UNIVERSIDAD POLITÉCNICA DE VICTORIA

Ingeniería en tecnologías de la información

Clasificación de sonidos respiratorios

Materia: Sistemas inteligentes

Grupo: ITI-27627

Alumno:

Balderas Briones Ricardo

Delarbre Quintanilla Mario Alberto

Flores Infante Sergio Gerardo

Loperena Bustillos Gerardo

Soto Gomez Juan Ramiro

Villanueva Marquez Francisco Javier

Docente:

Polanco Martagón Said

Enero - Abril 2020. Ciudad Victoria, Tamaulipas.

UNIVERSIDAD POLITÉCNICA DE VICTORIA

Índice

Índice **2**

Definición del problema **3**

Estado del arte **4**

Propuesta de solución **6**

Librerías necesarias 6

Extracción de características 7

Pre-procesamiento 8

Red neuronal convolucional 8

Evaluación 9

Desarrollo **10**

Entorno de desarrollo 10

Google Colaboratory 10

Convenciones 10

Navegando en el conjunto de datos 11

Extracción de características 14

Pre-procesamiento 15

Red neuronal convolucional 16

Evaluación 16

Conclusiones **17**

Bibliografía **18**

Definición del problema

El objetivo es desarrollar un modelo de clasificación basado en una red neuronal convolucional tiene como objetivo que a partir de las distintas características extraídas de múltiples audios respiratorios en formato .WAV sea capaz mediante algoritmos de clasificación determinar si un sonido respiratorio presenta alguna enfermedad o está sano. Se partirá desde los sonidos respiratorios debido a que son indicadores importantes de la salud y de las enfermedades respiratorias, debido a que el sonido emitido cuando una persona respira está directamente relacionado con los cambios dentro del tejido pulmonar y la posición de las secreciones dentro del pulmón. Por ejemplo, un sonido jadeante es un indicador común de que el paciente sufre una enfermedad obstructiva de las vías respiratorias como el asma o una enfermedad pulmonar obstructiva crónica.

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, donde se incluyen sonidos limpios de respiración y grabaciones ruidosas que simular condiciones de la vida real.

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 y se descartaran los archivos que listen información no relevante, como los datos demográficos del paciente debido a que escapa a la naturaleza del problema y no aporta a la resolución del objetivo planteado previamente.

102	Healthy
103	Asthma
104	COPD
105	URTI
106	COPD

Figura 1. Primeras filas del archivo “patient_diagnosis.csv”.

Estado del arte

Python es un lenguaje interpretado con una sintaxis expresiva que a menudo es comparada con pseudocódigo, por lo que poco a poco se ha ido estableciendo como uno de los lenguajes más populares para la computación científica y en la industria, gracias a su naturaleza interactiva y a su maduro ecosistema de librerías científicas (Python: Batteries Included, 2007, 21)

Las redes convolucionales tienen sus orígenes en la década de los ochenta, pero han sido recientemente adaptadas como un método de elección para varias tareas de clasificación de objetos, comúnmente de imágenes pero también de sonidos ambientales o géneros (Piczak, 2015, 1). La definición de una red convolucional es un modelo de tipo de red con pesos restringidos. Hay dos tipos de restricciones, la primera es conocida como localidad que implica que cada unidad en la capa i está conectada a un grupo de unidades en la capa $i - 1$ que es local, la segunda restricción llamada invarianza de traducción donde cada unidad en la capa $i - 1$ está conectado a una unidad en la capa i con la misma configuración de peso (Dieleman et al., 2011, 671).

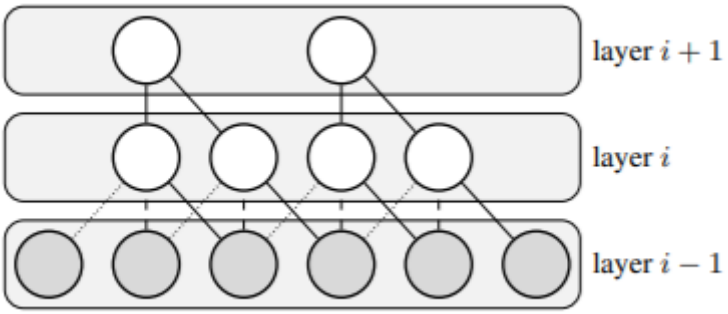

Figura 2. Representación visual de las conexiones que tienen las unidades entre capas (Dieleman et al., 2011, 671).

Una de las complicaciones más grandes para entrenar arquitecturas neuronales profundas de una manera supervisada es la cantidad de esfuerzo computacional e información etiquetada requerida para un aprendizaje eficiente. Desafortunadamente, hay muy pocos conjuntos de datos disponibles públicamente

no son tan comunes debido a los altos recursos que requieren la anotación de manera manual.

Uno de los primeros experimentos en probar la eficacia de las redes neuronales convolucionales en tareas relacionadas a la clasificación de sonidos se realizó sobre un conjunto de datos de sonidos ambientales, este experimento muestra que un modelo convolucional alcanza un nivel similar al de otros métodos de aprendizaje de características. Por lo que muestra que las redes neuronales convolucionales se pueden aplicar de manera efectiva en tareas de clasificación de sonido ambiental incluso con conjuntos de datos limitados aunque el resultado no es innovador debido a los tiempos de entrenamiento necesarios (Piczak, 2015, 6).

El trabajo con mayor implementación y tangible a nivel de usuarios finales relacionado a la clasificación de sonidos es, sin duda, el trabajo producido por las aplicaciones de música bajo demanda que cuentan con un servicio personalizado de recomendaciones musicales, como Spotify. En el caso de este último, específicamente aplica técnicas de aprendizaje automático sobre los datos que la plataforma puede recopilar como la música que otros usuarios con un gusto similar escuchan, pero la parte importante es la extracción de características numéricas sobre contenido de las canciones, para descubrir si una canción es feliz o triste, el tempo que es la velocidad en la que una pieza musical es reproducida (Zimmer, 2020) u otras que se enfoquen en los elementos acústicos (Lazzaro, 2019).

Propuesta de solución

Librerías necesarias

La única librería incluida dentro de Python será “os”, que provee una manera portable de realizar funciones dependientes del sistema operativo (Python Software Foundation, 2021), más específicamente de esta librería solo importamos algunos métodos específicos relacionados al manejo o verificación de archivos.

Posteriormente del salto de línea se encuentran las bibliotecas desarrolladas por terceros que tendrán que ser satisfechas para el correcto funcionamiento del proyecto. A continuación se desglosa una explicación general de los módulos necesarios:

- Pandas: Herramienta de código abierto que nos permitirá explorar, limpiar y procesar información (McKinney & Pandas Development Team, 2021, 3).
- Librosa: Tiene como propósito principal realizar análisis de música y audio, proporciona los componentes básicos necesarios para crear sistemas de recuperación de información musical (McFee et al., 2020, 1).
- Matplotlib: Es una herramienta completa que permite crear visualizaciones estáticas, animadas e interactivas en Python (Hunter, n.d., 90).
- Numpy: Es la librería de Python para la computación científica, provee utilidades relacionadas a la lógica, transformadas de Fourier además de álgebra lineal y operaciones estadísticas básicas (The SciPy community, 2021).
- Tensorflow: Plataforma de código abierto de extremo a extremo que permite desarrollar y entrenar modelos de aprendizaje automático (*TensorFlow*, n.d.).
- Seaborn: Permite realizar gráficas estadísticas en Python, está construido sobre Matplotlib y se integra de manera muy cercana con las estructuras de datos que provee Pandas
- Sklearn: Integra un amplio rango de algoritmos de aprendizaje automático para problemas de mediana escala supervisados y no supervisados. Su objetivo principal es dar un acceso al aprendizaje automático a personas no

especializadas usando un lenguaje de programación de alto nivel de propósito general (Pedregosa et al., 2011, 2826).

```
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 3. Celda de un Jupyter Notebook con las importaciones de librerías y asignaciones de alias.

Extracción de características

Antes de describir algunas características utilizadas, es necesario definir qué es un espectrograma, es la representación gráfica del espectro de frecuencias de la emisión sonora.

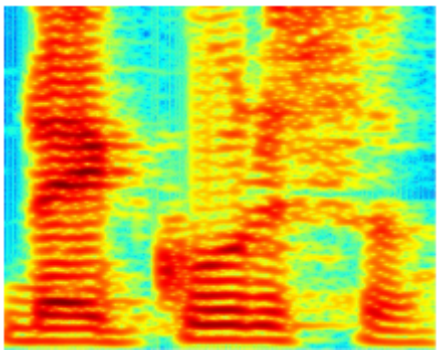

Figura 4. Ejemplo de un espectrograma en escala de colores.

El espectrograma puede revelar rasgos, como altas frecuencias o modulaciones de amplitud, que no pueden apreciarse incluso aunque estén dentro de los límites de frecuencia del oído humano. Algunas de las características extraídas para este problema son:

- Coeficientes Cepstrales en las Frecuencias de Mel: Hablando de una señal, son un conjunto de alrededor de diez a veinte características que describen de manera concisa la forma en general de un envolvente espectral. Han sido las características dominantes utilizadas para el reconocimiento de voz durante algún tiempo (Young et al., 1993, 5).
- Espectrogramas en escala Mel: Es un espectrograma con la escala Mel como su eje y (Gartzman, 2019).
- Frecuencias de croma: Representación de un audio en donde el espectro entero es proyectado en doce contenedores que representan doce semitonos (o croma) distintos del intervalo musical octava.

Pre-procesamiento

Antes de dividir el conjunto de datos, se tendrá que eliminar los diagnósticos con muy poca representación, en este caso, los diagnosticados con asma e infección del tracto respiratorio inferior.

Posteriormente, se van a dividir los conjuntos de entrenamiento y de pruebas, el primer conjunto va a contener alrededor del ochenta por ciento de los datos debido a que la fase de entrenamiento es la que más información requiere y el segundo será utilizado para medir la exactitud de las predicciones generadas por el modelo.

Red neuronal convolucional

Analizando los distintos modelos disponibles y basado en los trabajos científicos generados, se optó por un modelo secuencial debido a la naturaleza del problema que solo requiere una entrada y una salida correspondiente, que será generado y entrenado mediante los métodos que la librería Keras provee.

Con los dos conjuntos de datos definidos previamente, se utilizará el método “fit()” para entrenar el modelo, los argumentos principales y con más margen de experimentación son “batch_size” que dividirá los datos en lotes de cierto tamaño y el valor de “epoch” determinará las veces que el modelo tendrá que iterar sobre el conjunto de pruebas para calcular las métricas de validación.

UNIVERSIDAD POLITÉCNICA DE VICTORIA

Evaluación

El objetivo del modelo es evitar estar sobre-ajustado que es cuando funciona muy bien con los datos de entrenamiento, pero, su precisión disminuye cuando se utiliza con otros datos, por lo que no tendrá una aplicación real en el mundo real. También se busca evitar que esté desajustado, que es cuando ni siquiera puede funcionar correctamente con los datos de entrenamiento por lo que no puede realizar predicciones precisas.

También debido a que es un problema de clasificación, se usará una matriz de clasificación, que es una herramienta que permite la visualización del desempeño de un algoritmo que se emplea en aprendizaje supervisado. Donde cada columna representa el número de predicciones de cada clase, mientras que cada fila representa a las instancias en la clase real.

Desarrollo

Entorno de desarrollo

Google Colaboratory

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.

Debido a este enfoque, 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 para poder acceder a su contenido, igual a que si estuviéramos trabajando de manera local.

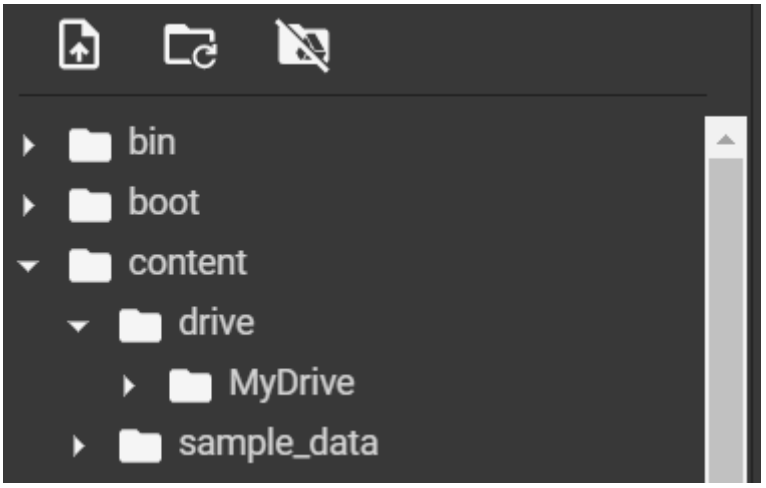

Figura 5: Carpeta “content” que representa el contenido de nuestro Google Drive como si fuera local.

Convenciones

Para cada función o clase necesaria, se colocará un “docstring” compuesto de la primera línea que va a resumir a grosso modo la función, luego un párrafo con más detalles de la implementación y posteriormente, según sea necesario, una lista y descripción de los argumentos de entrada y los valores retornados.

Este “docstring” es el atributo especial llamado “__doc__” del objeto que se declaró (Python Software Foundation, 2001).

Navegando en el conjunto de datos

Fuera de cualquier función, se va a declarar una variable que contenga una cadena de texto que será la ubicación de la carpeta donde se encuentra el contenido del conjunto de datos.

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

Figura 6:. Primera celda de código que contiene una variable visible en todo el programa.

Las clases son usadas para crear estructuras de datos definidas por el usuario y las funciones que definan son llamadas métodos que identifican los comportamientos y acciones que un objeto creado por la clase puede realizar (Amos, 2020). La siguiente clase va a modelar un diagnóstico del conjunto de datos, solo va a contar con el método especial “__init__” para la posterior inicialización de las instancias.

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

    Args:
        id: Identificador entero.
        diagnosis: Cadena que es el diagnositco 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 7: Clase “Diagnosis” con el método especial “__init__” y los atributos correspondientes.

Los argumentos serán “self” que representa a la instancia de la clase, posteriormente contará con un identificador numérico de tipo entero, el diagnóstico y la dirección del archivo de audio correspondientes.

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. Posteriormente, se hace uso de notación de conjuntos para crear dos compresiones de listas.

La primera compresión llamada “files” se va a encargar de listar todo el contenido del directorio, en este caso es directamente desde la unidad virtual montada y agregar en una lista el contenido si es un archivo. La segunda compresión llamada “wav_files” hace uso de la lista anterior que contiene todos los archivos del directorio y los agrega solo si son archivos en formato .WAV. Finalmente se acomodan los archivos en forma alfabética y se retorna esta lista.

```
def get_audio_files(files_path):
    """Retornar una lista de todos los archivos de audio del conjunto de datos.

    Se lista el directorio de "file_path" y se agrega a una lista todos
    aquellos que seana archivos, posteriormente, para cada archivo se verifica
    si son .WAV y se agrega a una lista.

    Args:
      files_path: Directorio donde se encuentran los archivos del conjunto de
        datos.

    Returns:
      sorted_files: Lista ordenada alfabeticamente de los archivos .wav.
    """
    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 8: Función “get_audio_files()” que contiene dos comprensiones de listas y retorna una tupla de valores.

Las dos comprensiones de listas se podrán resumir en una sola, pero debido a la naturaleza de la notación de conjuntos, se optó por adoptar el enfoque más sencillo y separar las dos tareas en dos listas diferentes.

Finalmente, la última función de esta fase es la de vincular la información correspondiente para cada diagnóstico. Lo primero en esta función es leer un archivo .CSV contenido en el conjunto de datos donde tiene la información de los diagnósticos de cada paciente, esto se va a lograr mediante una función proveída por “pandas” debido a la gran flexibilidad de uso que brinda.

```
def diagnosis_data(files_path):
    """Crear lista de las instancias de la clase Diagnosis con los atributos.

    Crear diccionario de los diagnosticos a partir de  'patient_diagnosis.csv',
    asignarlo a cada archivo obtenido de get_audio_files() mediante una
    instancia de 'Diagnosis' y retornar todas estás instancias en una lista.

    Args:
      files_path: Directorio donde se encuentran los archivos del conjunto de
      datos.

    Returns:
      diagnosis_list: Lista de las instancias de la clase Diagnosis con sus
      atributos
    """
    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 9: Función “diagnosis_data()” que carga la información del conjunto de diagnósticos y la agrega en una lista que es retornada.

En el primer ciclo “for” gracias a la versatilidad de la función para leer un archivo .CSV podemos obtener una secuencia de datos que contiene todos los valores dentro de una fila del archivo, estos valores serán asignados dentro de un diccionario.

Finalmente, en el último ciclo se agregan distintas instancias de la clase “Diagnosis” previamente declarada en una lista y se agregan los valores de los atributos de los elementos. El segundo atributo de la clase previamente mencionada se asigna mediante el valor de la llave de un diccionario, “f” representa una cadena con el nombre un archivo de audio y sus tres primeros caracteres, son el identificador correspondiente a la primera columna del archivo .csv.

Extracción de características

Se deben extraer las características que son relevantes para el problema que estamos intentando resolver. El proceso de extracción de características para usarlas en análisis es llamada extracción de características, para este paso se tiene que definir una función llamada “get_audio_features()” con un argumento.

```
def get_audio_features(filename):
    """Extraer las características de un audio y las concatena en un arreglo.

    Cargar el audio con Librosa para posteriormente calcular el valor absoluto
    o el promedio de los datos números de la caracteristica correspondiente.

    Args:
      filename: La ruta relativa donde se encuentra el archivo de audio.

    Returns:
      features: Unión de una secuencia de arreglos de las características,
    """
    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))
    return features
```

Figura 10: Método que extrae las características de un audio y las retorna en un nuevo arreglo.

La última función de esta sección es la encargada de generar las etiquetas e imágenes, para eso hace uso del valor retornado de la función “diagnosis_data()” que es una lista de instancias de la clase “Diagnosis()”, finalmente retorna dos arreglos generados mediante “numpy”, que son dos puntos de datos, lo que significa que son un conjunto de una o más mediciones en un solo miembro de la unidad de observación, donde “labels” es el valor que se quiere predecir en pasos posteriores.

```
def data_points(files_path):
    """Generar  arreglos con el id del diagnostico y características del audio.

    Crear dos ejes, uno que agregue un identificador personalizado de cada
    diagnostico y uno con todas las características de cada archivo.

    Args:
      files_path: Directorio donde se encuentran los archivos del conjunto de
      datos.

    Returns:
      np.array(labels): Lista de identificador de los diagnosticos de cada
      archivo.
      np.array(images): Lista de las características de audio de cada archivo.
    """
  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 11: Función encargada de generar los ejes necesarios.

Pre-procesamiento

Primero, 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 12: Eliminar diagnósticos con poca representación en el conjunto 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. El primero es utilizado por el modelo para aprender de él y poder realizar predicciones, el segundo es para evaluar el rendimiento de su predicción.

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

Figura 13: Creación de los conjuntos de entrenamiento y de prueba.

Red neuronal convolucional

Con la ayuda de Keras, crearemos un modelo secuencial que es el más adecuado debido a que no contamos con múltiples entradas ni con múltiples salidas. Procederemos a inicializarlo y entrenarlo con los datos definidos previamente dividiendo los datos en "lotes" de tamaño "batch_size" y repetidamente iterando sobre todo el conjunto de datos para un número determinado de "épocas".

```
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 14: Entrenamiento del modelo secuencial.

Evaluación

A partir del conjunto de pruebas, se realizó la prueba al modelo previamente creado, donde la función usada retornará el valor de pérdida y los valores de las métricas para el modelo en modo de prueba, este cálculo es realizado en lotes, determinado por el usuario, en este caso de serán sesenta elementos de estos conjuntos en cada iteración.

```
score = model.evaluate(x_test, y_test, batch_size=60, verbose=0)
print('Accuracy: {0:.0%}'.format(score[1]/1))
print("Loss: %.4f\n" % score[0])
```

```
Accuracy: 96%
Loss: 0.1635
```

Figura 15: Entrenamiento del modelo secuencial.

Dando como resultado una precisión del noventa y seis por ciento.

Matriz de confusión

Debido a que es un problema de clasificación, se usará una matriz de clasificación, que es una herramienta que permite la visualización del desempeño de un algoritmo que se emplea en aprendizaje supervisado. Donde cada columna representa el número de predicciones de cada clase, mientras que cada fila representa a las instancias en la clase real.

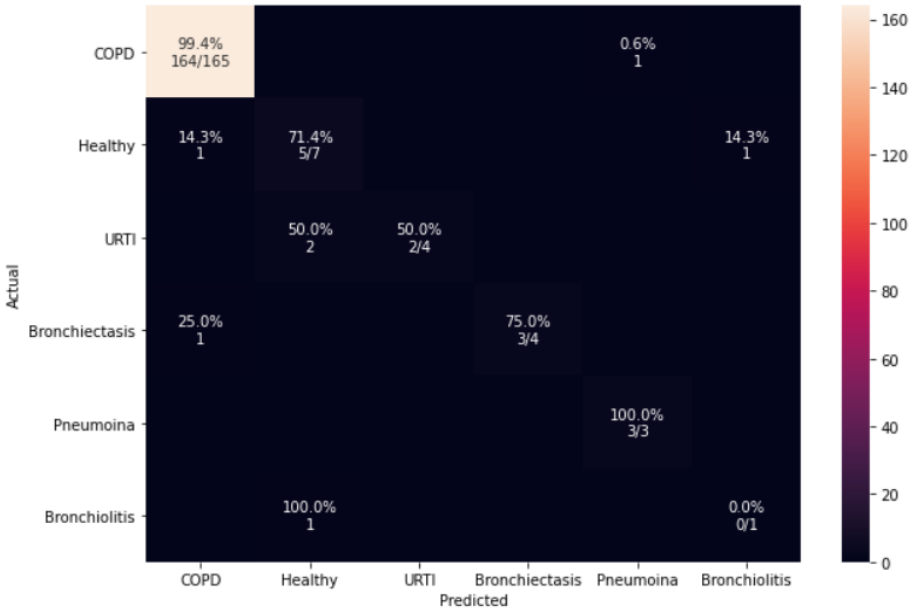

Figura 16: Matriz de confusión a partir de los resultados del modelo.

Conclusiones

El objetivo del modelo es evitar estar sobre-ajustado que es cuando funciona muy bien con los datos de entrenamiento, pero, que su precisión disminuye cuando se utiliza con otros datos, por lo que no tendrá una aplicación real en el mundo real. También se buscó evitar que esté desajustado, que es cuando ni siquiera puede funcionar correctamente con los datos de entrenamiento por lo que no puede realizar predicciones precisas.

Lamentablemente, con el conjunto de datos actual y las limitaciones tecnológicas, no es posible dar un diagnóstico que se considere acertado solo basándose en los sonidos respiratorios de los pacientes, debido a que el conjunto de enfermedades respiratorias comparten muchas similitudes en cuestión de síntomas. Además el papel de la clínica en este tipo de diagnósticos es muy importante complementando la información con otros elementos, como podría ser el de una radiografía.

Bibliografía

Amos, D. (2020, 07 06). *Object-Oriented Programming (OOP) in Python 3*. Real Python. Retrieved 02 09, 2021, from https://realpython.com/python3-object-oriented-programming/

Dieleman, S., Brakel, P., & Schrauwen, B. (2011). Audio-based Music Classification with a Pretrained Convolutional Network. In *ISMR*. Electronics and Information Systems department, Ghent University.

Gartzman, D. (2019, 08 19). *Getting to Know the Mel Spectrogram*. Towards Data Science. Retrieved 02 09, 2021, from https://towardsdatascience.com/getting-to-know-the-mel-spectrogram-31bca3e2d9d0

Hunter, J. (n.d.). Matplotlib: A 2D graphics environment. *Computing in Science & Engineering*, 90-95.

Lazzaro, S. (2019, 04 05). *Spotify's Head of Deep Learning Reveals How AI Is Changing the Music Industry*. The Observer. Retrieved 02 08, 2021, from https://observer.com/2015/05/spotifys-head-of-deep-learning-reveals-how-ai-is-changing-the-music-industry/

McFee, B., Lostanlen, V., Metsai, A., McVicar, M., Balke, S., Thomé, C., Raffel, C., Zalkow, F., Malek, A., Dana, Lee, K., Nieto, O., Mason, J., Ellis, D., Battenberg, E., Seyfarth, S., Yamamoto, R., Choi, K., viktorandreevichmorozov, … Kim, T. (2020). *librosa/librosa: 0.8.0* (0.8.0 ed.). Zenodo. 10.5281/zenodo.3955228

McKinney, W., & Pandas Development Team. (2021). *pandas: powerful Python data analysis toolkit*. https://pandas.pydata.org/docs/pandas.pdf

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., & Duchesnay, E. (2011). Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research*. http://jmlr.org/papers/v12/pedregosa11a.html

Piczak, K. J. (2015). Environmental sound classification with convolutional neural networks. In *2015 IEEE 25th International Workshop on Machine Learning for*

Signal Processing (MLSP) (pp. 1-6). Institute of Electronic Systems Warsaw University of Technology. 10.1109/MLSP.2015.7324337

Python: Batteries Included. (2007). In P. F. Dubois (Ed.), *Computing in Science & Engineering* (Vol. 9, pp. 1-29). IEEE/AIP.

Python Software Foundation. (2001, 05 29). *PEP 257 -- Docstring Conventions*. Python. Retrieved 03 02, 2021, from https://www.python.org/dev/peps/pep-0257/

Python Software Foundation. (2021, 02 09). *os — Miscellaneous operating system interfaces*. Generic Operating System Services. Retrieved 02 09, 2021, from https://docs.python.org/3/library/os.html

The SciPy community. (2021, 01 31). *What is NumPy?* User Guide. Retrieved 02 09, 2021, from https://numpy.org/doc/stable/user/whatisnumpy.html

TensorFlow. (n.d.). TensorFlow. Retrieved 02 09, 2021, from https://www.tensorflow.org/

Young, S. J., Woodland, P. C., & Byrne, W. J. (1993). *HTK: Hidden Markov Model Toolkit V1.5*. Cambridge University Engineering Department Speech Group and Entropic Research Laboratories Inc.

Zimmer, H. (2020, 11 08). *Music 101: What Is Tempo? How Is Tempo Used in Music?* MasterClass. Retrieved 08 02, 2021, from https://www.masterclass.com/articles/music-101-what-is-tempo-how-is-tempo-used-in-music#what-is-tempo