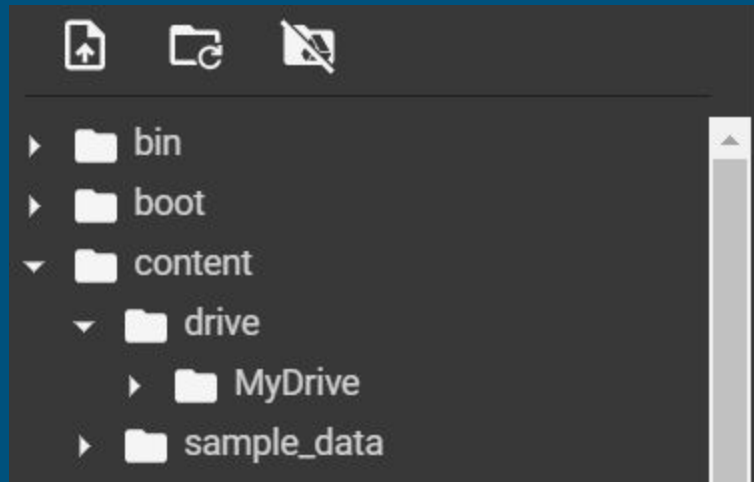


Figura 2: Ejemplo de unidad virtual montada.

Navegar a través del conjunto de datos

Fuera de cualquier función se va a declarar una variable con el directorio donde se ubican los archivos de audio.

```
files_path = "../content/drive/MyDrive/Universidad/8vo cuatrimestre/Sistemas inteligentes/
```

Figura 3: Variable declarada fuera de funciones con la ruta de los audios.

```
class Diagnostico():
    """Representar un diagnostico, con la ubicación del audio y su diagnostico.

    Args:
        id: Identificador entero.
        diagnosis: Cadena que es el diagnóstico del archivo de audio.
        path: Cadena concatenada que es la ruta del archivo .wav
    """

    def __init__(self, id, diagnosis, path):
        self.id = id
        self.diagnosis = diagnosis
        self.path = path
```

Figura 4: Clase encargada de modelar un diagnóstico.

La siguiente función tiene como propósito listar todos los archivos en formato .WAV de un directorio, en este caso la dirección del directorio se le asignó en forma de parámetro.

```
files = [f for f in listdir(files_path) if isfile(join(files_path, f))]  
wav_files = [f for f in files if f.endswith('.wav')]  
  
sorted_files = sorted(wav_files)  
return sorted_files
```

Figura 5: Compresiones de listas para obtener los archivos .wav de un directorio.

La última función de esta fase es la de vincular la información correspondiente para cada diagnóstico.

```
diagnosis = pd.read_csv("../content/drive/MyDrive/Universidad/8vo cuatrimestre/  
wav_files = get_audio_files(files_path)  
diag_dict = {101: "URTI"}  
diagnosis_list = []  
  
for index, row in diagnosis.iterrows():  
    diag_dict[row[0]] = row[1]  
  
id = 0  
for f in wav_files:  
    diagnosis_list.append(Diagnosis(id, diag_dict[int(f[:3])],  
                                   join(files_path, f)))  
    id += 1  
  
return diagnosis_list
```

Figura 6: Cuerpo de la función que crea instancias de la clase declarada previamente.

Extracción de características

Se deben extraer las características que son relevantes para el problema que estamos intentando resolver.

```
sound, sample_rate = librosa.load(filename)
stft = np.abs(librosa.stft(sound))
mfccs = np.mean(librosa.feature.mfcc(y=sound, sr=sample_rate, n_mfcc=40), axis=1)
chroma = np.mean(librosa.feature.chroma_stft(S=stft, sr=sample_rate), axis=1)
mel = np.mean(librosa.feature.melspectrogram(sound, sr=sample_rate), axis=1)
contrast = np.mean(librosa.feature.spectral_contrast(S=stft, sr=sample_rate), axis=1)
tonnetz = np.mean(librosa.feature.tonnetz(y=librosa.effects.harmonic(sound), sr=sample_rate), axis=1)

features = np.concatenate((mfccs, chroma, mel, contrast, tonnetz))
```

Figura 7: Extracción de características y obtención de un valor numérico.

La siguiente función va a separar los puntos de datos, entre los que serán las variables de entrada y las variables de salida.

```
images = []
labels = []
to_hot_one = {"COPD": 0, "Healthy": 1, "URTI": 2, "Bronchiectasis": 3, "Pneumonia": 4, "Bronchiolitis": 5, "Asthma": 6, "LRTI": 7}

for f in diagnosis_data(files_path):
    labels.append(to_hot_one[f.diagnosis])
    images.append(get_audio_features(f.path))

return np.array(labels), np.array(images)
```

Figura 8: Cuerpo de función que obtiene los puntos de datos y los retorna en un arreglo.

Pre-procesamiento

Se van a eliminar los datos que tienen información con muy poca representación en el conjunto de datos, en este caso serían los diagnósticos de asma y la infección del tracto respiratorio inferior.

```
def preprocessing(labels, images):  
    images = np.delete(images, np.where((labels == 7) | (labels == 6))[0], axis=0)  
    labels = np.delete(labels, np.where((labels == 7) | (labels == 6))[0], axis=0)
```

Figura 9: Función que elimina datos poco representativos en los puntos de datos.

Para entrenar y medir el rendimiento predictivo del modelo es necesario dividir el conjunto de datos en un conjunto de entrenamiento y en otro de prueba.

```
x_train, x_test, y_train, y_test = train_test_split(images, labels, test_size=0.2, random_state=10)
```

Figura 10. División de los datos en cuatro sub-conjuntos.

Ahora vamos a darle una nueva forma a los arreglos sin cambiar su información.

```
y_train = np.reshape(y_train, (y_train.shape[0], 6))  
x_train = np.reshape(x_train, (x_train.shape[0], x_train.shape[1], 1))  
y_test = np.reshape(y_test, (y_test.shape[0], 6))  
x_test = np.reshape(x_test, (x_test.shape[0], x_train.shape[1], 1))
```

Figura 11. Cambiar la forma de los arreglos.

Red neuronal convolucional

```
model = Sequential()

model.add(Conv1D(64, kernel_size=5, activation='relu', input_shape=(193, 1)))
model.add(Conv1D(128, kernel_size=5, activation='relu'))
model.add(MaxPooling1D(2))
model.add(Conv1D(256, kernel_size=5, activation='relu'))
model.add(Dropout(0.3))
model.add(Flatten())
model.add(Dense(512, activation='relu'))
model.add(Dense(6, activation='softmax'))

model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
history = model.fit(x_train, y_train, validation_data=(x_test, y_test), epochs=70, batch_size=200, verbose=1)
```

Figura 12. Capas de la red neuronal convolucional.

Evaluación

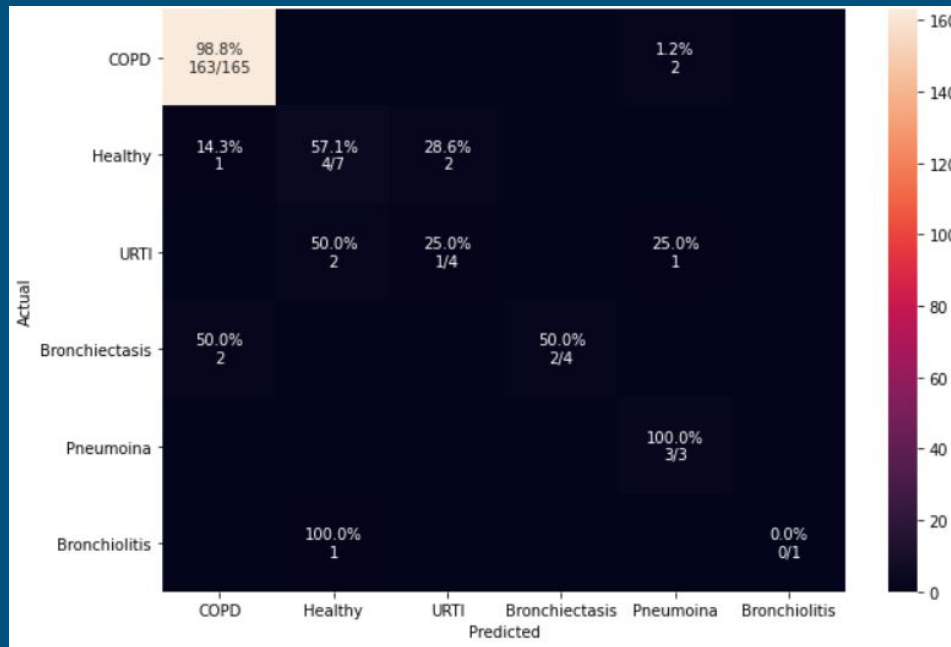


Figura 13. Matriz de confusión