1. TIPO DE DOCUMENTO: Trabajo de grado para optar el título de INGENIERO DE SONIDO.

2. TÍTULO: Desarrollo e implementación de un algoritmo basado en Deep Learning para la identificación de señales de audio de arma de fuego.

3. AUTOR (ES): Ojeda Santana Daniel Felipe; Ochoa Rincón Juan Sebastián.

4. LUGAR: Bogotá, D.C

5. FECHA: Febrero del 2022

6. PALABRAS CLAVES: Inteligencia artificial, Deep Learning, Redes neuronales artificiales, Redes Neuronales Convolucionales, Capas Convolucionales, Armas de fuego.

7. DESCRIPCIÓN DEL TRABAJO: Este proyecto tiene como fin la interpretación de algoritmos basados en Deep Learning y redes convolucionales en el lenguaje de programación Python para la identificación de disparos de armas de fuego con respecto a otros sonidos impulsivos y no impulsivos.

8. LÍNEA DE INVESTIGACIÓN: Aplicaciones sonoras para la sociedad

9. METODOLOGÍA: Esta investigación tiene un enfoque cuantitativo secuencial y probatorio, ya que comienza a partir de una idea general que se convierte en un problema delimitado y concreto.

10. CONCLUSIONES: En este proyecto se evidencia la posibilidad de identificar sonidos de disparos de armas de fuego mediante la implementación de redes convolucionales, mediante las cuales se puede obtener una precisión superior a un 70% con respecto a otros sonidos impulsivos y una precisión mayor a un 90%, ya que existe una gran diferencia en los cálculos de los Coeficientes de Mel, porque al no ser sonidos impulsivo, los valores en los coeficientes son diferentes ya que dependen de la frecuencia y el tiempo.

Desarrollo e implementación de un algoritmo basado en Deep Learning para la identificación de señales de audio de arma de fuego.

Juan Sebastián Ochoa Rincón

Daniel Felipe Ojeda Santana

Universidad de San Buenaventura, Sede Bogotá.

Facultad de Ingeniería.

Programa de ingeniería de sonido.

Bogotá, Colombia.

2022

Desarrollo e implementación de un algoritmo basado en Deep Learning para la identificación de señales de audio de arma de fuego.

Juan Sebastián Ochoa Rincón

Daniel Felipe Ojeda Santana

Director de proyecto:

Belman Jahir Rodríguez Niño

Universidad de San Buenaventura, Sede Bogotá.

Facultad de Ingeniería.

Programa de Ingeniería de Sonido.

Bogotá, Colombia.

2022

AGRADECIMIENTOS

DANIEL FELIPE OJEDA

Quiero agradecer, primero que todo, a mi familia, a mi hermano y a mis padres por el apoyo que me dieron tanto al momento de escoger la carrera como a lo largo de ella. También quiero agradecer a mi novia por el apoyo emocional que me dio los últimos tres semestres de estudio y por haberme acompañado en el proceso del desarrollo del documento de la tesis, así como en la corrección de la gramática, la ortografía y la redacción de este. Finalmente, quiero agradecer a mi compañero de tesis y de universidad Juan Sebastián Ochoa por haber decidido trabajar conmigo y tanto por haberme acompañado hasta este punto de la carrera como por los años de amistad que me ha brindado.

JUAN SEBASTIAN OCHOA

Antes que todo, quiero agradecer a Dios por permitirme obtener este logro en mi vida y por los que vendrán. Quiero agradecer a mi mamá por siempre acompañarme; gracias a ella soy la persona que soy hoy. No imagino el mundo sin tenerla, así que todo es por y para ella. Y a mi amigo y compañero de tesis Daniel Ojeda, con quien compartí muchas materias y con quien formé una gran amistad, le agradezco porque este es el resultado de muchas noches largas y de muchas frustraciones que al final mostraron un resultado y nos han trajeron hasta acá.

Por último, quiero agradecer a todas las personas que me apoyaron: familia, amigos y demás personas que estuvieron en el proceso.

TABLA DE CONTENIDO

LISTA DE TABLAS

LISTA DE FIGURAS

INTRODUCCIÓN

Este proyecto de grado está orientado hacia el campo de la programación y el análisis de audio mediante inteligencia artificial, donde se planea mejorar los sistemas de detección de disparos en su parte más frágil, es decir, la clasificación de la señal. Mediante la inteligencia artificial y algoritmos basados en Deep Learning, se desarrolla un algoritmo enfocado en la clasificación de señales acústicas de armas de fuego que sean iguales o inferiores a un calibre de 9 mm, ya que, en los sistemas creados y existentes, en el proceso de la detección y de la localización de disparos, la clasificación de la señal ha sido el punto más débil.

Los sistemas de detección de disparos son dispositivos creados mediante sensores acústicos para identificar y rastrear el origen de un disparo de arma de fuego o artillería. Los primeros dispositivos creados fueron desarrollados por *Reid* en el "Naval Surface Weapons Center" en el año de 1975 (Aguilar, 2015). En 1984, John Lahr y Fischer desarrollaron un sistema de detección de sonidos de altos niveles de presión sonora mediante cinco micrófonos dinámicos ubicados en un área de 84 x 28 metros. la transmisión de información se realizaba con técnicas de transmisión sísmica a través modulación FM, las cuales operaban en la banda de 160MHz y 170MHz con una potencia de 100 mW. Este sistema no funcionaba solo para disparos de armas de fuego, sino que también identificaba cualquier sonido impulsivo que sobrepasara los niveles de presión sonora establecidos (Lahr & Fischer, 1993). El primer sistema de detección de disparos más reconocido a lo largo de la historia fue desarrollado por Inglaterra alrededor del año 1994 para ser utilizado en operaciones contra francotiradores por las fuerzas británicas "United Nations Protection Force (UNPROFOR)" durante la guerra de Bosnia, la cual ha sido considerada el segundo conflicto más cruel en Europa después de la segunda guerra mundial (HRW, 2006).

Los sistemas de detección de disparos funcionan en tres etapas. La primera etapa consiste en la captura y registro de la señal de audio por medio de sensores acústicos, también conocidos como micrófonos. La etapa dos, en la que se enfocará este proyecto, es la etapa de clasificación de la señal, donde el sistema determinará si la señal que llega a los sensores acústicos es un sonido de disparo de arma de fuego o cualquier otro tipo de sonido. En la última etapa, se encuentra la localización del origen de la señal mediante la diferencia de tiempo de llegada, o en sus siglas en inglés "Time Difference of Arrival" (TDOA), que funciona por la variación temporal con la que llega la onda a cada uno de los micrófonos, esto sucede siempre y cuando el algoritmo, en la etapa anterior, determine que el sonido que llega a los micrófonos es un disparo de arma de fuego.

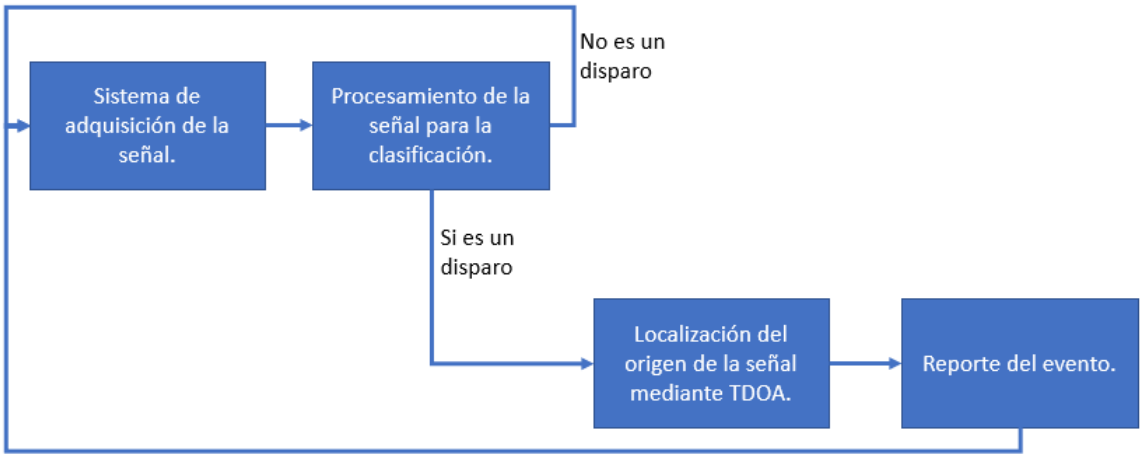


Figura 1. Proceso de un sistema de detección de disparos

La inteligencia artificial (IA) es una rama de la computación que tiene como propósito desarrollar máquinas, simulaciones, modelos y algoritmos capaces de realizar tareas humanas, generando aplicaciones para la predicción de patrones, sistemas de recomendación, asistentes personales, pilotos automáticos en vehículos y un sinfín de aplicaciones. En la última década, la inteligencia artificial ha crecido de manera exponencial, cambiando la manera en la que se había venido desarrollando el mundo. El área de la Inteligencia Artificial es un campo muy amplio que

abarca muchas áreas del conocimiento relacionadas con el aprendizaje automático y que ayudan a crear patrones y relacionar información teniendo en cuenta datos suministrados para que esta logre aprender e interpretar los datos y poder relacionarlos con datos de entrada para lograr generar un resultado de salida.

Mediante el uso de la Inteligencia Artificial, se elaborará un algoritmo basado en Deep Learning (aprendizaje profundo) que permita mejorar la clasificación de la señal en un sistema de detección de disparos, este ha sido el punto más débil en otros sistemas. Algunos de estos tienen dificultad en poder identificar si es un sonido producido por un arma de fuego con respecto a otros sonidos impulsivos, o que sobrepasen los niveles de presión sonoros establecidos con un umbral de detección y con características similares. Por medio de redes neuronales recurrentes, se busca programar un algoritmo que sea capaz de identificar los sonidos de disparos producidos por armas de fuego. Con el uso de Deep Learning, se puede entrenar un algoritmo que permita clasificar la información de entrada si se trata de un disparo de arma de fuego. Con este método, se pretende identificar si existe o no un sonido de disparo de arma de fuego sin depender de un umbral o Threshold (nivel mínimo requerido para que el sistema empiece a actuar para analizar la señal) de nivel de presión sonora, ya que el modelo estará alimentado con una cantidad de datos de audio diferentes. Los algoritmos basados en Deep Learning tienen la capacidad de analizar cada uno de los datos y extraer las características más relevantes que tengan en común los archivos de audio, a partir de esa información extraída, el algoritmo entrará a comprobar los sonidos con los datos de entrada que se quieren analizar.

Capítulo 1. Generalidades

1.1. Antecedentes

1.1.1. Ubicación de fuentes acústicas mediante Técnicas y software sismológicos (1993)

En este artículo, se muestra el inicio de los sistemas de triangulación en las diferentes ciudades de Estados Unidos, donde las llamadas de emergencia a la policía eran mayormente causadas por situaciones ligadas a disparos que no podían ser ubicados con exactitud por diferentes factores. Se empezó a implementar un sistema de triangulación básico que consistía en instalar 4 micrófonos en distintas terrazas en las que los amplificadores se conectaban a la entrada de 100 radios y se ubicaban los micrófonos a diferentes distancias para poder hacer una aproximación. El problema de aquel sistema era su demora en dar resultados, ya que necesitaba aproximadamente 3 horas para decir si se trataba de un disparo. Por esta razón, se utilizó un sistema de detección de terremotos, cabe recalcar que los terremotos son más fáciles de localizar (Lahr & Fischer, 1993).

1.1.2. Sistema para la localización de la dirección de disparos procedentes de armas de fuego utilizando técnicas de localización de fuentes sonoras (2014)

En este proyecto se buscó desarrollar el sistema completo de detección de disparos en los 3 puntos donde cada uno de ellos es un objetivo. Utilizando, en la clasificación, la técnica de correlación cruzada con el algoritmo base de Knapp y Carter en 1976, quienes fueron los primeros en proponerlo, se encontró que el sistema funciona al 100% en la triangulación, pero en el paso 2 de identificación de la señal se presentaban errores ya que, si se detectaban otras señales impulsivas con características similares, el sistema las detectaba como disparos de arma

de fuego, aunque no lo fueran. Se ha de recalcar que el sistema implementado en el proyecto, es un sistema móvil que fue localizado encima de un vehículo blindado (Morillas, 2014).

1.1.3. Avances en el análisis forense de grabaciones de disparos (2015)

En este artículo se realiza una caracterización de la acústica de las armas de fuego bajo condiciones controladas, por la falta de material acústico forense a la hora de esclarecer los hechos en una escena del crimen. En esta tesis se menciona que, tras el avance de la tecnología, muchos equipos son utilizados para realizar grabación ya sea únicamente de audio, o de audio y video. Lo que se buscó en el artículo es poder ampliar la base de datos de conocimientos forenses de audio a través de la captura de disparo de arma de fuego, para así, poder realizar las comparaciones correspondientes a la hora de realizar el análisis de los hechos, por ejemplo, con la BodyCam de la policía, la cual graba todos los sucesos con audio y video. Para la caracterización, se tuvo en cuenta las reflexiones y la reverberación que se pueden convertir en sonidos superpuestos y cambiar las características tomadas. Se realizaron grabaciones de 10 sonidos de armas diferentes (Maher, Robert C. & Routh, 2015).

1.1.4. Sistemas de detección de disparos en la aplicación de la ley civil (2015)

El documento en cuestión se centra en mostrar la historia que han tenido los sistemas de detección de disparos a lo largo de los años, en cómo se han desarrollado en los diferentes países y qué importancia tienen a la hora de aplicar la ley. Además, se muestra cómo ha sido su aplicación y sus modificaciones dependiendo de los factores necesarios. Aparte, muestra cuáles son las necesidades de implementar el sistema y cuáles son los beneficios que el mismo trae a la comunidad donde es aplicado. Claramente, el autor demuestra que se puede seguir mejorando el sistema, ya

que, la mayoría de las veces, en los diferentes sistemas, la principal falla es el algoritmo, donde este ha probado tener falencias, aunque sean muy débiles (Aguilar, 2015).

1.1.5. Sistema de reconocimiento de comandos por voz basado en redes de neuronas LSTM (2018)

La tesis desarrollada explica la importancia de las redes neuronales y especialmente, las que están basadas en LSTM. Basado en dichas redes neuronales, el autor recalca su importancia y realiza una captura de cierto tipo de archivos o comandos de voz en formatos wav, para así, poder entrenar un algoritmo. Este último, centrado en redes neuronales, que reconozca los comandos de voz realizando distintas tareas. Todo esto, apoyándose en inteligencia artificial y en los sistemas de asistencia virtual, como los que incorporan los teléfonos inteligentes de hoy en día (Google Assistant, Siri y Alexa). Cabe aclarar que las redes neuronales desarrolladas en esta tesis se basan en reconocimiento del habla (ASR). Tras el desarrollo de esta, no se logró superar por mucho la precisión esperada de los datos de referencia. El autor menciona que aún falta demasiado estudio en esta rama (Cabero, 2018).

1.1.6. Una correlación cruzada a corto plazo con la aplicación al análisis forense de disparos (2019)

El autor realiza un análisis detallado de la correlación cruzada en el campo de la acústica forense, demuestra la importancia de esta y cuáles son los factores que se estudian en ella, demostrando la importancia de la transformada rápida de Fourier (FFT). Gracias a esto, se pueden analizar las características de un disparo separando el ruido de la señal del disparo de otras. Se considera que el uso de este método es conveniente en muchos sentidos, el autor utiliza técnicas de

casos forenses para demostrar los faltantes que se pueden presentar y la coherencia que los mismos pueden tener, dentro de los cuales se analizan los diferentes sonidos de armas (Beck, 2019).

1.2. Planteamiento del problema

A través de los años, los forenses han buscado la manera más sencilla de analizar, identificar y clasificar un sonido de arma de fuego mediante diferentes técnicas. Entre estas, se encuentra la correlación cruzada, que analiza el tipo de arma, la distancia y los posibles ángulos de disparos (Beck, 2019). Al realizar este procedimiento, se generaron varios inconvenientes, ya que se deben tener presentes factores como: el rango de frecuencia, la amplitud, la atenuación por distancia, las reflexiones y las absorciones con las superficies.

En ese mismo año, Hostile Artillery Locator (HALO) realizó un sistema capaz de calcular la ubicación de armas, morteros y proyectiles en el campo de batalla en el conflicto armado que se estaba viviendo en Bosnia. El sistema consistía en implementar un sistema de sensores acústicos de capa límite para basar su procesamiento en dinámica de fluidos. También se utilizó para recopilar señales acústicas y así obtener una imagen del campo de batalla en el dominio acústico (Aguilar, 2015).

Años más tarde, Metravib creó, para las operaciones militares de Francia PILAR, un sistema dedicado a analizar los sonidos de proyectiles y explosivos mediante la firma supersónica producida por la diferencia de presión generada por la boca del cañón y la presión exterior. Esto con el propósito de calcular la elevación y estimar la trayectoria de la bala. En ese mismo sistema, también se usaron sensores acústicos que eran los encargados de calcular la distancia a la que se encontraba el tirador teniendo en cuenta la humedad, la velocidad de viento y la temperatura. El sistema completo era poco portable y no era capaz de diferenciar otros sonidos impulsivos con respecto a los generados por un arma de fuego o por una explosión. La identificación de la señal se

hacía mediante “Threshold”, lo que significa que, al momento de pasar un umbral de presión sonora, el sistema empezaba a realizar la localización (Millet & Baligand, 2006).

A finales de 1993, y a mediados de 1994, Joint Counter Sniper Program inició un proyecto para las Fuerzas Militares de Estados Unidos para crear un prototipo avanzado que determinara la localización, la clasificación y el calibre de un arma con la que se hubiese disparado. El prototipo que fue implementado consistía en el posicionamiento de 6 transductores acústicos y de un sensor de detección infrarroja acústica puestos estratégicamente en un área determinada y teniendo en cuenta las características acústicas del lugar (Aguilar, 2015).

De acuerdo con los antecedentes descritos anteriormente, existe una falencia en la etapa dos de los sistemas de detección de disparos: la clasificación que permite saber si un sonido es o no un disparo de arma de fuego. Debido a que muchos de los sistemas nombrados basan inicialmente su procesamiento de la señal teniendo en cuenta el umbral, los sonidos que no superan el umbral no pueden seguir con la siguiente etapa del análisis de la señal para la clasificación de esta misma. También se debe tener en cuenta que muchos de los sistemas no diferencian los disparos de armas de fuego de otros sonidos impulsivos como bombas o juegos pirotécnicos.

1.2.1. Pregunta problema

¿Cómo identificar señales de disparo de arma de fuego a partir de otras señales acústicas mediante Deep Learning?

1.2.2. Justificación del problema

En el mundo, el comercio de armas de fuego ha venido aumentando de forma exponencial, ya que ciertas personas las ven necesarias para usarlas como defensa propia o para realizar actos delincuenciales. Cada año, en países como Colombia y México, se producen más de 12.000 muertes

causadas por armas de fuego (Aguilar, 2015), lo que genera que la inseguridad esté en aumento. Debido a esto, de toda América Latina, Colombia se posiciona en el quinto puesto como país más violento e inseguro con 24 homicidios por cada 100.000 personas. Según la Política internacional sobre armas de fuego, el número de armas pertenecientes a civiles es de 4.971.000 en el año 2017, lo que equivale a un 10% de la población (*Gun Law and Policy: Firearms and Armed Violence, Country by Country*, n.d.).

Durante el transcurso de los años, se ha buscado la manera de disminuir la violencia, mediante el uso de armas de fuego, y proteger a las personas. Una de estas maneras está formada por la localización, la identificación y la caracterización de una señal de sonido de arma de fuego; ya sean armas de pequeño calibre o armas de francotiradores. Desde 1975, ha habido propuestas y prototipos que han buscado localizar la procedencia de un disparo. Naval Surface Weapons Center diseñó un sistema capaz de localizar la procedencia de un disparo mediante el uso de 5 sensores acústicos, en un área de 840 x 280 m, distribuidos de forma estratégica teniendo en cuenta el área a trabajar. La recopilación de datos de los transductores se podía llevar a cabo mediante una técnica de transmisión sísmica. Sin embargo, este análisis no permitió identificar dos señales impulsivas con amplitudes similares (Lahr & Fischer, 1993).

Como se ha mencionado, en la mayoría de los sistemas de detección de disparos, el procesamiento inicial de la señal inicia con los niveles de presión sonora. Cuando un sonido producido por un arma de fuego sobrepasa los umbrales de presión sonora establecidos, se inicia un procesamiento de la señal mediante diferentes técnicas. Entre estas está la correlación cruzada, que analiza tanto las diferentes características de un arma de fuego, como en su parte frecuencial, de tiempo y amplitud. Esto, mediante expertos acústicos, los cuales son los encargados de analizar cada una de las señales impulsivas que llegan a los sensores.

Mediante la Ingeniería de Sonido, se planea usar los conocimientos en programación y el crecimiento tan importante que está teniendo la Inteligencia Artificial en el mundo actual, para así, implementar un algoritmo basado en Deep Learning. Este último, entrenado con señales de audio de disparos de armas de fuego de pequeño calibre (menores o iguales a 9mm), que tiene como objetivo abarcar el problema de violencia que se genera en las ciudades y en la cual se ven implicadas este tipo de armas, teniendo en cuenta que entre los años 2014 y 2016 fueron incautadas 44.615 armas hechizas en su mayoría, que son armas de pequeño calibre o revólveres (Neira, Néstor Humberto Martínez, 2019). Usando el Deep Learning y las redes neuronales se realizará un algoritmo que permita saber si la señal que se le ingresó es o no, un disparo de arma de fuego teniendo en cuenta las características que el algoritmo extrae y encuentra en común. Una de las ventajas de usar este sistema es que no se va a depender del Threshold, este es el método que implementa la mayoría de los sistemas de detección de disparos. El algoritmo basado en Deep Learning, al momento de analizar y extraer las características en común de los audios que se le van a ingresar para realizar el entrenamiento, va a tener en cuenta más características de la señal para realizar la comparación. En caso de que la precisión (Accuracy) se quiera mejorar, una de las soluciones es entrenar el algoritmo con más datos de audio para que empiece a tener más referencias y tenga más información sobre las características en común. Otra opción de optimización se puede lograr modificando los parámetros e hiperparámetros del modelo. Teniendo en cuenta lo anterior, si se quiere realizar el mismo proceso con otro tipo de armas (Sniper), en el algoritmo solo se deben cambiar los datos de entrenamiento y realizar ajustes en el número de neuronas y capas a usar. Eso hace que, por medio de un mismo algoritmo, también se pueda realizar un proceso de análisis con diferentes tipos de armas.

1.3. Objetivo General

- Desarrollar un algoritmo basado en Deep Learning para la identificación de señales impulsivas de arma de fuego.

1.4. Objetivos Específicos

- Determinar el DataSet de señales impulsivas para el entrenamiento y evaluación del algoritmo.
- Implementar dos (2) algoritmos ya existentes basados en Deep Learning con aprendizaje supervisado que permita conocer si la señal es o no de disparo de arma de fuego.
- Evaluar la precisión (Accuracy) de los algoritmos con el 20% del DataSet para validación.

1.5. Alcances y Limitaciones

1.5.1. Alcances

El proyecto se enfocará en la clasificación de señales de arma de fuego mediante Deep Learning que será un método diferente a los que se han venido desarrollando. El algoritmo podrá determinar si un sonido es o no un disparo, sin importar el umbral de la señal. El algoritmo podrá identificar señales de disparo y diferenciarlas de otros sonidos sin importar su amplitud, a diferencia de muchos de los sistemas existentes que suelen procesar la señal del sonido producido por el arma, sonido que debe sobrepasar los niveles de presión sonora establecidos por el fabricante. Finalmente, el proyecto solo se basará en la etapa dos de un sistema de detección de disparos (clasificación de la señal), ya que es el punto donde existe mayor problema.

1.5.2. Limitaciones

Teniendo en cuenta que los sistemas de Deep Learning necesitan un proceso de representación, aprendizaje y despliegue, el sistema no funcionará en tiempo real. También, se debe tener en cuenta que al momento de realizar algoritmos de Deep Learning, se necesita un procesamiento computacional complejo.

Para poder entrenar el algoritmo, se necesitará una amplia base de datos de audio de disparos de armas de fuego de calibre menor o igual a 9mm. El sistema se validará en un espacio al aire libre en la ciudad de Bogotá para evitar posibles reflexiones generadas por las paredes. El algoritmo se implementará en Google Colab para evitar usar procesamiento de un computador físico, este procesamiento se realiza mediante internet. El procesamiento de máquina de CPU, GPU y memoria RAM se realizan mediante los servidores proporcionados por Google. Una de las ventajas que tiene trabajar con Google Colab es que el servidor tiene la capacidad de brindar más memoria RAM y procesamiento de GPU según las necesidades del algoritmo. Finalmente, el proyecto solo se basará en la etapa dos de un sistema de detección de disparos, por lo tanto, no se implementará un arreglo

de micrófonos ni de algoritmos de triangulación y tampoco se tendrán en cuenta disparos realizados con armas que tengan adecuación de algún tipo de silenciador o que sean de disparos secuenciales.

Capítulo 2. Marco Conceptual

2.1. Inteligencia Artificial (IA)

Muchos de los sistemas, aplicaciones y herramientas que se usan actualmente están desarrollados bajo el término de inteligencia artificial. Por ejemplo, Google Translate que es un sistema que permite traducir texto de una lengua a otra y que también tiene la capacidad de recibir información sonora en un idioma y traducirlo a un centenar de otros idiomas. Otro de los sistemas basados en inteligencia artificial que se usa en la vida cotidiana y que no nos damos cuenta de que funciona mediante la inteligencia artificial, es el que se usa en los correos electrónicos para identificar los enviados y los que son clasificados como *spam*, es decir, correos que se envían a millones de usuarios con una finalidad comercial. Mediante el reconocimiento de objetos usando la inteligencia artificial, podemos encontrar sistemas de conducción autónoma de vehículos como los que realiza *TESLA.* Esta compañía usa las imágenes de señales de tránsito, personas y objetos que se puedan encontrar en una vía para poder realizar cálculos y predicciones y así poder tomar decisiones para lograr la conducción autónoma en cuestión. Teniendo en cuenta lo anterior, se puede afirmar que la inteligencia artificial es una disciplina que busca imitar la inteligencia y el comportamiento de los humanos (Torres, 2020).

2.1.1. Machine Learning

El Machine Learning o aprendizaje automático es un subcampo de la inteligencia artificial que proporciona a los ordenadores la capacidad de aprender sin que estos necesiten una persona que indique las reglas que debe seguir. El Machine Learning consiste en desarrollar un algoritmo para cada problema. Existen principalmente tres categorías dentro del Machine Learning: El aprendizaje

supervisado (supervised Learning), el aprendizaje no supervisado (unsupervised Learning) y el aprendizaje por refuerzo (Reinforcement Learning).

Cuando se habla de aprendizaje supervisado, para el entrenamiento, se usan datos que incluyen la solución deseada, llamada etiqueta (Label). Este modelo, de manera iterativa, se va afinando mediante la generación de predicciones sobre los datos de entrenamiento y compara estas predicciones con la respuesta correcta que el algoritmo ya conoce. En el aprendizaje no supervisado, a diferencia del supervisado, los datos de entrenamiento no incluyen etiquetas, lo que significa que el algoritmo clasificará la información por sí mismo (Torres, 2020).

2.1.2. Deep Learning

El Deep Learning o aprendizaje profundo es un subcampo de la Inteligencia Artificial y es una disciplina inmersa en el Machine Learning. Este modelo sigue los mismos principios de Machine Learning, el cual proporciona a los ordenadores la capacidad de aprender sin que estos necesiten una persona que indique las reglas que debe seguir. A diferencia del Machine Learning, el Deep Learning es un modelo computacional que no requiere una etapa previa de extracción de características y necesita el uso de etiquetas. Esto quiere decir que el modelo tiene la capacidad de extraer las características y patrones más relevantes de forma autónoma de los datos en bruto (sin ningún tipo de procesamiento previo ni etiquetas), para finalmente tomar una decisión. Hay tres clasificaciones de aprendizaje dentro del Deep Learning: el aprendizaje supervisado, el aprendizaje no supervisado y el aprendizaje por refuerzo. (Sotaquirá Gutiérrez, 2018).

2.1.3. Redes neuronales artificiales

Las redes neuronales artificiales, en cierta forma, intentan imitar la actividad de las neuronas del cerebro humano donde ocurre el procesamiento de la información. Los modelos compuestos

por múltiples capas de procesamiento (capas construidas por neuronas artificiales) realizan una serie de trasformaciones lineales y no lineales que, a partir de los datos de entrada, generan una salida aproximada a la esperada (Torres, 2020).

Tomado de: (Atria, 2019)

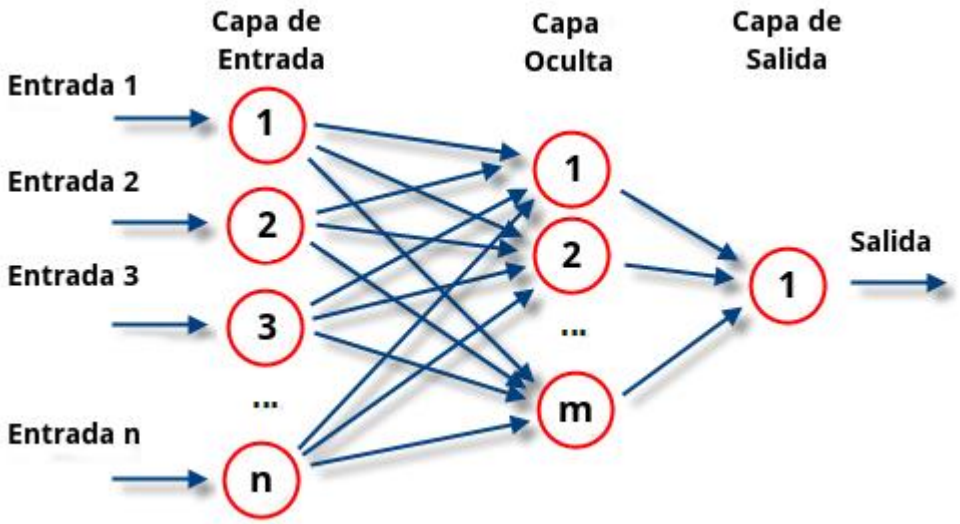


Figura 2. Red Neuronal artificial

2.1.4. Redes Neuronales Convolucionales

Las redes neuronales convolucionales son una arquitectura de una red para Deep Learning que se encarga de aprender indirectamente de los datos o muestras dadas, esto sin necesidad de extraer características manualmente de cada uno de ellos.

Las redes neuronales convolucionales son particularmente utilizadas para encontrar patrones. Un ejemplo de dichos patrones puede ser el reconocimiento de objetos, rostros o ciertas

características. Como estas redes son tan eficaces, se utilizan también para clasificar datos que pueden comprender desde datos de audio hasta series temporales y señales (MathWorks, n.d.).

2.1.5. Capas Convolucionales

Las capas convolucionales funcionan con diferentes subcapas, de las cuales, la primera extrae las características principales. En este proceso, las capas convolucionales procesan información y, mientras esto pasa, se va reduciendo la dimensión de la matriz.

Tomado de: (POCHO, n.d.)

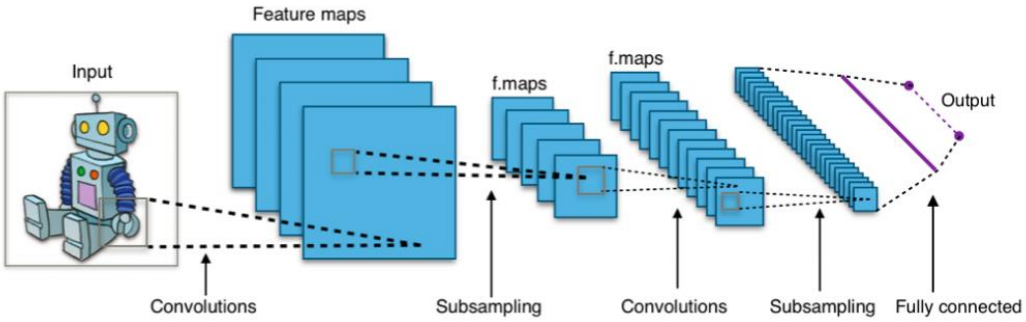


Figura 3. Redes convolucionales.

Cada capa convolucional realiza operaciones sobre cierto tipo de imagen, gracias a la aplicación de filtros. En cada paso se extraen características más específicas donde se logra la identificación del resultado. Una red convolucional transforma los datos de una cuadrícula conocida (Goodfellow et al., 2016).

2.1.6. Epoch

Epoch es un parámetro que se encarga de determinar el número de veces que el algoritmo se entrenará usando los datos de entrenamiento y test. Cuando se define un Epoch se puede interpretar

que cada muestra ha tenido la oportunidad de actualizar los parámetros del algoritmo. El número de Epoch en un algoritmo es tradicionalmente grande, gracias a esto permite que el algoritmo corra hasta que el error del modelo se haya minimizado lo más posible (Machine Learning Mastery, 2018b).

2.1.7. Función de activación

En las redes neuronales, tanto en las biológicas como en las artificiales, una neurona no solo transmite la entrada que recibe, sino que hace un paso más, conocido como la función de activación. Esta es análoga a la tasa de potencial de acción, la función se expresa en la ecuación:

$$Z = b + \sum_i W_i X_i \qquad (1)$$

Z: Salida.
b: Sesgo.
W: Peso.
X: Entrada.

La señal de entrada es transformada como salida; esto permite que las señales de salida no sean funciones lineales, puesto que el aprendizaje de una función lineal es limitado. De esta forma, la función de activación se encarga de acotar la salida con valores de probabilidad entre 0 y 1 dependiendo del tipo de función.

2.1.7.1. Softmax

El Softmax es una regresión logística que extiende la idea a un lugar de múltiples clases, se podría decir que Softmax asigna probabilidades decimales a cada clase en un problema de varias clases. Dichas probabilidades decimales deben sumar 1.0, esta redistribución ayudará a que el

entrenamiento converja más rápidamente de lo que lo haría de otra forma (Machine Learning Crash Course, n.d.).

Tomado de: (Machine Learning Crash Course, n.d.)

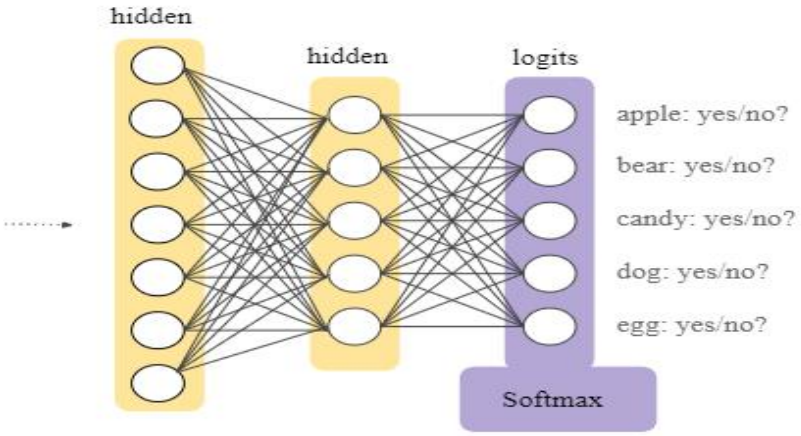


Figura 4. Una capa Softmax dentro de una red neuronal.

2.1.7.2. Sigmoide

La función de sigmoide es una función matemática con una característica en forma de S. Existen diferentes funciones sigmoides y esta es utilizada normalmente para especificar la función logística, recibiendo, también, el nombre de *función sigmoide logística*.

Todas las funciones sigmoides tienen propiedades que mapean toda la recta numérica en un rango pequeño, este rango se puede presentar entre 0 y 1 o -1 y 1. Entre otras cosas, gracias a esto, la función sigmoide convierte un valor real en uno que pueda interpretarse como una probabilidad.

La función sigmoidal puede usarse como una función de activación en una red neuronal artificial.

Tomado de: (DeepAI, n.d.)

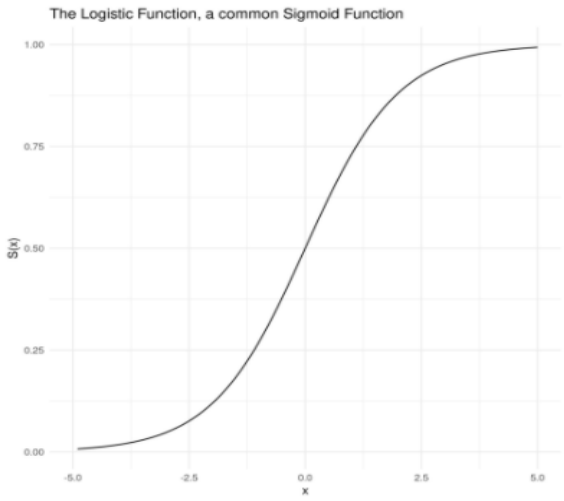


Figura 5. Función sigmoide

2.1.8. Función de pérdida

La función de pérdida, también conocida como *función de costo*, determina que tan buena es la red neuronal. Si el resultado es alto, indica que la red neuronal tiene rendimiento pobre y si el resultado es bajo, indica que la red está haciendo un buen trabajo (Torres, 2020)

2.1.9. Optimizadores

Dentro de un algoritmo existen varios optimizadores. Estos tienen como tarea acelerar la convergencia del algoritmo y prevenir mínimos locales. Existen varios optimizadores; el más utilizado en el algoritmo es el Adam. Este optimizador es una variante descendente en el que se encuentra una tasa para cada red.

Principalmente, se utiliza la estimulación promedio de gradientes o derivadas parciales. Estas se presentan en la función de error o como se conoce perdida. Dicha función se encarga de encontrar cada una de las tasas de aprendizaje (Torres, 2020).

2.1.9.1. Gradiente Descendiente

El método de gradiente descendente es uno de los métodos de optimización multivariantes más antiguos y conocidos. Este método trata de obtener una aproximación lineal de la función de error a través de la expresión:

$$E(w + \Delta w) = E(w) + \Delta w \acute{E}(w). (2)$$

De forma que la actualización de los pesos (w) viene dada por:

$$\Delta w = -\alpha E \acute{}(w), \alpha \succ 0. (3)$$

Siendo α el tamaño del paso o tasa de aprendizaje, que suele ser de tamaño reducido $0 < \alpha \leq 1$ (Flórez López & Fernández Fernández, 2008)

2.1.10. Overfitting

El Overfitting se produce cuando el modelo obtenido se ajusta tanto a los ejemplos adquiridos en el entrenamiento que no puede realizar las predicciones correctas con datos nuevos. En concreto, esto ocurre cuando la red modela los datos de entrenamiento muy bien, aprendiendo características de estos que no son generales.

Cuando sucede el sobreajuste u Overfitting, se presenta una situación en la que el modelo puede presentar un bajo porcentaje de error de clasificación. El inconveniente es que este porcentaje no se generaliza correctamente a la población de datos en los que se está interesado. Es claro que, en general, esta situación presenta un impacto negativo en la eficiencia del modelo cuando este se

utiliza para inferencia con datos nuevos. Por esta razón, es muy importante evitarla, así como también es de gran importancia reservar una parte de los datos de entrenamiento como datos de validación, los cuales permitirían detectar si esto ocurre (Torres, 2020).

Tomado de: (Aprende Machine Learning, 2017)

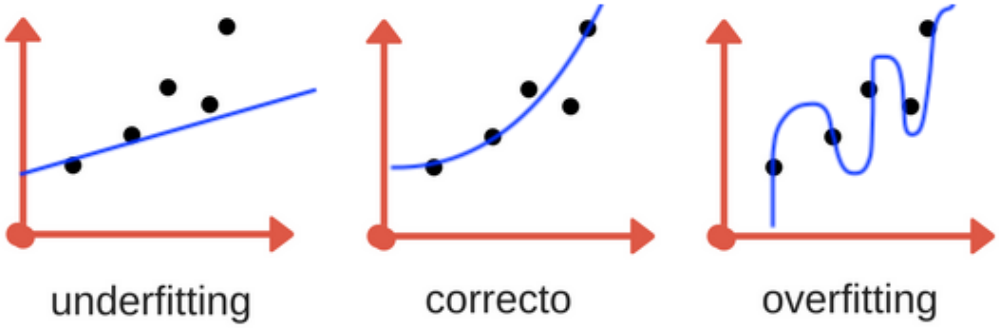


Figura 6. Overfitting o sobreajuste

Tomado de: (Aprende Machine Learning, 2017)

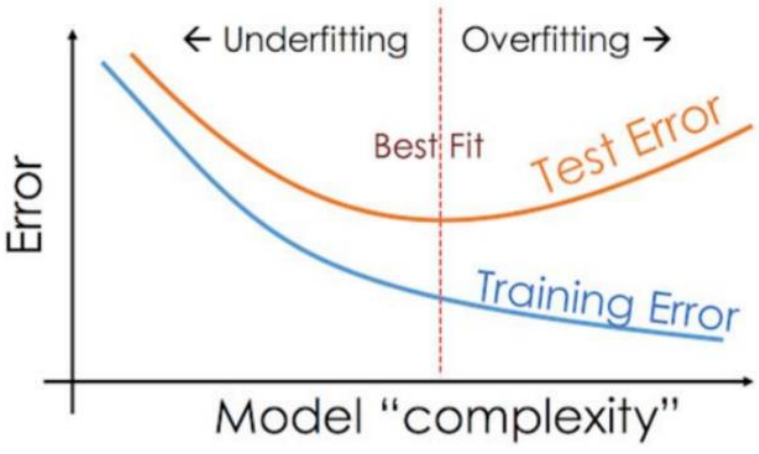


Figura 7. Overfitting y underfitting en grafica de Función de pérdida

2.1.11. Dropout

El Dropout es un método con el cual se realiza la desactivación aleatoria de algunas neuronas de la red. Las neuronas en cuestión dejan de ser tenidas en cuenta para el entrenamiento. Esto es una gran ventaja ya que las neuronas dependen de otras neuronas más cercanas, lo cual implica que, en su relación, se forman patrones específicos para un set de datos, generando problemas de Overfitting y de adaptación de la red (Torres, 2020).

2.1.12. Entropía cruzada

La entropía cruzada es una función de pérdida que se encarga de especificar la diferencia entre dos distribuciones de probabilidad. Aquí se establece qué tan cercana o alejada es la distribución del dato real con respecto a la distribución de la predicción. Esta sería la ecuación para definir la entropía cruzada (Torres, 2020):

$$H(p,q) = -\sum_{x} p(x) \log q(x) \quad (4)$$

P(x): probabilidad deseada.
q(x): probabilidad real.

2.1.13. Entropía Binaria

La entropía binaria es un proceso de Bernoulli con probabilidad de dos valores: 0 o 1. En forma de proceso de Bernoulli, la función de entropía se modela matemáticamente como una variable aleatoria que puede tomar solo dos valores 0 y 1. Estos se presentan siendo excluyentes y exhaustivos entre ellos (Torres, 2020).

$$H(p,q) = -\sum_{x} p(x) \log q(x) \quad (5)$$

P(x): probabilidad deseada.
q(x): probabilidad real.

2.1.14. Coeficientes de Mel

El Mel Cepstrum o MFCC representa el espectro de potencia de sonido a corto plazo. Su principal fundamento es transformar el coseno lineal del espectro de potencia logarítmica de frecuencia, que está representado en escala de Mel no lineal. (Llorente et al., 2007).

Tomado de: (Llorente et al., 2007)

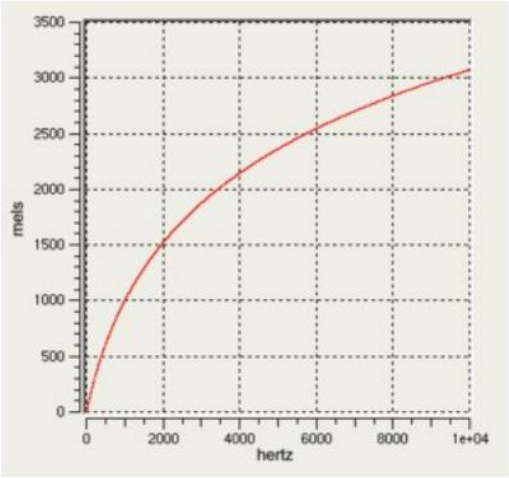


Figura 8. Escala de MEl.

Para extraer los Coeficientes de Mel de una señal, primero se segmenta la señal, después de esto, se calcula la estimación del periodo del espectro en potencia para cada sección y, finalmente, se utiliza el filtro Mel para filtrar el espectro de potencia de cada fotograma para luego poder unirlo con la energía filtrada de cada fotograma y así poder tener la energía total (Llorente et al., 2007).

2.1.15. Pooling

En las redes neuronales convolucionales es importante reducir el tamaño de los datos después de aplicar una capa de convolución, por lo tanto, se lleva a cabo un submuestreo llamado

también Pooling. El max-pooling se encarga de reducir la información por capas teniendo siempre en cuenta la información más relevante. Una forma de describirlo es más claramente es con un ejemplo, como se observa en la Figura 7 se describe claramente que se realiza un max-Pooling de 2x2. En el caso de una imagen, se tomarían algunos pixeles para el ancho y el alto, conservando el valor máximo para así poder reducir el número de neuronas en cada capa.

En estas formas de red, a medida que se incrementa el número de capas, aumenta el número de filtros kernel, mientras se disminuye el tamaño de las imágenes a través del max-Pooling.

Tomado de: (DeepAI, n.d.)

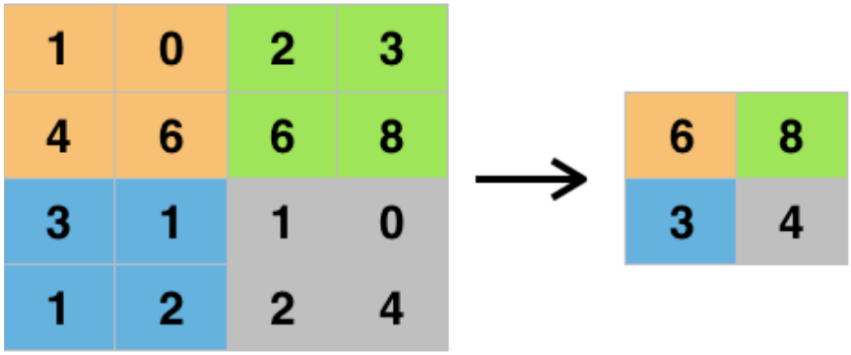


Figura 9. Explicación de Pooling.

2.1.16. Padding

El Padding se puede describir fácilmente como la forma de reducir la imagen. Es necesario obtener una imagen de salida de igual tamaño a la imagen original para el proceso de convolución. El padding es un proceso donde se completa con ceros los pixeles que forman parte de los bordes de la imagen de salida, como se puede observar en la Figura 8 (DeepAI, n.d.).

Tomado de: (DeepAI, n.d.)

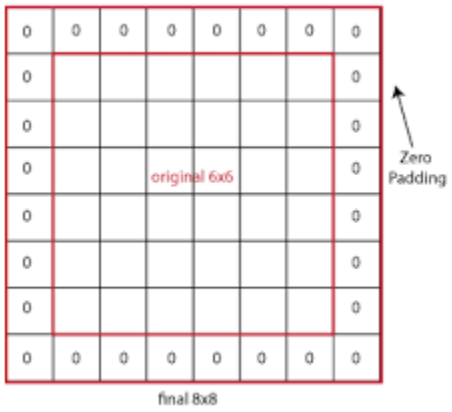


Figura 10. Padding

2.1.17. Stride

Stride es un componente de las redes neuronales en general o de las redes neuronales convolucionales. Stride también es un parámetro del filtro de la red neuronal que modifica la cantidad de movimientos sobre una imagen o un video. Por ejemplo, en el paso de una red neuronal, se establece en 1 y el filtro se moverá un píxel o una unidad a la vez. Esto quiere decir que el tamaño del filtro afectará el volumen de la salida por lo que muchas veces se establece un número entero en vez de un número decimal (Yosinski et al., 2014).

2.1.18. Matriz de confusión

Esta herramienta, de las conocidas en Machine Learning, se utiliza para evaluar el rendimiento de los modelos. Esta se conforma de una tabla con filas y columnas que contabilizan las predicciones comparándolas con los valores reales. La matriz de confusión es utilizada para entender mucho mejor qué tan bien o mal se comporta el algoritmo, esta matriz es muy útil para

mostrar de forma explícita cuando una clase se confunde con otra. Una matriz de confusión binaria se podría comportar de la siguiente manera (Torres, 2020).

Tomado de: (Torres, 2020)

		Predicción	
		Positivos	**Negativos**
Observación	**Positivos**	Verdaderos Positivos (VP)	Falsos Negativos (FN)
	Negativos	Falsos Positivos (FP)	Verdaderos Negativos (VN)

Figura 11. Matriz de confusión.

VP: El dato real (1) es igual al predicho (1).

FN: El dato real (1) es diferente al predicho (0).

FP: El dato real (0) es diferente al predicho (1).

VN: El dato real (0) es igual al predicho (0).

2.1.19. Kernel

En Machine Learning, un kernel se refiere a un método que permite aplicar clasificadores lineales a problemas no lineales mapeando datos en un espacio de mayor dimensión sin la necesidad de comprender ese espacio de mayor dimensión (Belkin et al., 2018).

2.1.20. ReLu

ReLu es una función de activación no lineal que ha ganado popularidad en el dominio del aprendizaje profundo. Entre las principales funciones de ReLu se encuentra aquella en la que no se activan todas las neuronas al mismo tiempo. Esto significaría que las neuronas solo se desactivarían si la salida de la transformación lineal es menor a 0.

Tomado de: (IArtificial.net, 2020)

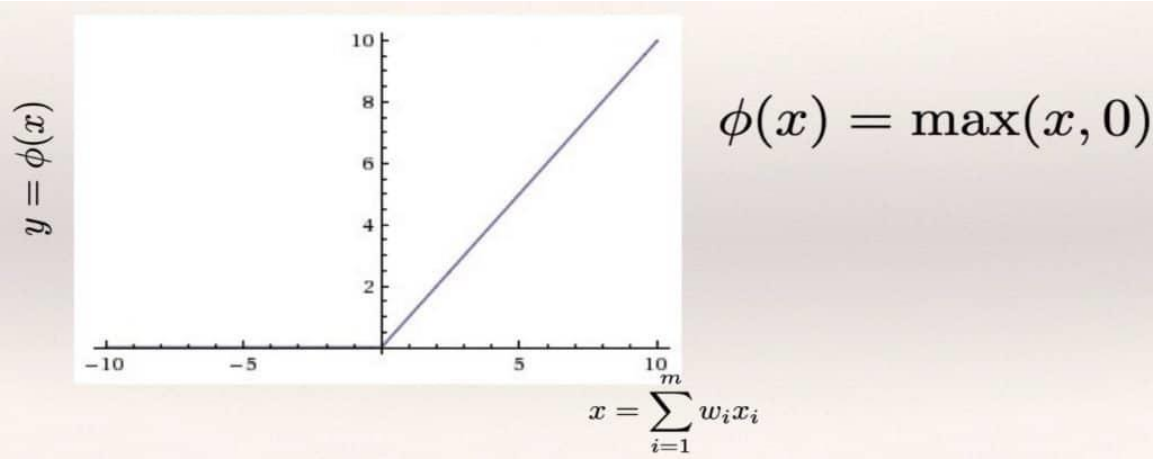


Figura 12 Función ReLu.

Cuando se presentan valores de entrada negativos, el resultado es cero; es decir que la neurona no se activa.

2.1.21. Armas de fuego

2.1.21.1. Muzzle blast

Las armas de fuego utilizan una combustión de pólvora para impulsar la bala fuera del cañón. El sonido producido por la combustión se emite muy rápido desde la pistola hacia todas las direcciones, pero la mayor parte de la energía acústica se expulsa en la dirección en la que el cañón del arma está apuntando.

La onda de choque no lineal y la energía sonora relacionada que emana del cañón se denominan *explosión de boca,* con una duración normalmente de 3 milisegundos. La onda acústica del estallido del cañón se propaga a través del aire a la velocidad del sonido y es parcialmente reflejada, absorbida y difractada por la superficie del suelo y otros obstáculos físicos, esto quiere

decir que la señal acústica recibida también tiene efectos de propagación, reflejos de múltiples rutas y reverberación (Maher, Robert C. & Routh, 2015).

Tomado de: Elaboración propia

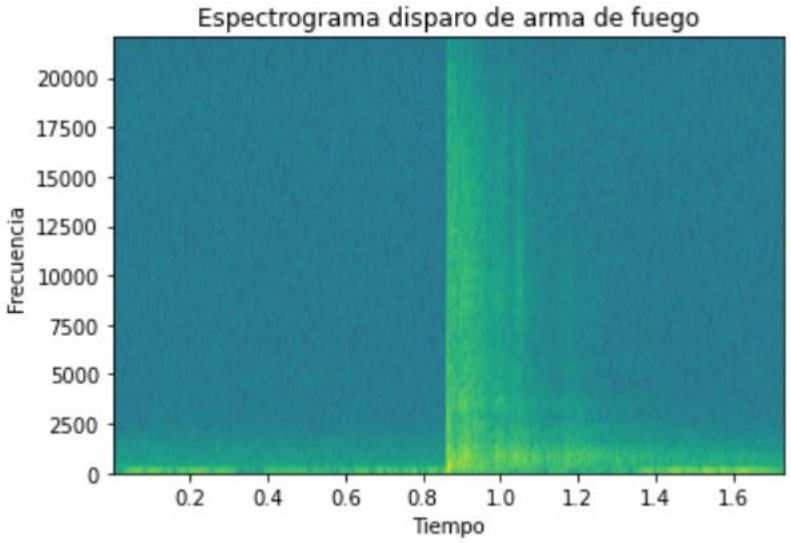


Figura 13. Espectrograma de disparo de arma de fuego

2.1.21.2. Proyectil supersónico

Dentro de las particularidades de las armas de fuego está el estallido del cañón. Otra fuente acústica importante es el disparo de la bala si esta viaja a velocidad supersónica. El paso de un proyectil a través del aire lanza una onda de choque acústica que se propaga hacia afuera desde la trayectoria de la bala. La onda de choque se expande detrás de la bala en tres dimensiones como un cono de onda de choque. El frente de onda se propaga hacia afuera a la velocidad del sonido (Maher, Robert C. & Routh, 2015).

2.1.22. Dither

El Dither es un proceso digital que se aplica cuando se quiere pasar de una frecuencia de muestreo mayor a una frecuencia de muestreo menor donde al momento de reducir la profundidad de bits se elimina información. Este proceso tiene como propósito agregar un ruido de bajo nivel donde se agrega información en el audio. Al momento de agregar la nueva información y convertir la profundidad de bits, la conversión eliminará parte de la nueva información agregada y eso evita que el error de perdida de información sea menor y no afecte en gran parte el audio (López León, 2014).

Tomado de: (LANDR Blog, 2018)

Figura 14. Proceso de Dithering.

Como se observa en la Figura 14, se tienen 3 imágenes las cuales representaran un audio. En la primera imagen, se tiene la imagen original sin realizar ningún proceso. La segunda imagen, es la reducción de la profundidad de bits sin realizar el proceso de Dithering, como se observa, se ve como la información y la calidad de la imagen es menor y pierde gran parte de la información. Y en la ultima imagen, se ve como se reduce la profundidad de bits de la imagen original, pero

agregándole el proceso de Dithering, se pierde parte de la información y la calidad pero no tanto como cuando no se realiza el proceso de Dithering (LANDR Blog, 2018).

Capítulo 3. Metodología

La metodología del proyecto es cuantitativa, fue definido por: “Hernández Sampieri et al. (2010) el estudio cuantitativo como secuencial y probatorio. Comienza a partir de una idea general que se convierte en un problema delimitado y concreto” (Hernández Sampiere, Roberto; Fernández Collado, Carlos; Baptista Lucio, 2012). Siendo este el más apropiado por su forma secuencial y probatorio. Basado en que se tiene una serie de pasos para cumplir el objetivo general, el enfoque empírico – analítico será el más apropiado. En ese, la recolección de datos se realizará de forma probatoria, basándose en los antecedentes anteriormente descritos. Para este fin, se establece una estrategia donde se determinarán, analizarán y se mirarán las variables dependientes.

Tomado de: Fuente propia

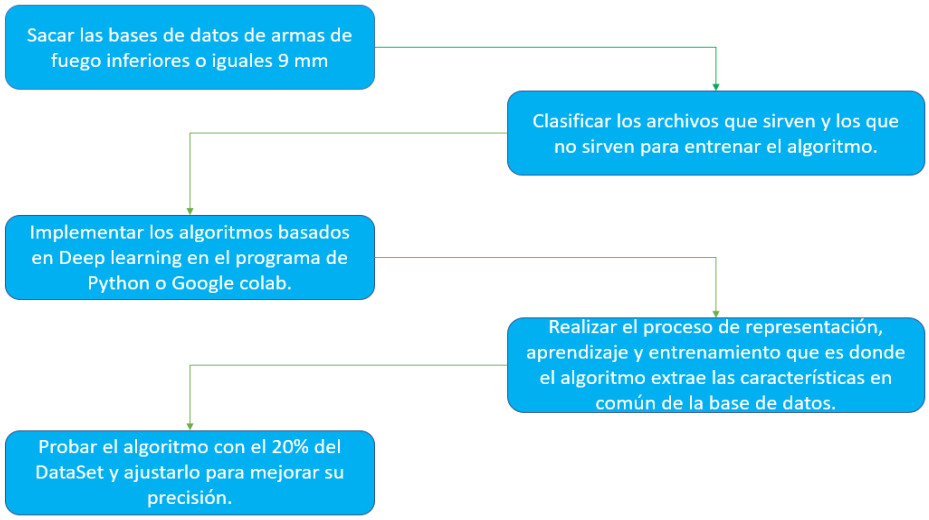


Figura 15. Diagrama metodológico del proyecto.

Las técnicas de recolección de datos que se utilizarán en el proceso para el entrenamiento y evaluación de los datos se realizarán por medio de la página “Gunshot Audio Forensics DataSet” que fue realizada en una zona rural de Arizona, Estados Unidos en el año de 2017. Esta base de

datos está comprendida por aproximadamente 10.000 grabaciones, en las que se encuentran sonidos de diferentes tipos de armas. Estas grabaciones están clasificadas según el arma con la que fue realizado el disparo. A partir de esa información, se usan los audios de las armas de pequeño calibre (menores o iguales a 9mm) (*Gunshot Audio Forensics DataSet – Gunshot Audio Analysis*, 2017).

Tomado de: Fuente propia

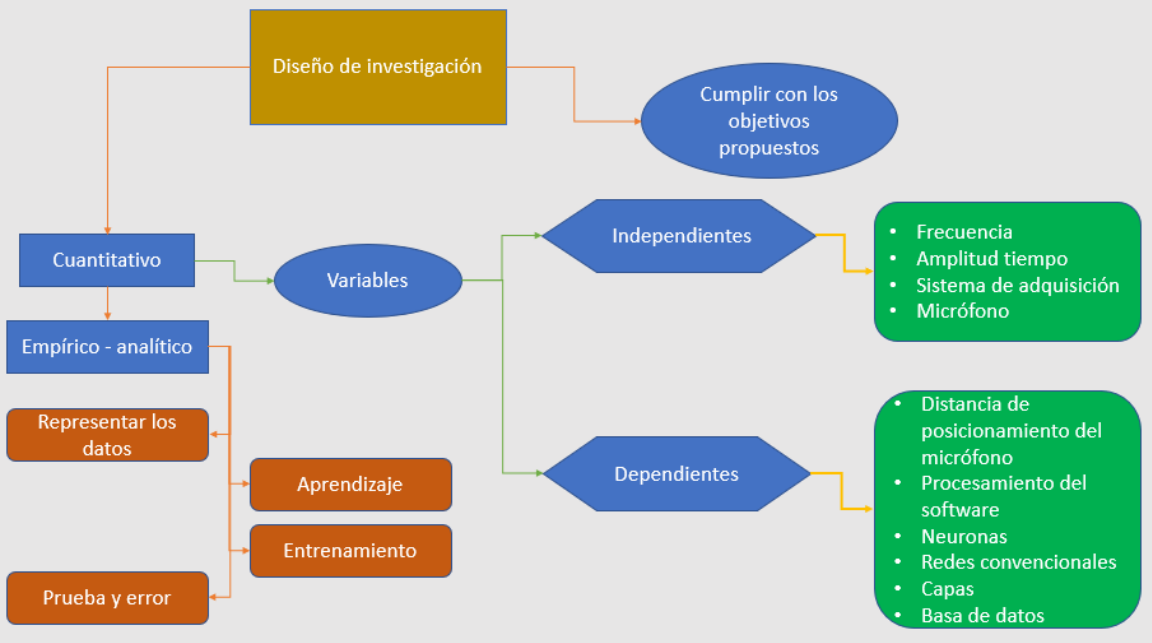


Figura 16. Diseño de la investigación

3.1. Variables independientes

- Frecuencia
- Amplitud
- Tiempo
- Sistema de adquisición
- Micrófono

3.2. Variables dependientes

- Distancia de posicionamiento del micrófono
- Procesamiento del software
- Neuronas
- Redes convolucionales
- Redes densamente conectadas
- Capas
- Base de datos

Capítulo 4. Desarrollo ingenieril

Para el desarrollo habrá 3 partes y cada una ellas corresponderán tanto a cada uno de los objetivos planteados como al proceso de realización de cada uno de ellos. La primera parte está basada en el primer objetivo específico: “Determinar el DataSet de señales impulsivas para el entrenamiento y evaluación del algoritmo”, en la segunda parte se implementan dos algoritmos basados en Deep Learning y en la última parte se presenta la evaluación de precisión (Accuracy) y pérdida de los algoritmos previamente entrenados con el 20% del DataSet para validación y el 80% para entrenamiento.

4.1. Determinar el DataSet

Se realizó la búsqueda del DataSet en distintas bases de datos que tuvieran diferentes sonidos de disparos de armas y calibres menores o iguales a 9 mm (el calibre máximo según lo planteado). El principal banco de datos se halló en una página llamada “The Gunshot Audio Forensics Dataset” desarrollada en Arizona, Estados Unidos en el verano de 2017, en la cual se encuentran sonidos de diferentes armas de fuego de distintos calibres. Teniendo en cuenta las limitaciones propuestas, se descargaron 2752 audios de disparos de armas de fuego de un calibre menor a 9mm (*Gunshot Audio Forensics Dataset - Gunshot Audio Analysis*, 2017). Estos sonidos de disparos fueron grabados con un micrófono zoom, un celular iPhone 7 y un Samsung Edge S7, todos los audios fueron grabados a una profundidad de 16 Bits, pero a diferentes frecuencias de muestreo. Un requerimiento que se buscaba en el DataSet es que todos los audios estuvieran en formato .wav ya que en este formato de audio no hay pérdida de información. Para realizar la clasificación binaria y multiclase se necesitó una base de datos con diferentes sonidos y con las mismas características de archivo (tipo de formato, frecuencia de muestreo y profundidad de bits) para que el algoritmo clasifique si es o

no un disparo de arma de fuego o que indique te tipo de sonido es el que ingresa. Se logró encontrar un DataSet de diferentes sonidos de un entrono común en la ciudad como: perros, pitos, aviones, carros, juegos pirotécnicos, frenadas bruscas, campanas, estrelladas, ambientes urbanos y otros más. En esa base de datos, se hallaron 2000 archivos de audio, donde todos están estandarizados formato .wav, frecuencia de muestreo de 44100 Hz, una duración de 5 segundos, en formato mono y una profundidad de 16 bits (Moreaux, 2018).

Tomado de: Fuente propia

Tipo de arma	Samsung Edge S7	Iphone 7	Mic. Zoom	
High standard sport king (0.22)	114	120	114	
S&W 34-1 (.22)	110	117	109	
Ruger 10/22 (0.22)	102	118	113	
Remingten 33 Bolt-Action	112	108	117	
Sig p225 (9 mm)	103	120	120	
Glock 19 (9mm)	115	124	120	Total
Lorcin 1380 (.380)	111	120	113	
S&W 10-8 (.380 PL)	103	114	118	2735

Tabla 1. Tabla de número de audios por arma y tipo de micrófono.

Ya teniendo DataSet tanto sonidos de disparos como sonidos normales, se buscó que todos tuvieran la misma duración (5 segundos) para que los algoritmos pudieran analizarlos sin ningún inconveniente. Los audios de menor duración se modificaron con silencios en un programa desarrollado en Google Colab para así dejar su tiempo de duración en 5 segundos. Como se explicó, los datos que tenían diferencia en las características de formato de audio son aquellos de disparos, los cuales no están estandarizados en frecuencia de muestreo, profundidad de bits y duración. Para eso, se desarrolló un algoritmo que permitiera cargar los audios a una frecuencia de muestreo y una

profundidad de bits específica y que fuera capaz de agregar silencio al audio para completar 5 segundos de muestra. En dado caso que la profundidad de bits fuera diferente, internamente, la librería usada realiza un proceso de Dither y transformación de frecuencia de muestreo sin perder calidad ni generar distorsión.

Tomado de: Fuente propia

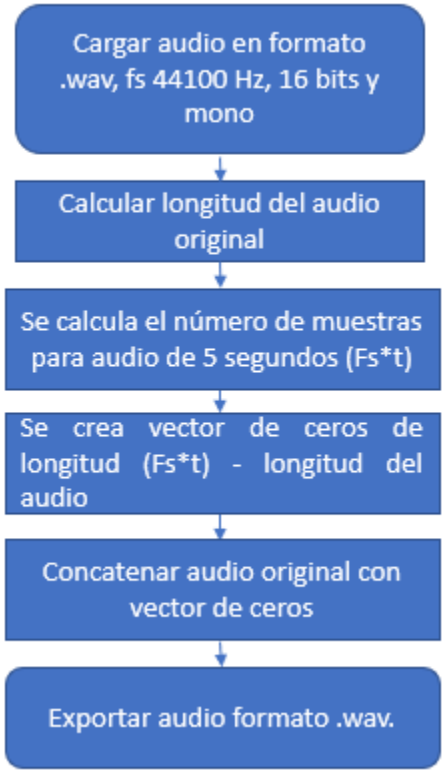


Figura 17. Diagrama de flujo de preprocesamiento de los audios

Teniendo en cuenta lo propuesto en los objetivos, se pretendió desarrollar dos algoritmos basados en Deep Learning, implementado una arquitectura de redes neuronales convolucionales donde el resultado fuera binario (es o no es un disparo) o en multiclase, buscando predecir qué tipo

de sonido era el de entrada. Para eso, la base de datos se dividió en dos carpetas (Binario y multiclase) teniendo en cuenta el tipo de clasificación.

4.1.1. Datos de clasificación binaria

Para la clasificación binaria, se separaron los datos teniendo en cuenta el resultado que se pretendió tener. Para este caso, se deseó saber si la señal de entrada era o no un disparo de arma de fuego. Como se explicó anteriormente en el marco teórico, para el aprendizaje supervisado, la intervención humana es necesaria para realizar la clasificación y nombrar la etiqueta de cada uno de los datos. En el momento en el que el algoritmo realizó el proceso de cálculos y secuencias específicas, arrojó un resultado de salida, teniendo en cuenta la etiqueta generada en el preprocesamiento de los datos. Para etiquetar los audios, se usó un comando de PowerShell en Windows, el cual, mediante una línea de código, permitió cambiar el nombre de todos los archivos de una carpeta por el deseado más un número consecutivo. Para los sonidos de disparos de arma de fuego, se usó la etiqueta "G_#" donde "G" significa "Gun" y "#" corresponde al número del archivo.

Tomado de: Fuente propia

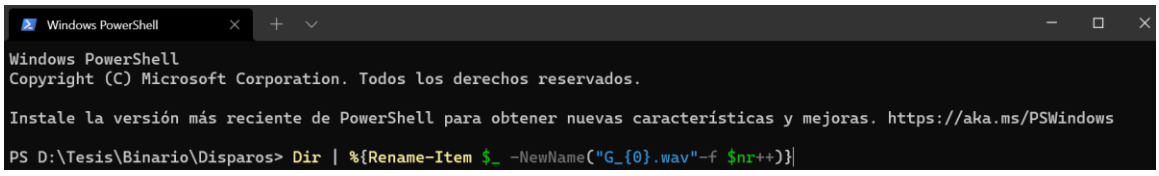


Figura 18. Código realizado para cambiar el nombre de los archivos de la carpeta

Para realizar la etiqueta de los audios que no son de disparos de arma de fuego, se realizó el mismo proceso aplicado a las etiquetas de las armas de fuego, pero cambiando el nombre de la etiqueta a “O_#”.

Tomado de: Fuente propia

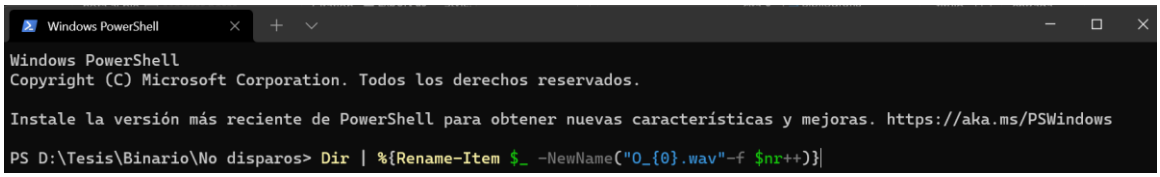


Figura 19. Código realizado para cambiar el nombre de los archivos de la carpeta

Finalmente, para la clasificación binaria se hicieron dos carpetas: la primera, llamada "Disparos" que comprende un conjunto de datos de 2735 audios en formato .wav de 5 segundos, a una frecuencia de muestreo de 44100 Hz, en formato mono y a una profundidad de 16 bits. La segunda, llamada "No disparos" en la que se encuentran 2300 audios en formato .wav, de 5 segundos, a una frecuencia de muestreo de 44100 Hz, en formato mono y a una profundidad de 16 bits.

Tomado de: Fuente propia

Sonidos Impulsivos	Sonido No Impulsivos
Disparos	Aspiradora
Fireworks	Bebe Llorando
Golpe de Puerta	Disparos
Pitos de Carro	Gallina
Teclado	Helicóptero
Tormenta	Motor
Vidrios Rotos	Oveja

Tabla 2. Clasificación de los datos algoritmo multiclase

Como se observa en la Tabla 2, en la clasificación "Sonidos No Impulsivos" se encuentran los sonidos de disparo de arma de fuego. Estos sonidos son impulsivos, pero se encuentran en la sección de los no impulsivos, ya que se quiere ver el comportamiento del algoritmo al comparar sonidos no impulsivos con otros que sí lo son.

4.1.2. Datos de clasificación multiclase

A diferencia de la clasificación binaria, en la clasificación multiclase se crearon diferentes carpetas según el contenido de los audios y la etiqueta que se le quiso atribuir a la salida del algoritmo. Para eso, en primer lugar, se separaron los audios según su sonido teniendo en cuenta la etiqueta de salida y el contenido del audio. Para conocer el contenido de cada uno de los 2000 audios descargados en Kaggle, dentro de la descripción, se encuentra un documento Excel que relaciona el nombre del archivo y la etiqueta de los audios teniendo en cuenta su sonido. A partir de esta información, fue posible crear diferentes carpetas y separar los audios. Finalmente, teniendo en cuenta su contenido, se nombraron los audios de la misma manera en la que se realizó para la clasificación binaria.

Tomado de: Fuente propia

Sonidos Impulsivos	Sonido No Impulsivos
Disparos	Aspiradora
Fireworks	Bebe Llorando
Golpe de Puerta	Disparos
Pitos de Carro	Gallina
Teclado	Helicóptero
Tormenta	Motor
Vidrios Rotos	Oveja

Tabla 3. Clasificación de los datos algoritmo multiclase

4.2. Algoritmo de clasificación binaria

4.2.1. Creación del archivo Json

Después de tener determinado todo el DataSet de todos los sonidos que son de disparos de armas de fuego y de los que no, se procedió a crear un archivo de extensión .json (JavaScript Object Notation) que es un formato de texto que permite el manejo e intercambio de datos de una manera simple y liviana, y que, además, contiene el diccionario, las etiquetas y los coeficientes de Mel (Busse et al., 2019). Este archivo es el encargado de tener toda la información necesaria para poder realizar la fase de entrenamiento y la predicción del modelo.

Para el desarrollo de los algoritmos, se usó el entorno de programación Google Colab desarrollado por Google, el cual permite ejecutar y programar lenguaje Python desde cualquier navegador web con las siguientes ventajas:

- Es gratuito
- No requiere procesamiento en hardware de nuestro computador
- Acceso a GPU y TPU para algoritmos de inteligencia artificial
- Permite compartir el libro de manera rápida y sencilla
- Usa las últimas versiones desarrolladas de cualquier librería

Como primer paso, se crearon dos carpetas en Google Drive teniendo en cuenta el algoritmo a desarrollar (Binario o Multiclase). En cada una de esas carpetas, se subieron los correspondientes audios ya separados y nombrados como se explica en la **sección 2.1.** para poder trabajar directamente desde Google Drive. Lo primero que se realizó fue un pequeño código que permitió que Google Colab tuviera la capacidad de cargar y usar datos específicos en una carpeta de Google Drive.

Tomado de: Fuente propia.

```
# Montar libreria para trabajar desde Google Drive
import numpy as np
from google.colab import drive
drive.mount("/content/gdrive")
ruta = ('gdrive/My Drive/Colab Notebooks/Tesis/Base de datos/'
'Base de datos (Binario)/Red neuronal convolucional binario/')
```

Figura 20. Código realizado en Google Colab para trabajar desde Google Drive

Teniendo la ruta de los audios y la capacidad de trabajar directamente desde Google Drive, se prosiguió a importar las librerías para crear el documento Json y poder manejar la información de cada uno de los audios. Ya habiendo importado cada una de las librerías, se nombró la ruta donde se encuentran los audios, se creó el nombre y la ubicación del archivo Json, la frecuencia de muestreo, la duración de los audios y se calculó el número total de datos que existe en cada uno de los audios.

Tomado de: Fuente propia.

```
import json
import os
import math
import librosa

DATASET_PATH = ruta
JSON_PATH = ('gdrive/My Drive/Colab Notebooks/Tesis/Base de datos/
'Base de datos (Binario)/Red neuronal convolucional binario/data_binario_10.json')
SAMPLE_RATE = 44100
TRACK_DURATION = 5
SAMPLES_PER_TRACK = SAMPLE_RATE * TRACK_DURATION
```

Figura 21. Creación de variables para creación de archivo Json

Se creó la función donde están los diccionarios, las etiquetas y los coeficientes MFCC dentro del archivo Json. Lo primero que realizó la función fue recorrer cada una de las carpetas que se encuentran dentro de la ruta especificada, en la que el nombre de cada una de las carpetas corresponde al diccionario que obtuvo el archivo Json, esto con el objetivo de extraer cada uno de los nombres de cada carpeta, siendo estos las etiquetas que usó el algoritmo para poder dar el resultado de salida.

Tomado de Fuente propia.

```
# Ingresa a los datos de cada una de las carpetas
for i, (dirpath, dirnames, filenames) in enumerate(os.walk(dataset_path)):


    if dirpath is not dataset_path:

        # Se generan las etiquetas teniendo en cuenta el diccionario
        semantic_label = dirpath.split("/")[-1]
        data["mapping"].append(semantic_label)
        print("\nProcesando: {}".format(semantic_label))
```

Figura 22. Recorrido de cada una de las carpetas para crear el diccionario

Después de que el algoritmo empezara a recorrer cada una de las carpetas, se cargaron cada uno de los archivos de audio existentes en cada una de las carpetas y la función realizó el cálculo del número de datos necesarios para segmentar el audio en 10 partes iguales y poder calcular los coeficientes MFCCS. Finalmente, se guardó el diccionario, los 10 datos de MFCCS de la segmentación de cada uno de los audios y las etiquetas asignadas.

Tomado de: Fuente propia.

```
                # Calcular MFCCS
                mfcc = librosa.feature.mfcc(signal[start:finish], sample_rate,
                                            n_mfcc=num_mfcc, n_fft=n_fft, hop_length=hop_length)
                mfcc = mfcc.T

                # Guardar los MFCCS
                if len(mfcc) == num_mfcc_vectors_per_segment:
                    data["mfcc"].append(mfcc.tolist())
                    data["labels"].append(i-1)
                    print("{}, segment:{}".format(file_path, d+1))

# Guardar los diccionarios, MFCCS y etiquetas en archivo Json
with open(json_path, "w") as fp:
    json.dump(data, fp, indent=4)
```

Figura 23. Cálculo de MFCCS y creación del archivo Json

Luego de crear y guardar el archivo con su diccionario, MFCCS y etiquetas, se graficó el número de datos existentes por cada una de las etiquetas creadas. Como se realizaron 10 segmentaciones por cada audio, al final de todo el proceso, se obtuvo 10 veces el número de audios de la base de datos.

Tomado de: Fuente propia.

```
import pandas as pd
import matplotlib.pyplot as plt
import collections, numpy

X, y = load_data(DATA_PATH)
plt.plot(y)
plt.title("Datos")
plt.xlabel("Numero de datos")
plt.ylabel("Etiqueta")
plt.show()

print('La etiqueta 0 hace referencia a los sonidos que no son disparos')
print('La etiqueta 1 hace referencia a los sonidos que son disparos')

print(collections.Counter(y))
```

Figura 24. Código para graficar la cantidad de datos

Tomado de: Fuente propia

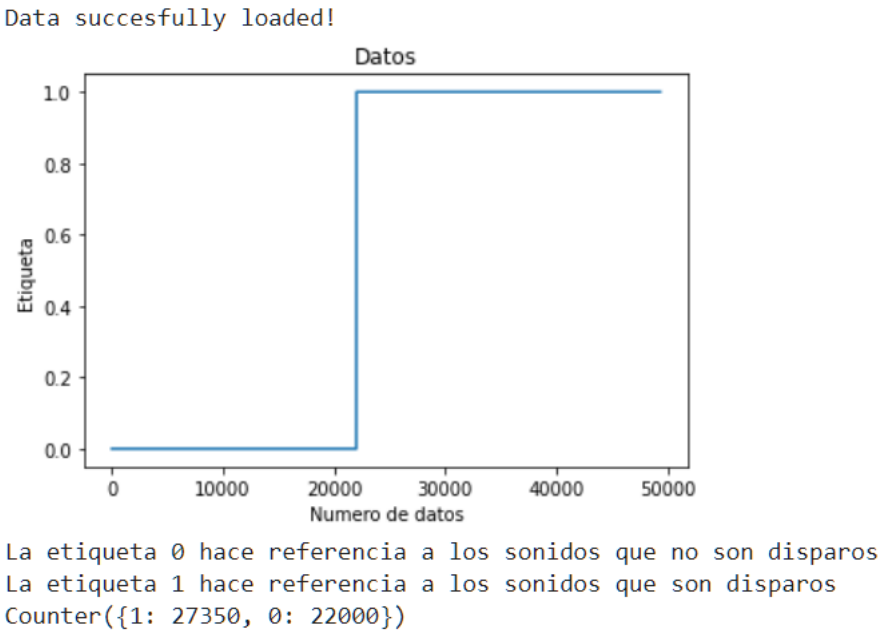


Figura 25. Grafica del número de datos existentes por cada una de las etiquetas

Como se observa en la Figura 24 se muestran el número de datos MFCCS creados y guardados en el archivo Json. Si estos se comparan con los datos de la Tabla 1, para los datos que sí son disparos, se tenían son 2735 audios y al momento de calcular los MFCCS se realizaron10 segmentaciones, entonces el número final de datos es de 27350.

Ya teniendo el diccionario, los valores de MFCC y las etiquetas de los datos generados guardados en el Json, se empezaron a importar los datos y a crear las capas, filtros, tamaño de los kernels y funciones de activación que permitieron que la red convolucional funcionara de manera adecuada y permitiera que el algoritmo generara una correcta predicción de nuevos datos teniendo en cuenta la precisión obtenida (Busse et al., 2019).

4.2.2. Funciones para realizar clasificación binaria.

Como se realizó en la creación del archivo Json, se importaron las librerías de *drive.mount* para poder trabajar los archivos directamente desde drive como se realizó en la imagen en la Figura 19, eliminando la última línea de código. Después de cargar las librerías para trabajar en Google Drive, se importaron las librerías para usar la GPU y se verificó la existencia de una GPU disponible para entrenar el algoritmo de manera mucho más rápida. Cuando se entrena un modelo de Deep Learning, se deben realizar muchas operaciones para poder calcular las diferentes variables que nos permiten llegar a un modelo optimo. Las CPU es capaz de realizar una a una cada operación necesaria mediante la ALU (Unidad Lógica Aritmética) que es la encargada de realizar sumar, restas, multiplicaciones y divisiones, a diferencia de la GPU que está diseñada en su hardware para realizar más operaciones de manera simultánea (Sotaquirá Gutiérrez, 2018).

Tomado de: Fuente propia.

```
import tensorflow as tf
from tensorflow import keras
print("GPU disponible: ", tf.config.list_physical_devices('GPU'))
print("Versión de TensorFlow: ", tf.__version__)
print("Versión de Keras: ", tf.keras.__version__)
```

```
GPU disponible:  []
Versión de TensorFlow:  2.6.0
Versión de Keras:  2.6.0
```

Figura 26. Verificación de GPU disponible

Después de verificar la disponibilidad para el uso de la GPU, se importó el archivo json, el cual contiene el diccionario, las etiquetas y los valores MFCC.

Tomado de: Fuente propia.

```
import json

DATA_PATH = ('gdrive/My Drive/Colab Notebooks/Tesis/Base de datos/'
'Base de datos (Binario)/Red neuronal convolucional binario/data_binario_10.json')

with open(DATA_PATH, "r") as fp:
  data = json.load(fp)

X = np.array(data["MFCC"])
y = np.array(data["Etiqueta"])
z = np.array(data["Diccionario"])
```

Figura 27. Importación del archivo json

Al tener cargados los datos del archivo json separados en diferentes variables, se creó una función que permitiera separar los datos de entrenamiento, de prueba y de validación. Para trabajar los datos en las capas convolucionales, los datos se operan sobre tensores ("generalización de vectores y matrices y se entiende fácilmente como una matriz multidimensional" (Machine Learning Mastery, 2018a)) 3D, llamados mapas de características (*feature maps)* en los cuales, se encontró el número de datos, número de MFCC por segmento, número de MFCC y el último dato que se agrega, que es la profundidad de los filtros (como se ve en las imágenes, las capas RGB) (Depth).

Tomado de: Fuente propia.

```
def prepare_datasets(test_size, validation_size):
    # Crear datos de test, train y validación
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=test_size)
    X_train, X_validation, y_train, y_validation = train_test_split(X_train,
                                            y_train, test_size=validation_size)

    # Agregamos un nuevo vector para crear Depth necesario en las CNN
    X_train = X_train[..., np.newaxis]
    X_validation = X_validation[..., np.newaxis]
    X_test = X_test[..., np.newaxis]
    return X_train, X_validation, X_test, y_train, y_validation, y_test
```

Figura 28. Función para preparar el DataSet

Después de tener preparado el DataSet, se creó la función que permitió realizar las predicciones de las muestras de entrenamiento y audios nuevos que el algoritmo nunca había visto ni analizado. Esto, permitiendo saber si el sonido que se carga y analiza es o no un sonido de disparo de arma de fuego teniendo en cuenta el modelo y los pesos del algoritmo.

Tomado de: Fuente propia

```
def predict(model, X, y):
    global prediction
    # Se agrega un nuevo vector el cual permite que la función de model.predict
    # de queras acepte las variables, ya que acepta una matriz de 4D
    X = X[np.newaxis, ...]
    # Función para realizar la predición
    prediction = model.predict(X)

    if prediction <0.5:
      prediction = 0
    else:
      prediction = 1
    print("Etiqueta real: {}, Etiqueta predicha: {}".format(y, (prediction)))

    return prediction
```

Figura 29. Función para la predicción de las muestras.

Como se observa en la Figura 28, se crearon dos condicionales *if* que tienen como función aproximar los valores calculados en la predicción a los valores 0 o 1. En el primer condicional la variable tiene que ser menor a 0.5 para determinar que el valor corresponde a la etiqueta 0 así como cuando la variable es mayor a 0.5, la etiqueta equivale a 1. Esos límites se establecen porque la última capa de la red neuronal tiene una función de activación *sigmoide* (la más usada para clasificación binaria).

Para verificar si el algoritmo tenía o no Overfitting, se graficó el historial de la precisión del algoritmo y la función de pérdida con los datos de entrenamiento y los de prueba. Estas gráficas

muestran la diferencia que existe entre la precisión y la Función de pérdida; si existe una gran separación entre la precisión y la Función de pérdida, significa que el algoritmo está sufriendo de Overfitting.

Fuente: Fuente propia

```
import matplotlib.pyplot as plt

def plot_history(history):
    fig, axs = plt.subplots(2)
    # Se crea plot de accuracy
    axs[0].plot(history.history["accuracy"], label="Presición de entrenamiento")
    axs[0].plot(history.history["val_accuracy"], label="Presición de test")
    axs[0].set_ylabel("Accuracy")
    axs[0].legend(loc="lower right")
    axs[0].set_title("Valor de presición")

    # Se crea plot de valor de perdida
    axs[1].plot(history.history["loss"], label="Error de entrenamiento")
    axs[1].plot(history.history["val_loss"], label="Error de test")
    axs[1].set_ylabel("Error")
    axs[1].set_xlabel("Epoch")
    axs[1].legend(loc="upper right")
    axs[1].set_title("Valor de perdida")

    plt.show()
```

Figura 30. Función para graficar Accuracy y función de pérdida.

Para poder evaluar la precisión del algoritmo, y como se planteó en el tercer objetivo específico, esta validación se realiza con el 20% del total de los datos. La matriz de confusión, como se explicó anteriormente, muestra de manera gráfica el rendimiento del modelo para observar que tan bien se comporta el algoritmo. En la Figura 29, se muestra cómo se genera la matriz de confusión. Se debe tener en cuenta que los datos deben tener un formato específico, para eso, se usa el comando *to_categorical* el cual convierte el valor entero en una matriz numérica que tiene valores binarios y columnas iguales al número de categorías. Antes de convertir las etiquetas de

números en enteros, en una matriz numérica; se realizó la predicción del modelo con los que se crearía la matriz de confusión. De la misma manera en la que se realizó en la función de predicción, se realizaron dos condicionales que permitieran obtener valores enteros para las etiquetas de las predicciones realizadas por el modelo.

Tomado de: Fuente propia.

```
from tensorflow.keras.utils import Sequence, to_categorical

y_pred_ = model.predict(X_test, use_multiprocessing=True, workers=6, verbose=1)
y_test_ = to_categorical(y_test, num_classes=2)
y_new_pred = np.zeros(len(y_pred_))
h = 0
for i in y_pred_:
  z = y_pred_[h]

  if z<0.5:
    z=0
  if z>0.5:
    z=1

  y_new_pred[h] = z
  h = h+1
```

Figura 31. Predicción para graficar la matriz de confusión

Tomado de: Fuente propia

```
from sklearn.metrics import confusion_matrix


def plot_confusion_matrix(y_true, y_pred, labels):
    matrix = confusion_matrix(y_true, y_pred)
    fig, ax = plt.subplots(figsize=(12,10))
    plt.imshow(matrix)
    ax.set_xticks(range(len(labels)));
    ax.set_xticklabels(labels, rotation=0)
    ax.set_yticks(range(len(labels)));
    ax.set_yticklabels(labels)
    max_confusions = 0
    confused_classes = (-1, -1)
    for i, true_label in enumerate(matrix):
        for j, predicted_label in enumerate(true_label):
            text = ax.text(j, i, matrix[i, j], fontweight='bold',
                           ha="center", va="center", color="#9a5833", fontsize = 'xx-large');
    plt.tick_params(axis=u'both', which=u'both',length=0)
    plt.title("Matriz de Confusión");
```

Figura 32. Función para crear la matriz de confusión.

Ya con todas las funciones necesarias para cargar los datos, se tuvo que realizar las diferentes predicciones que ayudarían a verificar el funcionamiento de la red y las fases de entrenamiento, predicción y validación. Luego, se creó la función para construir la red convolucional que sería la encargada de entrenar el modelo y calcular los pesos necesarios para realizar la predicción de los datos de entrada.

4.2.3. Creación del modelo de clasificación binaria con redes convolucionales

Para crear la red convolucional, lo primero que se debe tener en cuenta es el número de capas que se desea tener, ya que, al aumentar el número de capas, la red tarda un poco más en el procesamiento y las operaciones que se necesitan calcular. Para el caso de esta red convolucional, se optó por crear tres capas convolucionales, una capa *Flatten* la cual permite pasar de un tensor de 3D a un tensor de 1D. Después de implementar la capa de *Flatten*, se agregó una capa densamente

conectada. Finalmente, se añadió la capa de salida, la cual permitiría realizar la clasificación binaria; razón por la cual, la capa de salida tuvo que ser de una neurona.

Tomado de: Fuente propia

```
from sklearn.model_selection import train_test_split
from tensorflow.keras.layers import Dropout, BatchNormalization

def build_model(input_shape):
    model = keras.Sequential()

    # Primera capa convolucinal.
    model.add(keras.layers.Conv2D(32, (5, 5), activation='relu', input_shape=input_shape))
    model.add(keras.layers.MaxPooling2D((5, 5), strides=(2, 2), padding='same'))
    model.add(keras.layers.BatchNormalization())

    # Segunda capa convolucional.
    model.add(keras.layers.Conv2D(64, (3, 3), activation='relu'))
    model.add(keras.layers.MaxPooling2D((3, 3), strides=(2, 2), padding='same'))
    model.add(keras.layers.BatchNormalization())

    # Tercera capa convolucional
    model.add(keras.layers.Conv2D(32, (2, 2), activation='relu'))
    model.add(keras.layers.MaxPooling2D((2, 2), strides=(1, 1), padding='same'))
    model.add(keras.layers.BatchNormalization())

    # Capa Flatten y capa densamente conectada.
    model.add(keras.layers.Flatten())
    model.add(keras.layers.Dense(64, activation='relu'))
    model.add(keras.layers.Dropout(0.25))

    # Capa de salida binaria
    model.add(keras.layers.Dense(1, activation='sigmoid'))

    return model
```

Figura 33. Creación del modelo de una red neuronal convolucional

Como se observa en la Figura 32, la configuración de las tres capas convolucionales es exactamente igual, lo que cambia son los parámetros que permiten que la precisión del algoritmo sea mayor o menor. Tener muchas capas o filtros en el modelo, no hace que se obtenga una precisión mayor. Tener una mala configuración del modelo puede generar problemas de Overfitting

y que la precisión no sea tan alta. Lo primero realizado en el modelo, fue la importación de las librerías que permitirían separar los datos de entrenamiento de los datos de test, teniendo en cuenta los porcentajes que se asignaran. También, se tuvo que importar la librería que permite usar funciones como el Dropout o el BatchNormalization que ayudan a mejorar la precisión del algoritmo y evitar problemas de Overfitting.

Tomado de: Fuente propia

```
# Primera capa convolucinal.
model.add(keras.layers.Conv2D(32, (5, 5), activation='relu', input_shape=input_shape))
model.add(keras.layers.MaxPooling2D((5, 5), strides=(2, 2), padding='same'))
model.add(keras.layers.BatchNormalization())
```

Figura 34. Configuración de una capa convolucional

En cualquiera de las capas, lo primero que se hizo fue crear la capa convolucional. Teniendo ya creada la capa, se definió el número de filtros que permitirían extraer las características más relevantes de los datos a trabajar; en este caso, los coeficientes de Mel se calcularon y se guardaron en el archivo Json. En el caso de esta red neuronal, cada neurona de la capa fue conectada a una pequeña región de 5 x 5 neuronas (25 neuronas en total) de la capa de entrada (input_shape) la cual consta de una matriz de 44, 13, 1; que consiste en el número de MFCCS calculados por segmento (40), número total de MFCCS calculados y la profundidad. Teniendo en cuenta lo anterior, la ventana de tamaño de 5 x 5 recorre toda la capa de entrada de 44 x 13. Esta ventana se desliza a lo largo de toda la capa de neuronas de entrada. La ventana empieza a deslizarse desde la parte superior izquierda de la matriz.

Tomado de: Fuente propia.

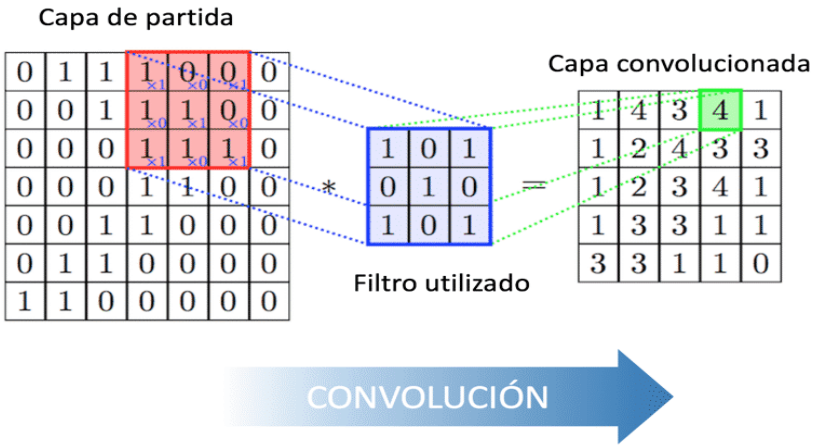


Figura 35. Entrada de 7 x 7 píxeles con una ventana de 3 x 3 crea un espacio de 5 x 5 neuronas en la capa oculta.

La capa convolucionada se calcula mediante la siguiente formula:

$$(m - f + 1)\, X\, (n - f + 1) \quad (6)$$

Donde *m* es el número de columnas de la capa de entrada, *n* el número de filas y *f* el tamaño del filtro usado. Teniendo la formula anterior, el cálculo del tamaño de la capa convolucionada de la Figura 34 sería el siguiente:

$$(7 - 3 + 1)\, X\, (7 - 3 + 1) \quad (7)$$

$$5X\ 5$$

Lo anterior significa que la capa convolucionada es de 5 x 5 lo que es correcto teniendo en cuenta la Figura 34. Para la red convolucional que se está creando, se tiene en la primera capa un filtro de tamaño de 5 x 5 y una capa de entrada de 44 x 13, lo que significa que la capa convolucionada es del siguiente tamaño:

$$(44 - 5 + 1)\, X\, (13 - 5 + 1) \quad (8)$$

$$40X\ 9$$

Tomado de: Fuente propia.

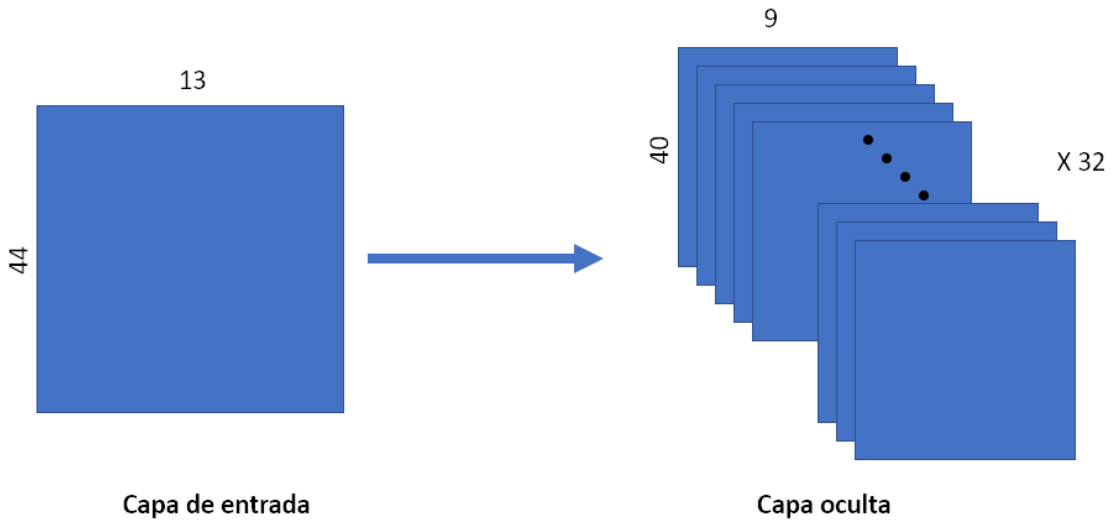


Figura 36. Capa convolucional compuesta por 32 filtros y una capa de entrada de 44 x 13

Después de tener claro el número de filtros que se usarían y el ventaneo, se definió la función de activación que sirve para introducir la no linealidad en las capacidades de modelado de la red.

Luego de agregar la capa convolucional y todas sus variables, se agregó la capa de Pooling que como los filtros kernel, a partir de la capa de salida de la capa convolucional (40 x 9), se definió el tamaño de la ventana que es de (5 x 5) como se muestra en la Figura 33. Esta ventana recorre la capa de entrada y selecciona el valor máximo que se encuentra en otra ventana que se estableció en la capa de salida convolucional. A diferencia de la capa convolucional, en la capa de Pooling se define el *stride*, que indica el número de pasos que se mueve la ventana de los filtros. Valores grandes de *stride* hacen decrecer el tamaño de la información que pasa a la siguiente capa. Finalmente, en la capa de Pooling se agrega el Padding que añade ceros a la matriz con el objetivo de que la dimensión de filas y columnas en la matriz sea la misma a la salida y a la entrada. Estos ceros se agregan de manera uniforme en la izquierda, derecha, arriba y abajo.

Tomado de: Fuente propia.

```
model.add(keras.layers.MaxPooling2D((5, 5), strides=(2, 2), padding='same'))
```

Figura 37. Capa MaxPooling

La capa de *BatchNormalization* tiene como propósito normalizar las entradas de la capa, de tal manera que tengan una activación de salida media de cero y la desviación estándar de uno. Esto es análogo a cómo se estandarizan las entradas de las redes.

Tomado de: Fuente propia.

```
model.add(keras.layers.BatchNormalization())
```

Figura 38. Capa de Batch Normalization

Como se ve en la Figura 32, se crean 3 capas convolucionales de manera idéntica a la primera, pero cambiando sus parámetros. Seguido, se crea una capa de *flatten*, una densamente conectada y una capa de DropOut. Como se explicó anteriormente, la capa *flatten* hace pasar de un tensor de 3D a uno de 1D. Después de tener los tensores en 1D, se implementa una capa densamente conectada enfocada en realizar la clasificación de los datos. La capa de Dropout ayuda a mitigar el sobreajuste del modelo. Esta capa se basa en ignorar ciertos conjuntos de neuronas de la red durante la fase de entrenamiento de manera aleatoria. Con “ignorar” se refiere a que algunas neuronas no se consideran durante una iteración concreta del proceso de aprendizaje.

Tomado de: Fuente propia.

```
# Capa Flatten y capa densamente conectada.
model.add(keras.layers.Flatten())
model.add(keras.layers.Dense(64, activation='relu'))
model.add(keras.layers.Dropout(0.25))
```

Figura 39. Capas Flatten y densamente conectada

Finalmente, después de implementar toda la red que permite filtrar la información, extraer características y ajustar el modelo, se creó la capa de salida la cual está conformada por una capa densamente conectada, donde la neurona correspondiente es el número de categorías que se desean clasificar. Para la clasificación binaria, se usa una neurona en esta última capa y para clasificar más de una categoría, se usa el número de categorías que se tienen. La función de activación que se usa es la sigmoide que, como se mencionó anteriormente, es la mejor y la más apropiada para realizar clasificación binaria.

Tomado de: Fuente propia.

```
# Capa de salida binaria
model.add(keras.layers.Dense(1, activation='sigmoid'))
```

Figura 40. Capa de salida

Después de haber definido todas las funciones que se usarían y de crear toda la red convolucional, se ejecutó el código que permitiría llamar cada una de ellas y que sean ejecutadas. Como primera instancia, se le indica a Google Colab que ejecutara el código con la GPU, lo que hizo que el tiempo de entrenamiento fuera menor. Después de determinar qué herramienta de hardware se usaría, se definió el tamaño que se utilizaría para realizar el entrenamiento y la validación. Para este caso, se estableció que se usaría el 25% de los datos para el entrenamiento

(Tabla 1), que es un numero recomendado para realizar en entrenamiento del algoritmo (Busse et al., 2019), mientras que solo se utilizaría el 20% de los datos para realizar la validación, como se planteó en el tercer objetivo específico.

Tomado de: Fuente propia.

```
if __name__ == "__main__":
  with tf.device('/device:GPU:0'):

    # Separación de los datos de entrenamiento, validación y test
    X_train, X_validation, X_test, y_train, y_validation, y_test = prepare_datasets(0.25, 0.20)
    # Creación de la red
    input_shape = (X_train.shape[1], X_train.shape[2], X_train.shape[3])
    model = build_model(input_shape)

    # Compilación del modelo
    optimiser = keras.optimizers.Adam(learning_rate=0.0001)
    model.compile(optimizer=optimiser,
                  loss='binary_crossentropy',
                  metrics=['accuracy'])
    model.summary()

    # Entrenamiento del modelo
    history = model.fit(X_train, y_train, validation_data=(X_validation, y_validation), epochs=100)

    # Graficación de accuracy y función de perdida
    plot_history(history)

    # Evaluación del modelo
    test_loss, test_acc = model.evaluate(X_test, y_test, verbose=2)
    print('\nAccuracy de datos de test:', test_acc)
```

Figura 41. Código para correr todo el algoritmo

Después de separar los datos de validación con los datos de test, se definió el tamaño de la capa de entrada a la red. Para eso, se usó el comando *shape* que especifica la forma en la que está constituida una matriz. El *input shape* está definido principalmente por tres parámetros: el número de MFCCS calculados por segmento en el vector (44), el número MFCCS (13) y la profundidad (1). Ya teniendo definido el tamaño de la capa de entrada, se agregaron esas variables a la función *build_model* (Figura 40). Con las variables de entradas requeridas por la red, fue necesario compilar

el modelo con el optimizador Adam con una tasa de aprendizaje de 0.0001, la función de pérdida de entropía cruzada binaria y la métrica que se evalúa (Accuracy), entre más pequeño sea el valor de la tasa de aprendizaje, más tarda en entrenar el algoritmo y puede generar problemas de Overfitting (Torres, 2020). Con el método *summary*, se puede ver la información de los parámetros y del formato de los tensores de salida de cada capa:

Tomado de: Fuente propia.

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_3 (Conv2D)           (None, 40, 9, 32)         832

 max_pooling2d_3 (MaxPooling  (None, 20, 5, 32)        0
 2D)

 batch_normalization_3 (Batc  (None, 20, 5, 32)        128
 hNormalization)

 conv2d_4 (Conv2D)           (None, 18, 3, 64)         18496

 max_pooling2d_4 (MaxPooling  (None, 9, 2, 64)         0
 2D)

 batch_normalization_4 (Batc  (None, 9, 2, 64)         256
 hNormalization)

 conv2d_5 (Conv2D)           (None, 8, 1, 32)          8224

 max_pooling2d_5 (MaxPooling  (None, 8, 1, 32)         0
 2D)

 batch_normalization_5 (Batc  (None, 8, 1, 32)         128
 hNormalization)

 flatten_1 (Flatten)         (None, 256)               0

 dense_2 (Dense)             (None, 64)                16448

 dropout_1 (Dropout)         (None, 64)                0

 dense_3 (Dense)             (None, 1)                 65

=================================================================
Total params: 44,577
Trainable params: 44,321
Non-trainable params: 256
```

Figura 42. Información de la red neurona convolucional.

Después de visualizar la información de la red, se empezó a entrenar la red con los datos de entrenamiento y se realizó la validación del funcionamiento de esta con los datos de validación y 100 Epoch. Finalmente, se graficaron las variables de Accuracy y función de pérdida de todo el entrenamiento.

Tomado de: Fuente propia.

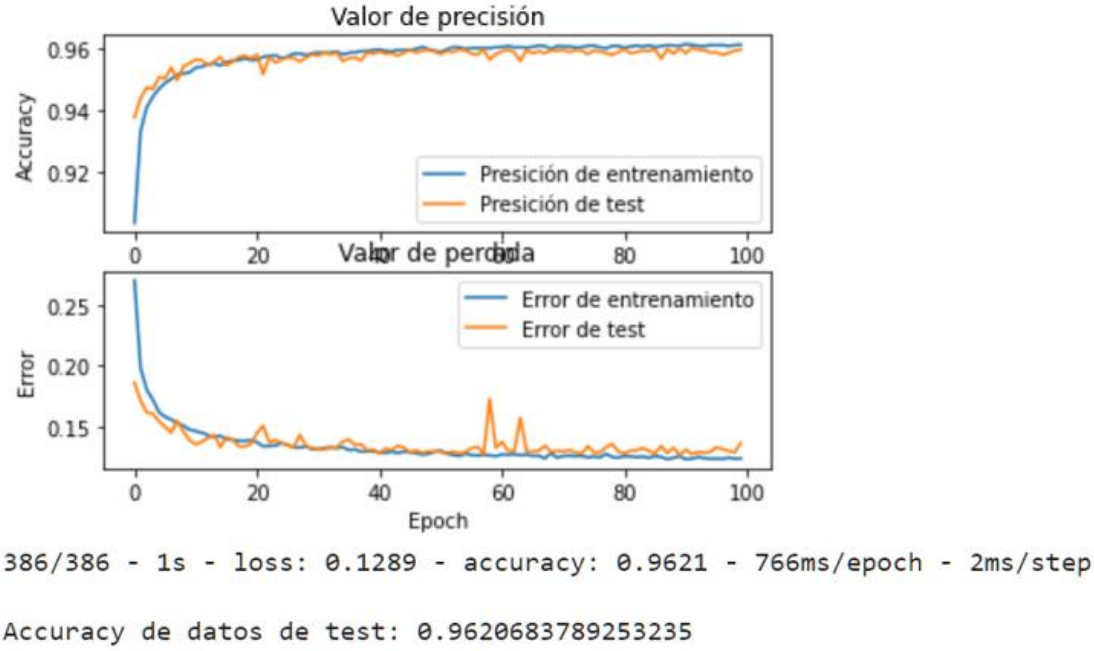


Figura 43. Historial de entrenamiento

Como se observa en la Figura 42, el Accuracy del modelo con la configuración de la Figura 42 es de 96.2% y una función de pérdida de 0.12. También, si se compara el error que generó la red neuronal creada con la Figura 5, se observa que no existen problemas de Overfitting.

Después de conocer el comportamiento del algoritmo y la precisión que se está obteniendo, se realizó la gráfica de la matriz de confusión para ver el comportamiento de la red con los datos. Lo

primero que se realizó fue la predicción y la adaptación de los datos, como se muestra en la Figura 30. Ahora una demostración de la adaptación de los datos de las etiquetas:

Tomado de: Fuente propia.

```
from tensorflow.keras.utils import Sequence, to_categorical

etiqueta = [0, 1]
etiqueta_categorical = to_categorical(etiqueta, num_classes=2)
print("Para la etiqueta ", etiqueta[0],",el categorical es: ", etiqueta_categorical[0])
print("Para la etiqueta ", etiqueta[1],",el categorical es: ", etiqueta_categorical[1])
```

```
Para la etiqueta  0 ,el categorical es:  [1. 0.]
Para la etiqueta  1 ,el categorical es:  [0. 1.]
```

Figura 44. Demostración función to_categorical

Finalmente, para graficar la matriz de confusión se realizó una predicción del modelo y se ingresaron las etiquetas para realizar la gráfica:

Tomado de: Fuente propia.

```
# Matriz de confusión
import collections

y_pred = (y_new_pred)
y_true = np.argmax(y_test_, axis=1)
labels = ['No disparos', 'Disparos']
plot_confusion_matrix(y_true, y_pred, labels)
print(collections.Counter(y_test))
```

Figura 45. Variables para graficar matriz de confusión

Como resultado se obtiene lo siguiente:

Fuente: Fuente propia.

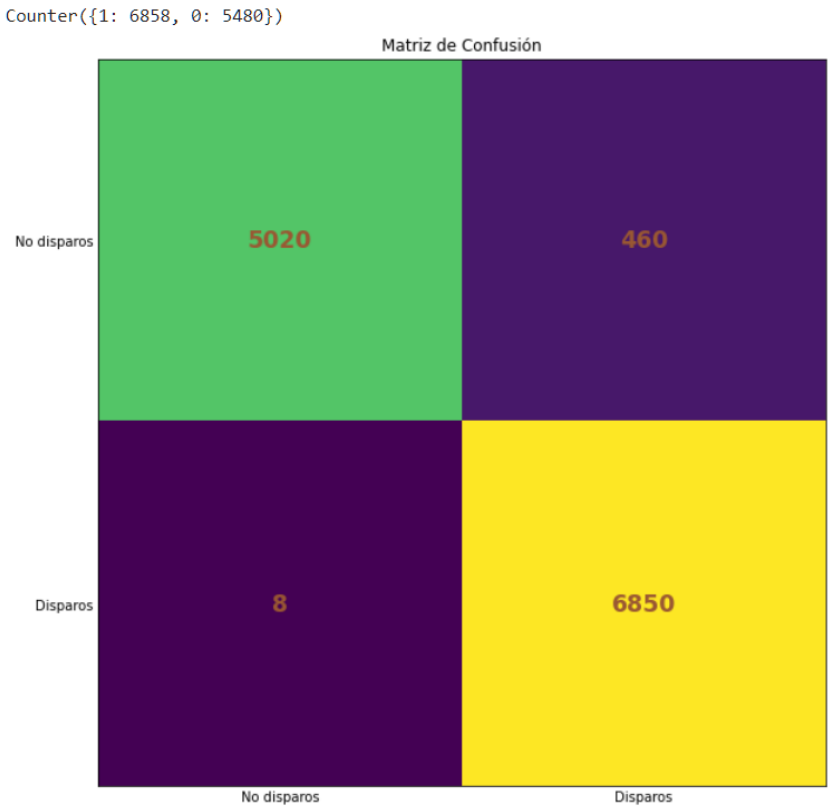


Figura 46. Matriz de confusión datos de validación

Teniendo en cuenta la matriz de confusión de la Figura 45, el algoritmo clasificó los disparos como disparos con un 99.88% de precisión, mientras los que no son disparos, los clasificó de manera correcta con el 91.60% de los datos.

4.3. Algoritmo de clasificación multiclase

Para realizar el algoritmo de clasificación multiclase, se tomaron muchas de las funciones y códigos realizados en la clasificación binaria, ya que la estructura de estos algoritmos es muy similar, a excepción de ciertos parámetros y variables que son específicas para la clasificación de varias categorías.

4.3.1. Creación del archivo Json

Para la creación del archivo Json, se usó el mismo algoritmo que se muestra en la **sección 2.2.1** a diferencia de la ruta y nombre donde se guardó el archivo. También se debe tener en cuenta que al ser una clasificación multiclase, fue necesario agregar las otras características a clasificar. Estas categorías y número de datos se encuentran a detalle en la Tabla 2. En la Figura 46, se observa una representación grafica de los datos cargados para realizar la clasificación multiclase

Fuente: Fuente propia

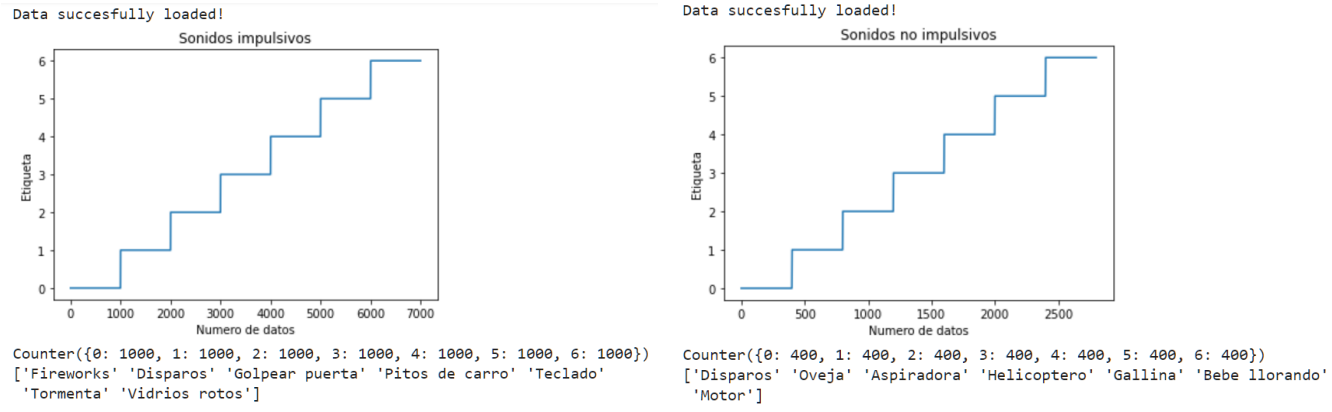


Figura 47. Datos de clasificación multiclase.

4.3.2. Funciones para realizar clasificación multiclase.

Como se dijo anteriormente, para realizar la clasificación multiclase usando redes neuronales convolucionales se usaron la mayoría de las funciones ya mostradas y explicadas en la clasificación binaria. Como primer paso, se importaron los datos de Google Colab, como se realizó en la Figura 26, teniendo en cuenta que la creación del archivo Json para la clasificación multiclase tiene otra ruta y otro nombre. Después de cargar el DataSet, se creó la función que permitiría crear la capa de

profundidad necesaria para las redes neuronales convolucionales y la que permitiría separar el 20% de los datos de validación para evaluar el algoritmo (Figura 27).

Para la predicción del algoritmo de clasificación multiclase la función es totalmente diferente a la que se realiza en la clasificación binaria, ya que en la clasificación multiclase no se tienen dos categorías que se puedan resolver con una función que en su salida sea 0 o 1.

Tomado de: Fuente propia.

```
def predict(model, X, y):

    # Se agrega un nuevo vector el cual permite que la función de model.predict
    # de queras acepte las variables, ya que acepta una matriz de 4D
    X = X[np.newaxis, ...]

    # Función para realizar la predición
    prediction = model.predict(X)

    # Obtiene el mayor valor del vector.
    predicted_index = np.argmax(prediction, axis=1)

    print("Etiqueta real: {}, Etiqueta predicha: {}".format(y, (predicted_index)))
```

Figura 48. Función de predicción clasificación multiclase

Si se compara la predicción de la clasificación binaria (Figura 28) y la clasificación multiclase (Figura 47), en la clasificación multiclase no se necesita colocar condicionales ya que la función *np.argmax*, que devuelve los índices de los valores máximos a lo largo del vector, para el caso, sería la etiqueta que el algoritmo predice.

Igual que en la clasificación binaria, para realizar la matriz de confusión, lo primero que se debe tener en cuenta es la predicción del algoritmo con los datos de test. También se debe tener en cuenta la conversión de las etiquetas con la función *to_categorical*. Al tener más de 2 clases, se debe cambiar el número de clases.

Tomado de: Fuente propia.

```
from tensorflow.keras.utils import Sequence, to_categorical

y_pred_ = model.predict(X_test, use_multiprocessing=True, workers=6, verbose=1)
y_test_ = to_categorical(y_test, num_classes=7)
```

Figura 49. Adaptación de los datos para crear la matriz de confusión

4.3.3. Creación del modelo de clasificación binaria con redes convolucionales

Después de definir todas las funciones, se crea el modelo:

Tomado de: Fuente propia.

```
from sklearn.model_selection import train_test_split
from tensorflow.keras.layers import Dropout, BatchNormalization
def build_model(input_shape):
    model = keras.Sequential()

    # Primera capa convulucional
    model.add(keras.layers.Conv2D(350, (5, 5), activation='sigmoid', input_shape=input_shape))
    model.add(keras.layers.MaxPooling2D((2, 2), strides=(1, 1), padding='same'))
    model.add(keras.layers.BatchNormalization())
    model.add(Dropout(0.5))

    # Segunda capa convulucional
    model.add(keras.layers.Conv2D(68, (5, 5), activation='sigmoid'))
    model.add(keras.layers.MaxPooling2D((3, 3), strides=(1, 1), padding='same'))
    model.add(keras.layers.BatchNormalization())

    # Tercera capa convulucional
    model.add(keras.layers.Conv2D(42, (3, 3), activation='sigmoid'))
    model.add(keras.layers.MaxPooling2D((2, 2), strides=(1, 1), padding='same'))
    model.add(keras.layers.BatchNormalization())

    # Segunda capa convulucional
    model.add(keras.layers.Conv2D(16, (3, 3), activation='sigmoid'))
    model.add(keras.layers.MaxPooling2D((2, 2), strides=(1, 1), padding='same'))
    model.add(keras.layers.BatchNormalization())
    model.add(Dropout(0.1))

    # Capa Flatten y capa densamente conectada
    model.add(keras.layers.Flatten())
    model.add(keras.layers.Dense(16, activation='sigmoid'))

    # Capa de salida
    model.add(keras.layers.Dense(7, activation='softmax'))

    return model
```

Figura 50. Modelo de clasificación multiclase

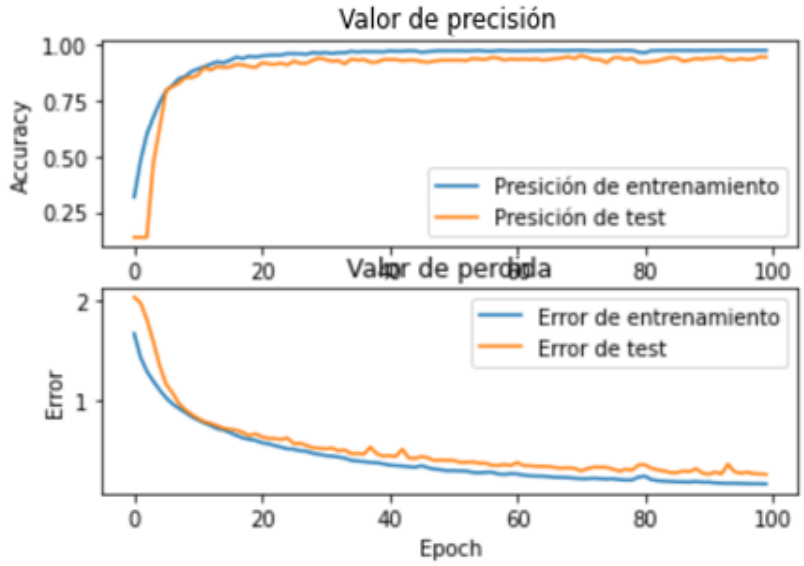
A diferencia de la clasificación binaria, la capa de salida está conformada por 7 neuronas, que es el número de clases a analizar y clasificar. Como se explicó, la función de activación *Softmax* es la función más optimizada y la más usada para realizar clasificación multiclase.

Fuente: Fuente propia.

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 40, 9, 350)        9100
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 40, 9, 350)        0
_________________________________________________________________
batch_normalization (BatchNo (None, 40, 9, 350)        1400
_________________________________________________________________
dropout (Dropout)            (None, 40, 9, 350)        0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 36, 5, 68)         595068
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 36, 5, 68)         0
_________________________________________________________________
batch_normalization_1 (Batch (None, 36, 5, 68)         272
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 34, 3, 42)         25746
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 34, 3, 42)         0
_________________________________________________________________
batch_normalization_2 (Batch (None, 34, 3, 42)         168
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 32, 1, 16)         6064
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 32, 1, 16)         0
_________________________________________________________________
batch_normalization_3 (Batch (None, 32, 1, 16)         64
_________________________________________________________________
dropout_1 (Dropout)          (None, 32, 1, 16)         0
_________________________________________________________________
flatten (Flatten)            (None, 512)               0
_________________________________________________________________
dense (Dense)                (None, 16)                8208
_________________________________________________________________
dense_1 (Dense)              (None, 7)                 119
=================================================================
Total params: 646,209
Trainable params: 645,257
Non-trainable params: 952
```

Figura 51. Información de la red neurona convolucional.

Tomado de: Fuente propia.




```
22/22 - 0s - loss: 0.2813 - accuracy: 0.9386 - 140ms/epoch - 6ms/step

Test accuracy: 0.9385714530944824
```

Figura 52. Historial de entrenamiento

Como se observa en la Figura 51, el algoritmo tiene una buena precisión y no sufre de problemas de Overfitting.

Tomado de: Fuente propia.

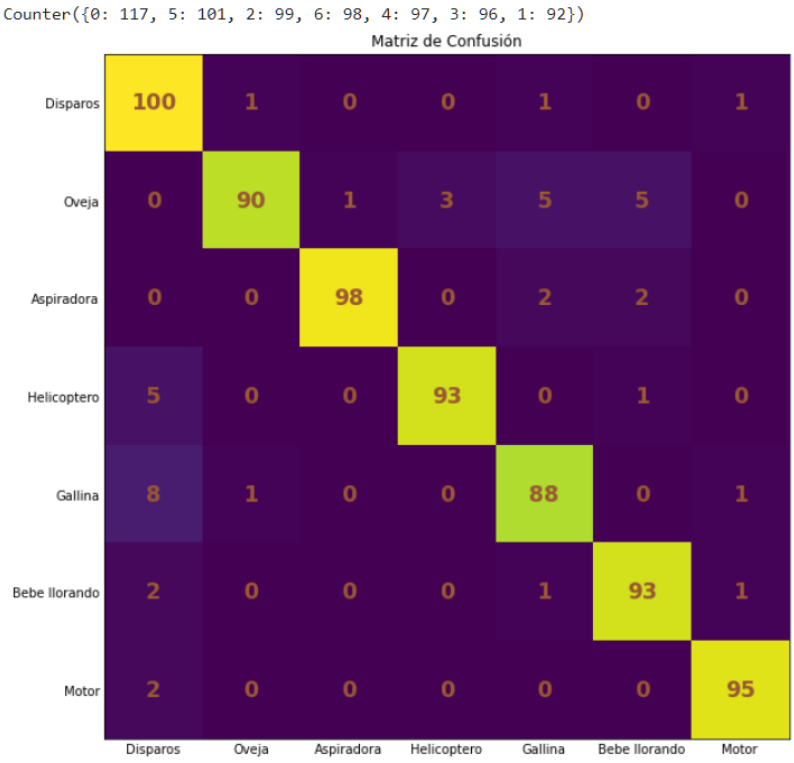


Figura 53. Matriz de confusión clasificación multiclase.

El algoritmo se comporta de una manera óptima ya que no sufre de Overfitting, tiene una precisión de más del 90% y clasifica los disparos de arma de fuego con un 85.47% de precisión.

Capítulo 5. Análisis y presentación de resultados.

5.1. Accuracy y funciones de pérdida

A continuación, se muestran las gráficas de Accuracy y de función de pérdida mediante el historial que se creó a partir de la función de la Figura 29. Se muestran siete modelos, los cuales comprenden clasificación binaria con una configuración de 100 y 200 Epoch para ver el comportamiento del algoritmo al aumentar los Epoch. Para la misma clasificación binaria, se muestra el comportamiento con otra configuración cambiando las variables (Filtros, padding, Strides, funciones de activación, numero de capas) de la red neuronal y con 100 Epoch. Para la clasificación multiclase, se muestran cuatro configuraciones. La primera consta de sonidos impulsivos con una configuración de red neuronal convolucional de 100 Epoch. La segunda consta de la misma configuración con la misma cantidad de repeticiones, pero con sonidos no impulsivos. Las otras dos configuraciones constan de dos redes a las cuales se les modificaron los filtros y se entrenaron con 100 Epoch.

5.1.1. Clasificación binaria

5.1.1.1. Primer modelo

El primer modelo consta con los siguientes parámetros en la red convolucional:

Tomado de: Fuente propia.

```
from sklearn.model_selection import train_test_split
from tensorflow.keras.layers import Dropout, BatchNormalization

def build_model(input_shape):
    model = keras.Sequential()

    # Primera capa convolucinal.
    model.add(keras.layers.Conv2D(32, (5, 5), activation='relu', input_shape=input_shape))
    model.add(keras.layers.MaxPooling2D((5, 5), strides=(2, 2), padding='same'))
    model.add(keras.layers.BatchNormalization())

    # Segunda capa convolucional.
    model.add(keras.layers.Conv2D(64, (3, 3), activation='relu'))
    model.add(keras.layers.MaxPooling2D((3, 3), strides=(2, 2), padding='same'))
    model.add(keras.layers.BatchNormalization())

    # Tercera capa convolucional
    model.add(keras.layers.Conv2D(32, (2, 2), activation='relu'))
    model.add(keras.layers.MaxPooling2D((2, 2), strides=(1, 1), padding='same'))
    model.add(keras.layers.BatchNormalization())

    # Capa Flatten y capa densamente conectada.
    model.add(keras.layers.Flatten())
    model.add(keras.layers.Dense(64, activation='relu'))
    model.add(keras.layers.Dropout(0.25))

    # Capa de salida binaria
    model.add(keras.layers.Dense(1, activation='sigmoid'))

    return model
```

Figura 54. Primer modelo.

El primer modelo tiene una configuración de tres capas convolucionales con dos capas de neuronas densamente conectadas. Este modelo se entrenó con 100 Epoch. Los resultados de Accuracy y de función de pérdida dieron de la siguiente manera:

Fuente: Fuente propia.

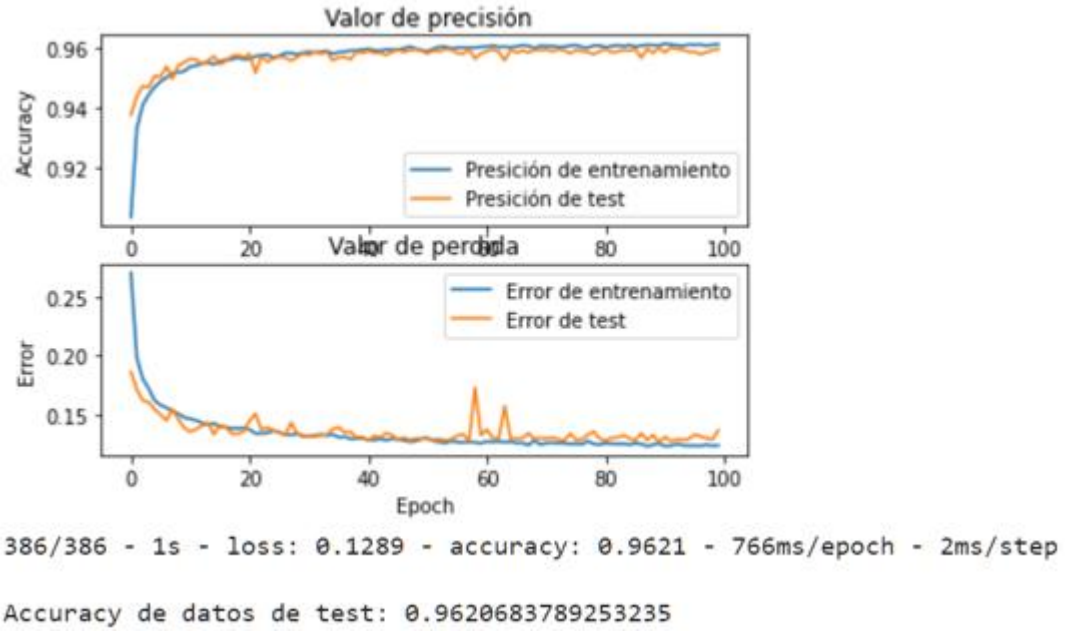


Figura 55. Accuracy y función de pérdida del primer modelo.

Como se observa en la Figura 54, la configuración tomada del primer modelo tiene un Accuracy de un 96.2%. La configuración de la red convolucional no está generando Overfitting ni Underfitting, ya que los datos de entrenamiento no se alejan en ningún momento de una manera drástica a lo largo de todo el entrenamiento. En la función de pérdida del Epoch 58, se observa que existe un pequeño pico en ese momento y se debe a que al momento en que el algoritmo realizó la predicción, usó un dato que fue muy similar y no logró identificarlo bien, entonces no tuvo un buen valor de perdida al momento de predecir. También es importante revisar la matriz de confusión y ver cómo realizó la predicción de los datos.

Tomado de: Fuente propia.

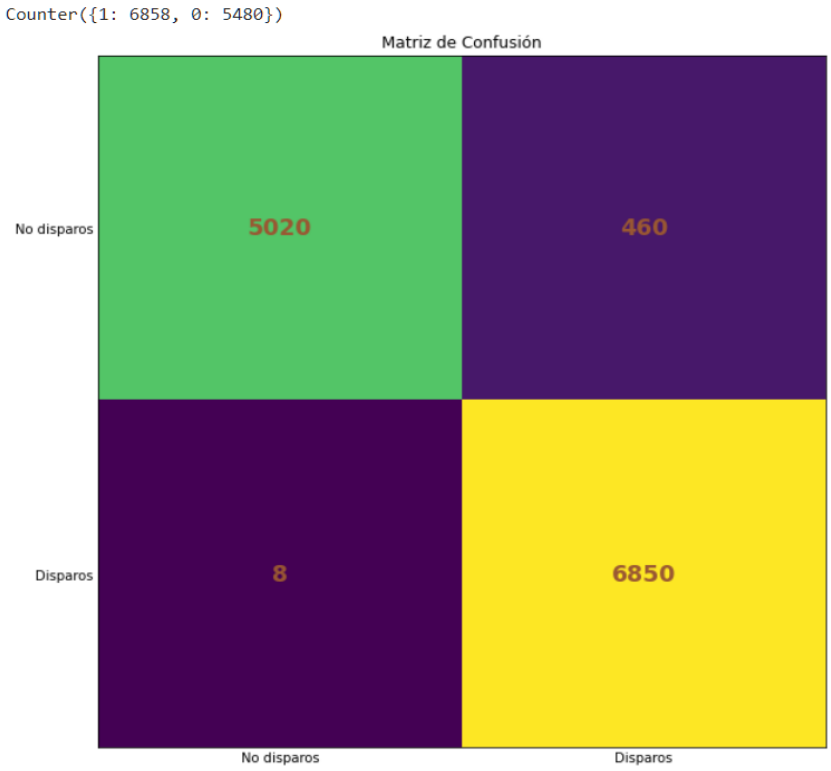


Figura 56. Matriz de confusión del primer modelo datos de validación.

Al momento de calcular el porcentaje de los datos de sonidos que son disparos y que predijo como disparos, da un 99.88% de precisión.

5.1.1.2. Segundo modelo

Ahora se tiene el mismo modelo con las mismas variables, pero cambiando el número de Epoch por 200, ya que muchas veces, el aumentar la precisión va de la mano con la cantidad de Epoch.

Tomado de: Fuente propia.

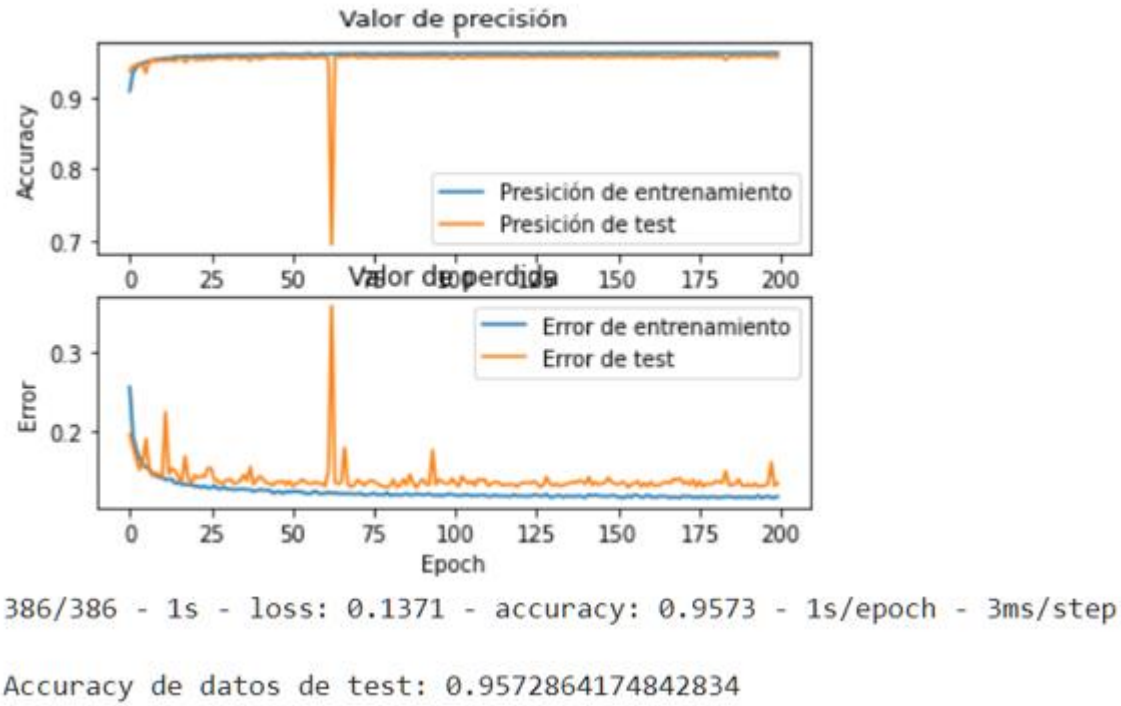


Figura 57. Accuracy y Función de pérdida segundo modelo.

Si se compara la Figura 53 con la Figura 55, existe mayor diferencia entre el error de test y el de entrenamiento al momento de agregar más Epoch. También se puede ver que aumentar el número Epoch no hace necesariamente que aumente el Accuracy. En este caso, tener más repeticiones de entrenamiento generó que la precisión fuera menor en un 0.035%, lo cual es algo muy insignificante, sin embargo, se debe tener en cuenta que el algoritmo realizó el doble de repeticiones.

Tomado de: Fuente propia.

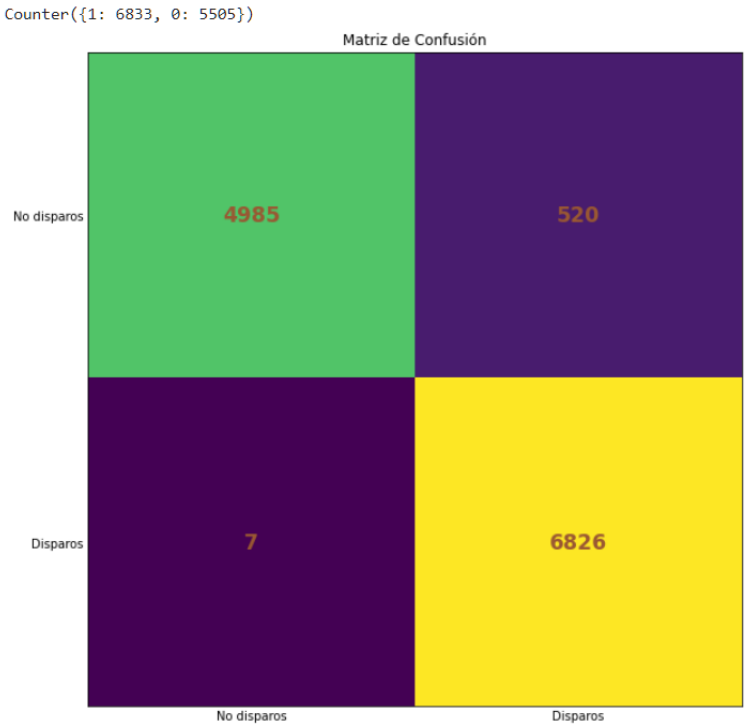


Figura 58. Matriz de confusión del segundo modelo

Al momento de calcular el porcentaje de los datos de sonidos que son disparos y que predijo como disparos, da un 99.89% de precisión, es decir 0.01% más a comparación del primero; aunque sigue siendo muy insignificante, el segundo modelo se entrenó el doble de veces.

$$Porcentaje = \frac{Numero\ de\ datos * 100\%}{Total\ de\ los\ datos}$$

$$Porcentaje = \frac{6826 * 100\%}{6833} = 99{,}89\%$$

5.1.1.3. Tercer modelo

Para el tercer modelo, se aumentó el número de filtros de cada una de las capas, ya que se pensaría que aumentar el número de filtros (Figura 58) generaría que el algoritmo fuera capaz de

identificar cambios más pequeños y eso haría que fuera más preciso. La configuración de esta red es la siguiente:

Tomado de: Fuente propia.

```
def build_model(input_shape):
    model = keras.Sequential()

    # Primera capa convolucinal.
    model.add(keras.layers.Conv2D(512, (5, 5), activation='sigmoid', input_shape=input_shape))
    model.add(keras.layers.MaxPooling2D((5, 5), strides=(2, 2), padding='same'))
    model.add(keras.layers.BatchNormalization())

    # Segunda capa convolucional.
    model.add(keras.layers.Conv2D(1024, (3, 3), activation='sigmoid', input_shape=input_shape))
    model.add(keras.layers.MaxPooling2D((3, 3), strides=(2, 2), padding='same'))
    model.add(keras.layers.BatchNormalization())

    # Tercera capa convolucional
    model.add(keras.layers.Conv2D(64, (2, 2), activation='sigmoid', input_shape=input_shape))
    model.add(keras.layers.MaxPooling2D((2, 2), strides=(1, 1), padding='same'))
    model.add(keras.layers.BatchNormalization())

    # Capa Flatten y capa densamente conectada.
    model.add(keras.layers.Flatten())
    model.add(keras.layers.Dense(128, activation='sigmoid'))
    model.add(keras.layers.Dropout(0.25))

    # Capa de salida binaria
    model.add(keras.layers.Dense(1, activation='sigmoid'))

    return model
```

Figura 59. Tercer modelo

El tercer modelo, con muchos más filtros por cada capa, genera que existan más parámetros que el algoritmo tiene que calcular, lo que se traduce en mayor tiempo de procesamiento.

Tomado de: Fuente propia.

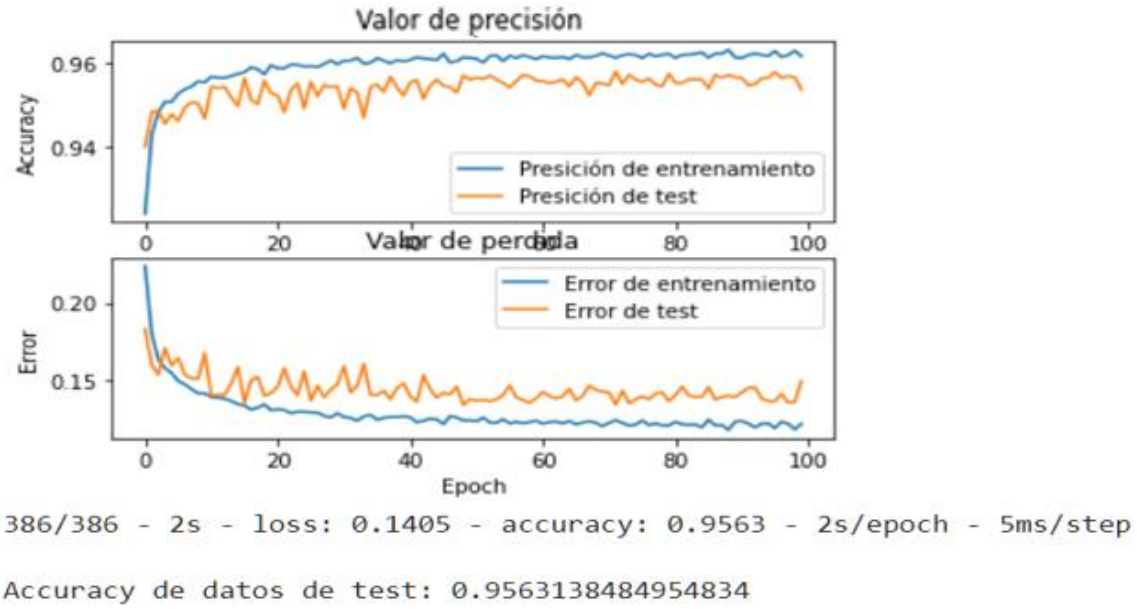


Figura 60. Accuracy y función de pérdida del tercer modelo

Como se observa en la Figura 58, aumentar el número de filtros de cada una de las capaz no genera que el Accuracy aumente, pero sí hace que la diferencia entre el Accuracy de los datos de entramiento y la función de pérdida sea mayor, ocasionando que, al momento de realizar la predicción, se disminuya un poco la precisión y empiece a verse un comportamiento de Overfitting.

Tomado de: Fuente propia.

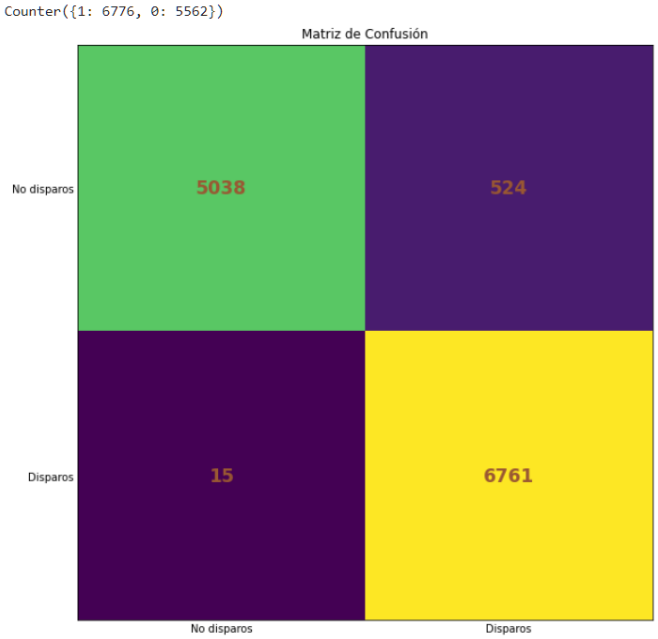


Figura 61. Matriz de confusión del tercer modelo datos de validación

Al momento de calcular el porcentaje de los datos de sonidos que son disparos y que predijo como disparos, da un 99.77% de precisión. Esto muestra que, al existir una mayor diferencia entre la función de pérdida, la precisión disminuye un poco.

5.1.2. Clasificación multiclase

En la clasificación multiclase, se tienen dos algoritmos que se diferencian por la base de datos de sonidos que las conforman: la primera que tiene en su base de datos sonidos impulsivos y la otra que está conformada por sonidos no impulsivos (Figura 46). Gracias a esto, se verá el comportamiento del Accuracy que les corresponde.

5.1.2.1. Primer modelo

Con sonidos impulsivos

El primer modelo para analizar será una red neuronal conformada por cuatro capas convolucionales y dos capas densamente conectadas. Las clases que conforman este algoritmo son: Fuegos artificiales, Disparos de arma de fuego, golpear puerta, pitos de carro, teclado, tormenta, vidrios rotos.

Tomado de: Fuente propia.

```
def build_model(input_shape):
    model = keras.Sequential()

    # Primera capa convulucional
    model.add(keras.layers.Conv2D(350, (5, 5), activation='sigmoid', input_shape=input_shape))
    model.add(keras.layers.MaxPooling2D((2, 2), strides=(1, 1), padding='same'))
    model.add(keras.layers.BatchNormalization())
    model.add(Dropout(0.5))

    # Segunda capa convulucional
    model.add(keras.layers.Conv2D(68, (5, 5), activation='sigmoid'))
    model.add(keras.layers.MaxPooling2D((3, 3), strides=(1, 1), padding='same'))
    model.add(keras.layers.BatchNormalization())

    # Tercera capa convulucional
    model.add(keras.layers.Conv2D(42, (3, 3), activation='sigmoid'))
    model.add(keras.layers.MaxPooling2D((2, 2), strides=(1, 1), padding='same'))
    model.add(keras.layers.BatchNormalization())

    # Segunda capa convulucional
    model.add(keras.layers.Conv2D(16, (3, 3), activation='sigmoid'))
    model.add(keras.layers.MaxPooling2D((2, 2), strides=(1, 1), padding='same'))
    model.add(keras.layers.BatchNormalization())
    model.add(Dropout(0.1))

    # Capa Flatten y capa densamente conectada
    model.add(keras.layers.Flatten())
    model.add(keras.layers.Dense(16, activation='sigmoid'))

    # Capa de salida
    model.add(keras.layers.Dense(7, activation='softmax'))

    return model
```

Figura 62. Primer modelo multiclase.

Tomado de: *Fuente propia.*

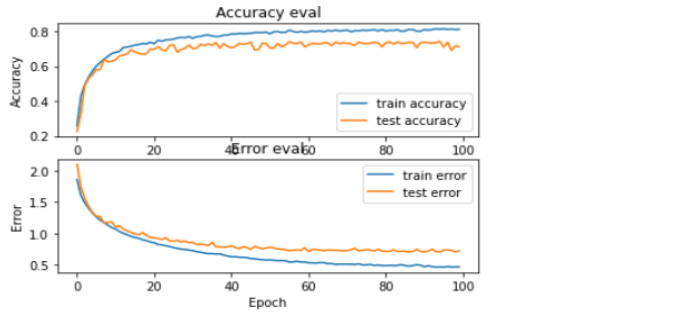


Figura 63. Valores de Accuracy y Función de pérdida sonidos impulsivos

Después de realizar el entrenamiento, la precisión del algoritmo de sonidos impulsivos donde se encuentran 7 categorías es de 74%, lo que es bajo, pero como se mostrará más adelante, para los datos que el algoritmo está comparando tiene un buen comportamiento

Tomado de: Fuente propia.

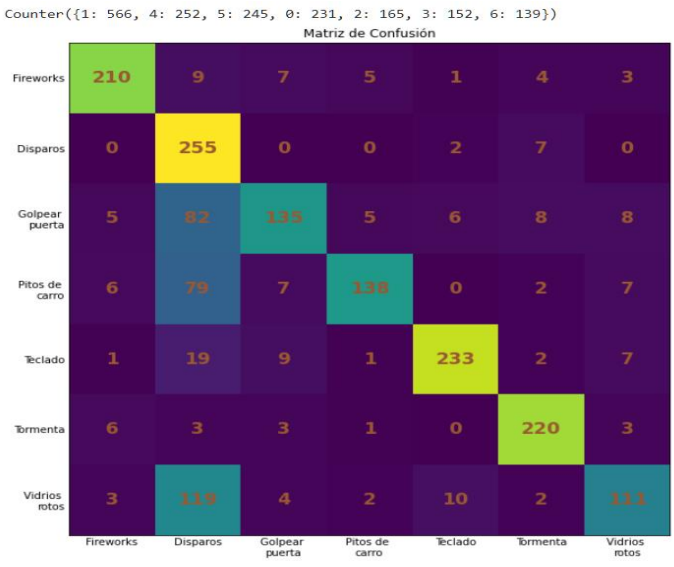


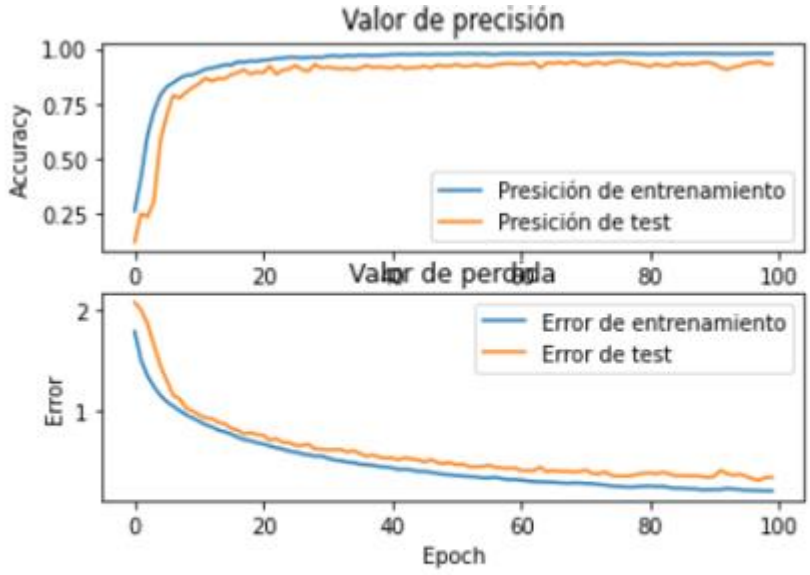
Figura 64. Matriz de confusión sonidos impulsivos

En este modelo, la precisión del algoritmo no es tan buena como lo ha sido con los anteriores, esto se debe a que los datos que se comparan son datos que tienen características similares en tiempo. Este modelo es capaz de identificar los disparos de manera correcta con un 45.05% de precisión y los confunde con un 21.02% con vidrios rotos.

5.1.2.2. Segundo modelo

Ahora, implementemos el mismo modelo anterior, pero usando sonidos que no son impulsivos conformados por: Disparos, ovejas, aspiradora, helicóptero, gallina, bebe llorando y motor de vehículo. Usando la misma cantidad de capas, filtros y Epoch, se obtiene la siguiente precisión y Función de pérdida.

Tomado de: Fuente propia.




```
22/22 - 0s - loss: 0.3231 - accuracy: 0.9300 - 141ms/epoch - 6ms/step

Test accuracy: 0.9300000071525574
```

Figura 65. Accuracy y Función de pérdida segundo modelo sonidos no impulsivos

Al tener la misma red y cantidad de repeticiones con las que el algoritmo se entrenó, pero cambiando los datos a audios no impulsivos, se ve que el comportamiento del Accuracy es mucho mayor y la separación de los datos de entrenamiento y de test no son tan alejados el uno del otro. Al comparar los audios disparos de armas de fuego con otros sonidos que tiene una duración mayor a un segundo, la precisión del mismo modelo aumenta un 18.4% con respecto al primer modelo.

Tomado de: Fuente propia.

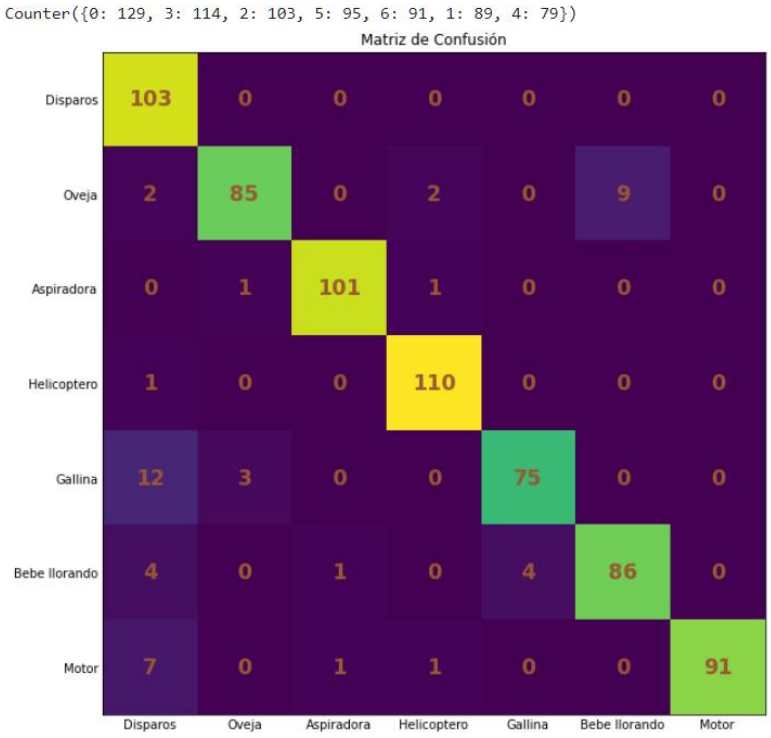


Figura 66. Matriz de confusión segundo modelo sonidos no impulsivos

Para modelo donde se encuentran sonidos que no son impulsivos, la precisión de la predicción es mayor, ya que el algoritmo es capaz de identificar el 79.84% de disparos de armas de fuego. Otra

cosa que se puede notar es que se tiene que los sonidos de motor tienen una precisión de un 100% y en el primer modelo es de 79.85%, lo que significa una mejora del 20.14%.

5.1.2.3. Tercer modelo

Ahora se le realiza una modificación en las neuronas a la red convolucional para ver el comportamiento que toma, pero disminuyendo el número de filtros.

El modelo queda de la siguiente manera:

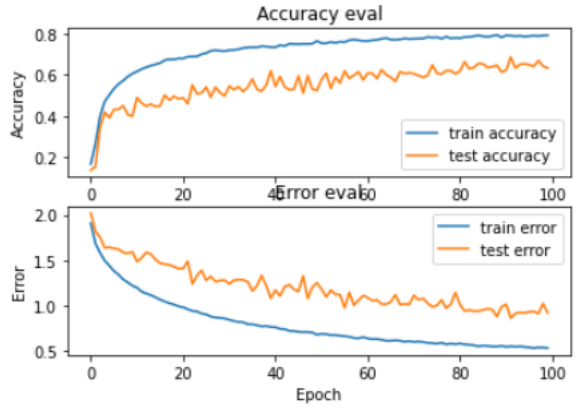
Tomado de: Fuente propia.

```
from sklearn.model_selection import train_test_split
from tensorflow.keras.layers import Dropout, BatchNormalization
def build_model(input_shape):
    model = keras.Sequential()

    # Primera capa convulucional
    model.add(keras.layers.Conv2D(32, (5, 5), activation='sigmoid', input_shape=input_shape))
    model.add(keras.layers.MaxPooling2D((2, 2), strides=(1, 1), padding='same'))
    model.add(keras.layers.BatchNormalization())
    model.add(Dropout(0.5))

    # Segunda capa convulucional
    model.add(keras.layers.Conv2D(128, (5, 5), activation='sigmoid'))
    model.add(keras.layers.MaxPooling2D((3, 3), strides=(1, 1), padding='same'))
    model.add(keras.layers.BatchNormalization())

    # Tercera capa convulucional
    model.add(keras.layers.Conv2D(28, (3, 3), activation='sigmoid'))
    model.add(keras.layers.MaxPooling2D((2, 2), strides=(1, 1), padding='same'))
    model.add(keras.layers.BatchNormalization())

    # Segunda capa convulucional
    model.add(keras.layers.Conv2D(16, (3, 3), activation='sigmoid'))
    model.add(keras.layers.MaxPooling2D((2, 2), strides=(1, 1), padding='same'))
    model.add(keras.layers.BatchNormalization())
    model.add(Dropout(0.1))

    # Capa Flatten y capa densamente conectada
    model.add(keras.layers.Flatten())
    model.add(keras.layers.Dense(16, activation='sigmoid'))

    # Capa de salida
    model.add(keras.layers.Dense(7, activation='softmax'))

    return model
```

Figura 67. Tercer modelo para clasificación multiclase sonidos impulsivos

Como en la clasificación binaria, tener un número menor de filtros en la capa aumento la precisión del algoritmo, pero eso no aplica para todos los casos:

Tomado de: Fuente propia.




```
55/55 - 0s - loss: 0.9414 - accuracy: 0.6309 - 161ms/epoch - 3ms/step

Test accuracy: 0.6308571696281433
```

Figura 68. Accuracy y Función de pérdida de tercer modelo de sonidos impulsivos

Para este caso, disminuir el número de neuronas no aumenta ni mejora la precisión, Por lo contrario, disminuye la precisión y empieza a generar problemas de Overfitting.

Tomado de: Fuente propia.

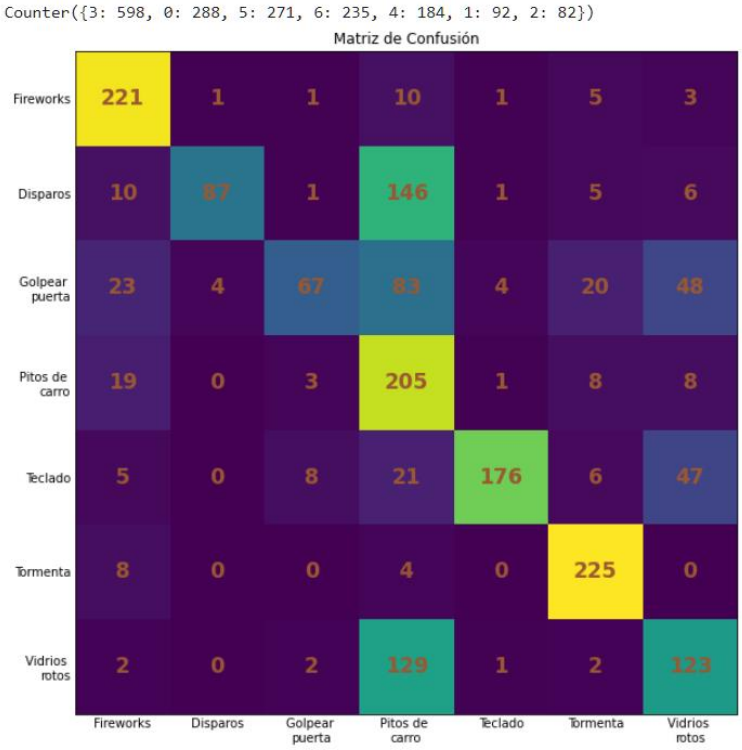
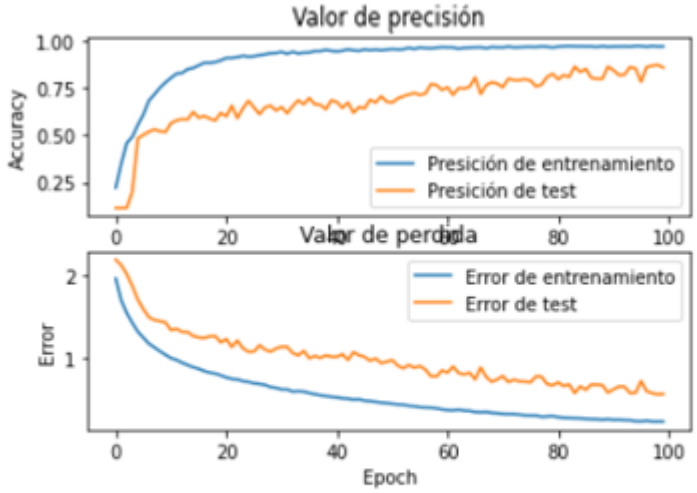


Figura 69. Matriz de confusión del tercer modelo sonidos impulsivos

5.1.2.4. Cuarto modelo

Por último, se implementa el modelo anterior, pero con sonidos no impulsivos para revisar el comportamiento que toma la precisión y la matriz de confusión

Tomado de: Fuente propia.




```
22/22 - 0s - loss: 0.5496 - accuracy: 0.8500 - 72ms/epoch - 3ms/step

Test accuracy: 0.8500000238418579
```

Figura 70. Accuracy y Función de pérdida cuarto modelo

Tomado de: Fuente propia.

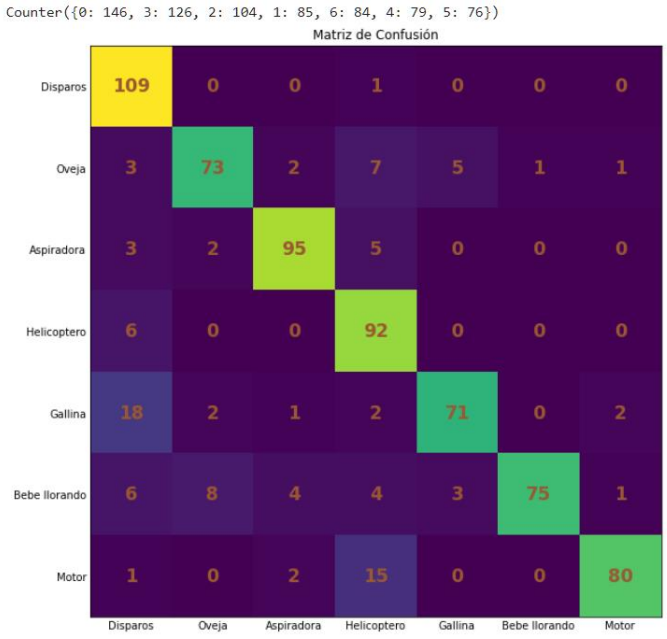


Figura 71. Matriz de confusión cuarto modelo.

Para el cuarto modelo, como se observa en las gráficas, la precisión del algoritmo es menor a la que se puede observar en el segundo modelo (**sección 5.1.2.2**) donde la precisión tiene una precisión de un 93%, lo que significa que tiene una precisión mayor en un 8%. Esto se puede ver en la Figura 70, calculando el porcentaje de la predicción de sonidos de disparos el cual predijo que manera correcta 109 que equivalen a un 74% y donde el primer modelo tiene una precisión de un 79.84%, lo cual es más de un 4% en el aumento de la predicción. El cuarto modelo con comparación del primero, en la Función de pérdida, los datos tienen un comportamiento un poco menos constante a comparación del segundo modelo. En un caso hipotético, donde se desee aumentar el número de Epoch del cuarto modelo, posiblemente empiece a tener problemas de Overfitting por el comportamiento que toma el historial.

Finalmente, después de realizar en entrenamiento y la validación de los diferentes algoritmos realizados y propuestos, se da el cumplimiento de el ultimo objetivo específico, donde se evalúa la precisión del algoritmo con el 20% de los datos de validación. Esto se muestra en la parte del código donde se corren todas las funciones (Figura 40) en la línea de condigo *prepare_dataset.*

Capítulo 6. Conclusiones

- A partir de los diferentes modelos implementados, incrementar el número de filtros o Epoch no siempre es la mejor solución para aumentar la precisión (acuraccy primer modelo: 96.20%, accuracy segundo modelo: 95.63%), como se observa en el tercer modelo de clasificación binaria, debido a que muchas ocasiones lo que ocurre es que el modelo empieza a tener problemas de Overfitting (Figura 60), porque empieza a memorizar patrones de los datos de entrenamiento y eso genera que la precisión de los datos de entrenamiento sea alta y la precisión de los datos de test sea menor. Aplicar más filtros en la red, puede generar que el filtrado de la información de los datos de entrada discrimine valores importantes que generan que la red no tenga una buena precisión porque confunde mucho los datos al momento de hacer la predicción.
- Se pudo evidenciar en los modelos de clasificación multiclase existe una gran diferencia en el Accuracy de los modelos en un 18.38%, aunque se implementará el mismo modelo pero a diferente tipo de respuesta en tiempo de los sonidos (impulsivos y no impulsivos) la precisión para el modelo de los sonidos no impulsivos (Figura 65) es mucho mayor a la de sonidos impulsivos (Figura 63), esto se debe que a lo largo del tiempo, existen más valores en la amplitud de la señal a lo largo del tiempo, esto equivale a unos valores de MFCCS más a los que se pueden encontrar en un sonido impulsivo.
- En los algoritmos de clasificación binaria, modificar el número de filtros que tiene cada capa convolucional no afecta en gran medida la precisión del algoritmo ya que, al tener una base de datos tan amplia de las dos diferentes clases, tiene la capacidad de realizar comparaciones de diferentes sonidos puede no ser impulsivos. Como se

observó en uno de los historiales de la precisión binaria, existen puntos donde la precisión y la Función de pérdida tiene pequeños picos los cuales muestran que en ese punto el algoritmo trató de predecir una muestra la cual tiene características similares en tiempo y en frecuencia, como se observa en la figura 57.

- Los valores obtenidos en los modelos realizados tienen buenos valores, ya que los valores de Función de pérdida son pequeños y eso hace que la combinación de las funciones de perdida, los optimizadores, las funciones de activación, los Strides y los filtros funcionaron de manera correcta para el objetivo del algoritmo. Para los modelos vistos en el capítulo 5, para la clasificación binaria y multiclase la configuración de los primeros modelos (clasificación binaria y clasificación multiclase) fue la más optima y la que brindo mejores resultados. Para llegar a ese resultado, se tuvieron que realizar muchas pruebas y estar cambiando muchos de los parámetros de la red neuronal, ya que no existe una fórmula que permita crear una red de cualquier tipo y que de una precisión alta. Lo que, si se puede realizar, es crear capas las cuales permitan que el modelo no sufra de Overfitting, como puede ser el Dropout y el Batch Normalization.
- Para la clasificación binaria, usar una función de activación *Softmax* en la capa de salida, genera que el algoritmo nunca aumente la precisión y siempre se mantenga constante a lo largo de todo el entrenamiento, ya que la función *Softmax* esta optimizada y creada para la clasificación multiclase (ver Anexo 8.2)

Capítulo 7. Recomendaciones.

- Aparte de los modelos de Transfer Learning, se pueden entrenar modelos a partir del Fine Tuning y aprendizaje por refuerzo, los cuales son algoritmos ya creados para la clasificación y que están previamente entrenados con millones de datos.
- Al realizar este proyecto se evidenció la poca información que existe sobre los sistemas de detección de disparo en Colombia, puesto que la mayoría de las pruebas y resultados son realizados en otros países, principalmente en Chile. Tras el desarrollo del proyecto, se pudo evidenciar que este es un campo en desarrollo con un gran potencial de crecimiento, ya que el Machine Learning está en constante crecimiento.
- Se recomienda intentar ver cómo se comportaría el algoritmo con Transfer Learning, ya que la precisión (Accuracy) es uno de los pilares del proyecto y, tras el desarrollo de este, se podría lograr superar la precisión (Accuracy), ayudando a tener mejores resultados y redes más simplificadas, pero más potentes. Esto podría dar paso a la realización de un proyecto de realizar un montaje completo de reconocimiento de disparos de armas de fuego y localización del origen del disparo.

REFERENCIAS

Aguilar, J. R. (2015). Gunshot detection systems in civilian law enforcement. *AES: Journal of the Audio Engineering Society*, *63*(4), 280–291. https://doi.org/10.17743/jaes.2015.0020

Aprende Machine Learning. (2017). *Qué es overfitting y underfitting y cómo solucionarlo*. https://www.aprendemachinelearning.com/que-es-Overfitting-y-underfitting-y-como-solucionarlo/

Atria, I. (2019, October 22). *Qué son las redes neuronales y sus funciones*. https://www.atriainnovation.com/que-son-las-redes-neuronales-y-sus-funciones/

Beck, S. D. (2019). A short-time cross correlation with application to forensic gunshot analysis. *Proceedings of the AES International Conference*, 8.

Belkin, M., Ma, S., & Mandal, S. (2018). *To Understand Deep Learning We Need to Understand Kernel Learning*.

Busse, C., Krause, T., Ostermann, J., & Bitzer, J. (2019). Improved gunshot classification by using artificial data. *Proceedings of the AES International Conference*, 7.

Cabero, J. M. V. (2018). *Sistema de Reconocimiento de Comandos por Voz Basado en Redes de Neuronas LSTM*.

DeepAI. (n.d.). *Sigmoid Function Definition*. Retrieved November 18, 2021, from https://deepai.org/machine-learning-glossary-and-terms/sigmoid-function

Flórez López, R., & Fernández Fernández, J. M. (2008). *Las Redes Neuronales Artificiales*. Netbiblo, S.L. https://books.google.com.co/books?id=X0uLwi1Ap4QC&pg=PA82&dq=gradiente+

descendente+redes+neuronales&hl=es-419&sa=X&ved=2ahUKEwji44rx0JH0AhVHSjABHX9lBasQ6AF6BAgEEAI#v=onepage&q=gradiente descendente redes neuronales&f=false

Goodfellow, I., Bengio, Y., & Courville, A. (2016). *Deep Learning*. The MIT Press. https://books.google.com.co/books?hl=es&lr=&id=omivDQAAQBAJ&oi=fnd&pg=PR5&dq=Deep+Learning+(Adaptive+Computation+and+Machine+Learning+series)+pdf&ots=MNP5cqoGPS&sig=5FDvnhQLrff-ZhKUzKPCPwzDR4Y#v=onepage&q&f=false

Gun Law and Policy: Firearms and armed violence, country by country. (n.d.). Retrieved November 26, 2020, from https://www.gunpolicy.org/

Gunshot Audio Forensics Dataset - Gunshot Audio Analysis. (2017). http://cadreforensics.com/audio/

Hernández Sampiere, Roberto; Fernández Collado, Carlos; Baptista Lucio, M. del P. (2012). *Metodología de la investigación* (Vol. 66).

HRW. (2006). Genocide, War Crimes and Crimes Against Humanity: A Topical Digest of the Case Law of the International Criminal Tribunal for the Former Yugoslavia. In *Human Rights Watch*.

IArtificial.net. (2020). *Redes neuronales desde cero (I)*. https://www.iartificial.net/redes-neuronales-desde-cero-i-introduccion/

Lahr, J., & Fischer, F. (1993). *Location of acoustic sources using seismological techniques and software*. http://jclahr.com/science/psn/gunshots/of93221/

LANDR Blog. (2018). *¿Qué es el dither y cuándo se usa?* https://blog.landr.com/es/que-

es-el-dither-y-cuando-se-usa/

Llorente, C. R., Roberto, D., Chicote, B., Juan, D., & Montero Martínez, M. (2007). *Diseño, implementación y evaluación de técnicas de identificación de emociones a través de la voz*. 233.

López León, R. (2014). *Análisis comparativo de los efectos auditivos del Dithering en procesos de masterización digital estéreo.* [Univerisdad de San Buenaventura Seccional Medellín]. http://bibliotecadigital.usbcali.edu.co/bitstream/10819/3972/1/Analisis_Comparativo_Efectos_Lopez_2014.pdf

Machine Learning Crash Course. (n.d.). *Multi-Class Neural Networks: Softmax*. Retrieved November 18, 2021, from https://developers.google.com/machine-learning/crash-course/multi-class-neural-networks/softmax

Machine Learning Mastery. (2018a, February 14). *Una introducción suave a los tensores para el aprendizaje automático con NumPy*. https://machinelearningmastery.com/introduction-to-tensors-for-machine-learning/

Machine Learning Mastery. (2018b, July 20). *Difference Between a Batch and an Epoch in a Neural Network*. https://machinelearningmastery.com/difference-between-a-batch-and-an-epoch/

Maher, Robert C. & Routh, T. K. (2015). Advancing Forensic Analysis of Gunshot Recordings. *Audio Engineering Society Convention 131*, 1–8.

MathWorks. (n.d.). *Redes neuronales convolucionales*. Retrieved November 18, 2021, from https://la.mathworks.com/discovery/convolutional-neural-network-matlab.html

Millet, J., & Baligand, B. (2006). Latest achievements in gunfire detection systems. *Battlefield Acoustic Sensing for ISR Applications, RTO-MP-SET-107*, 1–14. http://oai.dtic.mil/oai/oai?verb=getRecord&metadataPrefix=html&identifier=ADA478974

Moreaux, M. (2018). *Environmental Sound Classification 50 | Kaggle*. https://www.kaggle.com/mmoreaux/environmental-sound-classification-50

Morillas, J. L. (2014). *SISTEMA PARA LA LOCALIZACIÓN DE LA DIRECCIÓN DE DISPAROS PROCEDENTES DE ARMAS DE FUEGO UTILIZANDO TÉCNICAS DE LOCALIZACIÓN DE FUENTES SONORAS*. 1–89.

Neira, Néstor Humberto Martínez, M. P. R. D. (2019). *Documentos de Política Pública y Armas y homicidios*. *1*(01), 23.

POCHO, C. (n.d.). *Redes neuronales convolucionales explicadas* . Retrieved November 18, 2021, from https://pochocosta.com/podcast/redes-neuronales-convolucionales-explicadas/

Sotaquirá Gutiérrez, M. Á. (2018). *Codificando Bits*. https://codificandobits.github.io/

Torres, J. (2020). *Python Deep Learning* (Primera ed).

Yosinski, J., Clune, J., Bengio, Y., & Lipson, H. (2014). *How transferable are features in deep neural networks?* 1–14.

Capítulo 8. ANEXOS

8.1. Algoritmo de pre-procesamiento archivos de audio

Pre-procesamiento de los datos

En este cuaderno se pretende realizar todo el pre-procesamiento de los datos, lo que consiste en igualar el tiempo de los audios y frecuencia de muestreo. Para esto, se usará la librería librosa y numpy para el manejo de arreglos.

```
import librosa
import numpy as np
from google.colab import drive
drive.mount("/content/gdrive") #Se monta el drive para trabajar directamente desde la nube
ruta_audios = 'gdrive/My Drive/Colab Notebooks/Tesis/Base de datos (Binario)/Disparos sin e
ditar/' #Se carga la ruta donde estan los audios
```

Ahora, se carga el nombre de los audios mediante la librería **os**

```
import os
nombre_audios = os.listdir(ruta_audios)
```

Conversión de los datos

Se convierten los audios a 44100 Hz, 16 bits y una longitud de 5 segundos, agregando silencio a los datos que tengan menor duración, con el objetivo que todos los datos tengan la misma longitud de datos. Para esto, se usará la librería **soundfile**

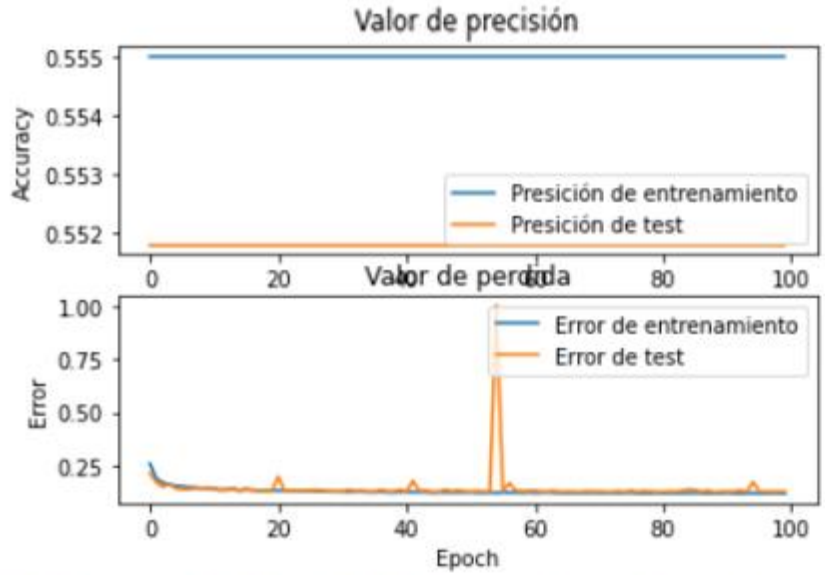
```
import soundfile as sf

for i in nombre_audios:
  audio_cargado = print('El audio cargado es: ', ruta_audios+i)
  cargar_audio = librosa.load((ruta_audios+i),
        sr = 44100, mono = True, dtype = 16)
#Se carga el audio en formato mono, frecuencia de muestreo 44100 Hz
y una profundidad de bits de 16
  audio_procesado = cargar_audio[0] #Se extrae unicamente el arreglo del audio
```

```
 longitud_audio = len(audio_procesado) #Se guarda una variable con la longitud del audio
 longitud_datos = 5*44100 #Se calcula la longitud de datos para 5  segundos de audio
 arreglo_nuevo = np.zeros(longitud_datos-
longitud_audio) #Se crea un arreglo de zeros con la longitud de 5 segundos
 compilado = np.append(audio_procesado,arreglo_nuevo) #Se llena el arreglo del audio con los
zeros para completar 5 segundos
 nombre = audios_prueba[h] #Se extrae el nombre del audio
 print(audios_prueba[h])
 h = h+1
 audio_final = sf.write('gdrive/My Drive/Colab Notebooks/Tesis/Base de datos/Base de datos (
Binario)/Disparos'+
      nombre, compilado, 44100) #Se guarda el  nuevo audio de 5 segundos, 44100 Hz y 16 bits
```

8.2. Clasificación binaria con función Softmax

```
from sklearn.model_selection import train_test_split
from tensorflow.keras.layers import Dropout, BatchNormalization

def build_model(input_shape):
    model = keras.Sequential()

    # Primera capa convolucinal.
    model.add(keras.layers.Conv2D(32, (5, 5), activation='relu', input_shape=input_shape))
    model.add(keras.layers.MaxPooling2D((5, 5), strides=(2, 2), padding='same'))
    model.add(keras.layers.BatchNormalization())

    # Segunda capa convolucional.
    model.add(keras.layers.Conv2D(64, (3, 3), activation='relu', input_shape=input_shape))
    model.add(keras.layers.MaxPooling2D((3, 3), strides=(2, 2), padding='same'))
    model.add(keras.layers.BatchNormalization())

    # Tercera capa convolucional
    model.add(keras.layers.Conv2D(32, (2, 2), activation='relu', input_shape=input_shape))
    model.add(keras.layers.MaxPooling2D((2, 2), strides=(1, 1), padding='same'))
    model.add(keras.layers.BatchNormalization())

    # Capa Flatten y capa densamente conectada.
    model.add(keras.layers.Flatten())
    model.add(keras.layers.Dense(64, activation='relu'))
    model.add(keras.layers.Dropout(0.25))

    # Capa de salida binaria
    model.add(keras.layers.Dense(1, activation='softmax'))

    return model
```

Figura 72. Creación del modelo con función de salida Softmax




```
386/386 - 1s - loss: 0.1355 - accuracy: 0.5537 - 786ms/epoch - 2ms/step

Accuracy de datos de test: 0.5537364482879639
```

Figura 73. Accuracy y Función de pérdida

8.3. Enlaces

Drive con archivos y código.

https://drive.google.com/drive/folders/1hKyBXzT_1ZkOwWzocZeoGi7XVcG8aaf2?usp=sharing