Benemérita Universidad Autónoma de Puebla

Facultad de Ciencias de la computación

“Interfaz de control Cerebro-Computadora de un Dron mediante redes convolucionales”

Tesis presentada para obtener el título de Maestría en Ciencias de la Computación

Presenta:

Yufni Abraham Castro Balbuena

Directores de Tesis:

Dr. Mario Anzures García

Dr. José Alejandro Rangel Huerta

Puebla, Puebla, México, Noviembre 2022

Índice

Lista de Figuras

Lista de Tablas

1 Introducción

Una interfaz es la conexión funcional entre dos sistemas que permite el intercambio de información entre ellos. Una interfaz cerebro-computadora es una conexión unidireccional que interpreta patrones en las ondas electromagnéticas producidas por la sinapsis de las neuronas para activar un comando en el ordenador.

Las interfaces cerebro-computadora se han utilizado para controlar robots simples como mesas con ruedas, brazos mecánicos, minijuegos de computadora, drones, entre otros.

El control de un dron se basa en dos joysticks para el movimiento en un ambiente tridimensional. Cada uno de ellos se encarga del movimiento en dos dimensiones, el primero para controlar los movimientos arriba, abajo, izquierda y derecha. El segundo adelante, atrás, giro izquierda y giro derecha.

Las redes convolucionales son un tipo de red neuronal especializada en encontrar patrones en imágenes. En los últimos años han tenido resultados sorprendentes en distintas áreas al identificar patrones complejos como clasificación de objetos, creación de rostros o plegamiento de proteínas.

Por tanto, en este trabajo se busca hacer uso de redes convolucionales para identificar patrones en las imágenes creadas a partir de mediciones de ondas cerebrales, para crear una interfaz cerebro-ordenador que sirva como control para los movimientos de un dron en un entorno bidimensional reemplazando las funciones del segundo joystick.

1.1 Interfaz Cerebro Computadora

El término “Brain-Computer Interface” fue acuñado por Jacques Vidal [1] en los años 70’s mientras trabajaba para la Universidad de California. Desde ahí han existido múltiples intentos de “controlar cosas con la mente”. Sin embargo, y a pesar de que comprendemos cómo funciona el cerebro, al menos desde el punto de vista químico, físico y anatómico. Fenómenos como el sueño y las memorias siguen siendo un misterio [2]. Comenzando con la primera pregunta clave.

1.1.1 ¿Cómo funciona el cerebro?

El cerebro humano es la estructura más compleja que conocemos y se ha tratado de explicar su funcionamiento haciendo alusión a otros mecanismos. Cuando en 1776 Jamer Watt creó la máquina de vapor (Y siendo la máquina más compleja y avanzada de la época) se hizo alusión al corazón y al cerebro. Varios conceptos de la psicología, ciencia que comenzó durante la revolución industrial. Tomaron conceptos de la máquina de vapor para explicar el comportamiento humano. Hasta el día de hoy estamos familiarizados con la frase de “estar bajo presión” que viene del funcionamiento de las calderas de vapor.

Con la llegada de la electricidad y los sistemas de distribución eléctrica, los médicos encontraron una similitud entre las complejas redes neuronales y los cableados que comenzaban a cubrir las grandes ciudades. Hoy en día es común escuchar que el procesador es el cerebro de una computadora y la realidad es que al menos al día de hoy, es el sistema más parecido que hemos creado al cerebro humano.

La célula principal del cerebro son las neuronas (véase la Figura 1). Las neuronas son las células que conforman el sistema nervioso. Sistema que controla las funciones conscientes e inconscientes de nuestro organismo, desde liberar insulina hasta mover nuestros dedos para escribir en un teclado. Su longitud va desde menos de un milímetro a varios metros. Entre sus muchos componentes los dos más relevantes para este trabajo son el axón y la dendrita.

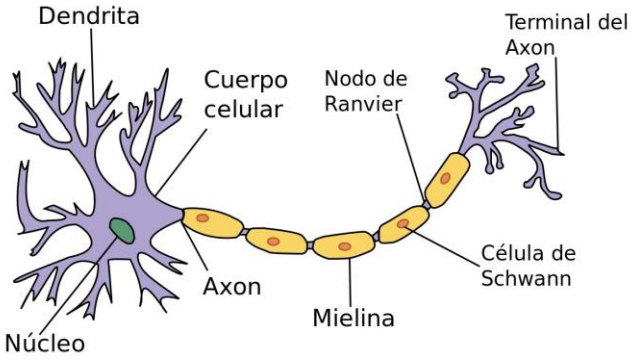

Figura 1. Estructura de una neurona.

El axón es la terminación de la neurona, por medio de ella se conecta con otras neuronas. Las dendritas son los receptores de la neurona y por medio de ellas puede recibir la información de otras neuronas. ¿Qué clase de información? Química o eléctrica.

Cuando dos neuronas se comunican entre sí el axón de la neurona libera neurotransmisores, estos son recibidos por las dendritas de otras neuronas produciendo un impulso eléctrico que ronda los 70 mv. Esto activa la neurona receptora que a su vez repite el proceso con las neuronas que están conectadas a esta. Este proceso es conocido como sinapsis.

Cuando una serie de neuronas son estimuladas de la misma manera repetidas veces, la conexión entre ellas se vuelve más rápida [3]. El axón y la dendrita cada vez se colocan más cerca y el proceso se vuelve un intercambio eléctrico. Es por ello que los deportistas tienen tiempos de reacción casi inmediatos después de años de práctica. las neuronas involucradas en esos movimientos se encuentran unidas. En cambio, al aprender una nueva habilidad el proceso puede ser lento, ya que el cerebro necesita hacer nuevas conexiones para lograr que el cuerpo ejecute la tarea.

1.1.2 Electroencefalograma

Uno de los métodos para medir la actividad en el cerebro es el electroencefalograma. Inventado en 1924 por Hans Berger [4]. El método consiste en colocar un material conductor de electricidad sobre la cabeza de una persona. De acuerdo con el principio de inducción electromagnética, al circular una corriente eléctrica por un material conductor, alrededor de este se genera un campo magnético. Un material conductor puede producir una corriente eléctrica al estar cerca de un campo magnético. Bajo este principio funcionan transformadores, motores y bobinas (véase la Figura 2).

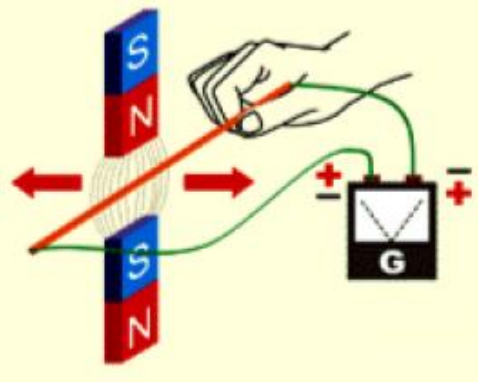

Figura 2. Inducción electromagnética.

El material conductor, para este caso las neuronas producen un campo magnético que puede producir una corriente eléctrica en otro material conductor, llamado electrodos. De acuerdo a la activación de las neuronas se producirá corriente eléctrica en los electrodos que puede ser medida y graficada. A mucha actividad neuronal habrá grandes variaciones de corriente en los electrodos y a poca actividad se podrá observar una señal menor frecuencia tal y como se distingue en la figura 3 [15].

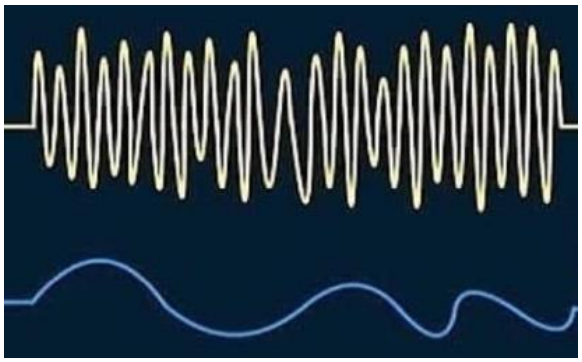

Figura 3. Variación de frecuencia en ondas cerebrales.

1.1.3 Los tipos de ondas

Como mencionamos anteriormente el problema con los métodos de los que disponemos para medir la actividad cerebral es que no nos dicen que neuronas se están activando o comunicando entre ellas. Solo sabemos que grupos se están activando y en base a eso los hemos clasificado por tareas, sabemos que parte del cerebro se activa cuando vemos un paisaje y cual cuando escuchamos una canción.

Si observamos durante 24 horas las señales producidas por un equipo de electroencefalografía encontraremos una que van variando en frecuencia y amplitud. De acuerdo a estas variaciones han sido clasificadas en cinco tipos (véase la Figura 4).

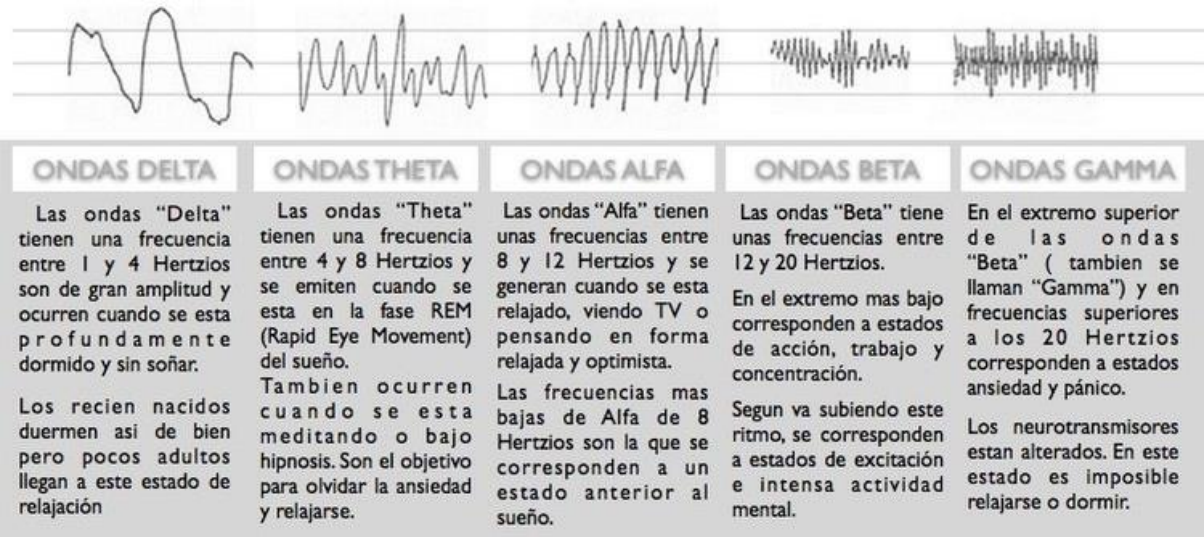

Figura 4. Clasificación de ondas cerebrales.

Las ondas Delta corresponden a etapas de sueño nREM (*Non Rapid Eye Movement*) periodo de relajación profunda donde se ralentizan o desactivan las funciones del cuerpo. Tienen una gran amplitud y frecuencias muy lentas, aun no se entiende porqué se producen, pero la teoría es que en esta etapa las neuronas fortalecen las conexiones neuronales importantes que se crearon durante el día y desechan las que no lo son. Según esta teoría estarían relacionadas con el aprendizaje [2].

Las ondas Theta junto con las delta son de los misterios que esconde el cerebro [2]. Corresponden a periodos de sueño donde se desactivan varias funciones que se activan al estar conscientes pero se mantiene una ”consciencia” que permite soñar. Están relacionadas con el periodo de sueño REM (*Rapid Eye mov*ement) donde se

dan los sueños nítidos. Son mayores en frecuencia y menores en amplitud que las Delta.

Las ondas Alfa mayores en frecuencia y mayores en amplitud que las Beta corresponden a una persona despierta sin actividad. Es de esperar mayor frecuencia debido a todas las funciones que entran en juego para mantener a una persona consciente.

Las ondas Beta con frecuencias rápidas y poca amplitud corresponden a estados de alerta y concentración. Estados donde hay alta actividad cerebral, una bailarina consciente e inconscientemente tiene que controlar los movimientos de brazos y piernas, equilibrio, músculos abdominales, gestos, posición espacial, respiración, visión, sumado a todos los demás procesos que necesitamos simplemente para estar vivos. Todos esos cálculos complejos dan lugar a que las neuronas se activen y desactiven en distintas maneras y tiempos y nos regalen las ondas Beta.

Las ondas Gamma corresponden a ataques de ansiedad y pánico, literalmente es cuando todo está en llamas dentro del cerebro. Se forman ideas una tras otra, el cuerpo se prepara para pelear, se contraen los músculos, se libera bilis, las venas y arterias se contraen para evitar morir desangrado, el cerebro está en alerta constante esperando un ataque. Pueden tener frecuencias más rápidas que las de la red eléctrica y sus amplitudes son bajas.

La relación frecuencia amplitud que se da con las neuronas es sencilla de explicar, si las neuronas trabajan en conjunto se da una gran amplitud ya que muchas de ellas se activan al mismo tiempo por periodos largos generando frecuencias bajas. Cuando el cerebro busca realizar muchas actividades a la vez. Las neuronas se activan en ritmos y períodos diferentes generando frecuencias rápidas por las activaciones en distintos tiempos, pero generando amplitudes mínimas ya que una neurona no genera el mismo potencial que 1000 activándose al mismo tiempo.

Para este trabajo las ondas relevantes son aquellas que se dan al estar despierto, ondas Alfa y Beta. y son también las que presentan el problema de contener la mayor cantidad de ruido por todas las demás funciones involucradas con el funcionamiento del cuerpo humano.

1.1.4 ¿Cómo leer las ondas?

Más allá de los detalles técnicos es relativamente fácil “escuchar” el cerebro. Pero una vez que podemos visualizar las ondas cerebrales ¿Como las leemos? He aquí el dilema. Observando las figuras cinco y seis ¿Qué nos dicen?

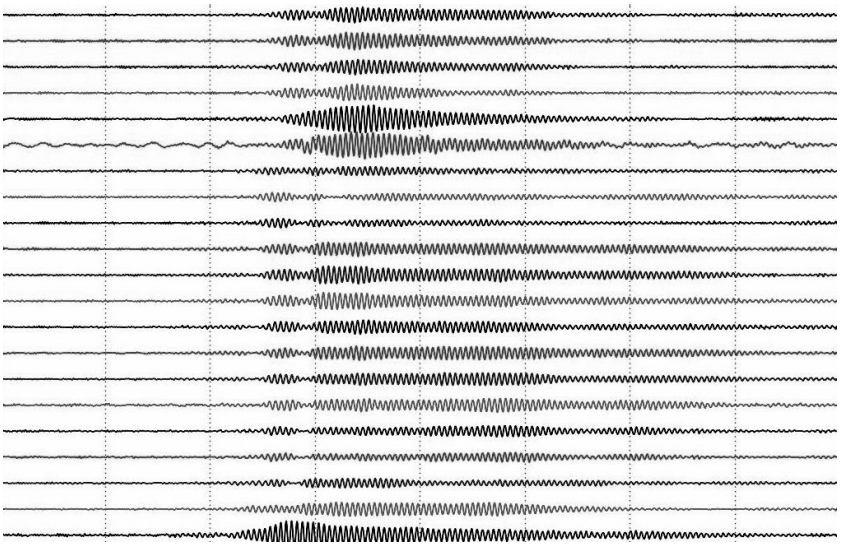

Figura 5. ¿Ondas Cerebrales?

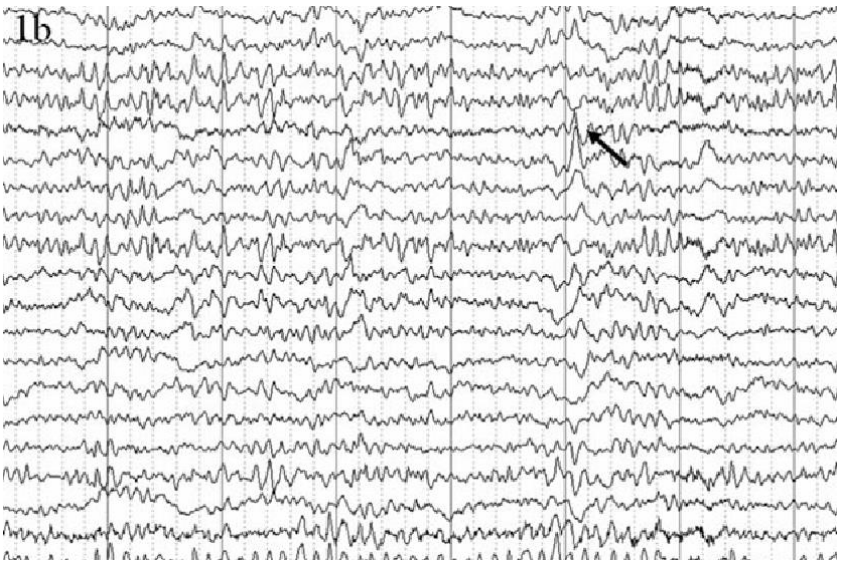

Figura 6. Ondas Cerebrales.

Bueno, para empezar una de ellas proviene de un sismograma (Figura 5) y no de un cerebro. La verdadera complicación al tratar de entender lo que pasa dentro del cerebro es que hay tantas neuronas activándose a la vez que resulta confuso interpretar lo que pasa. Además, no hay una técnica que nos permita escuchar a una

única neurona, menos a un grupo de manera individual. Sin embargo, previamente dijimos que el lóbulo parietal controla el movimiento. ¿Cómo sabemos esto?

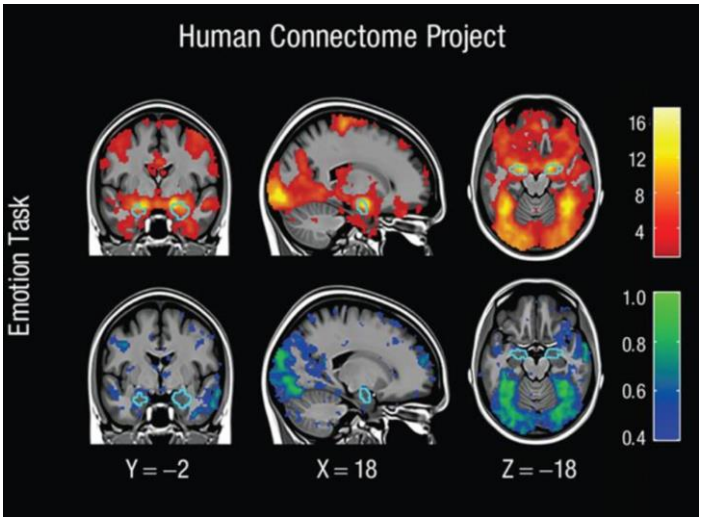

Figura 7. Resonancia electromagnética.

Una técnica más avanzada, la resonancia electromagnética (véase la Figura 7), nos ayuda a comprender mejor. Si bien, no sabemos qué pasa con cada neurona podemos observar cómo distintas partes del cerebro se activan al realizar tareas como hablar, escuchar, moverse, etc. De esta manera sabemos qué áreas intervienen en las diversas funciones de nuestro cuerpo. Podemos ver esta actividad en el cerebro con la electroencefalografía colocando electrodos en toda la superficie craneal y viendo cuales electrodos reaccionan al realizar una tarea, pero sin lugar a dudas la resonancia nos ofrece una imagen más legible de lo que pasa dentro.

1.1.5 Interfaz Cerebro Ordenador

La interfaz propuesta por Jacques Vidal para conectar el cerebro humano con una aplicación de computadora se describe en la figura 8.

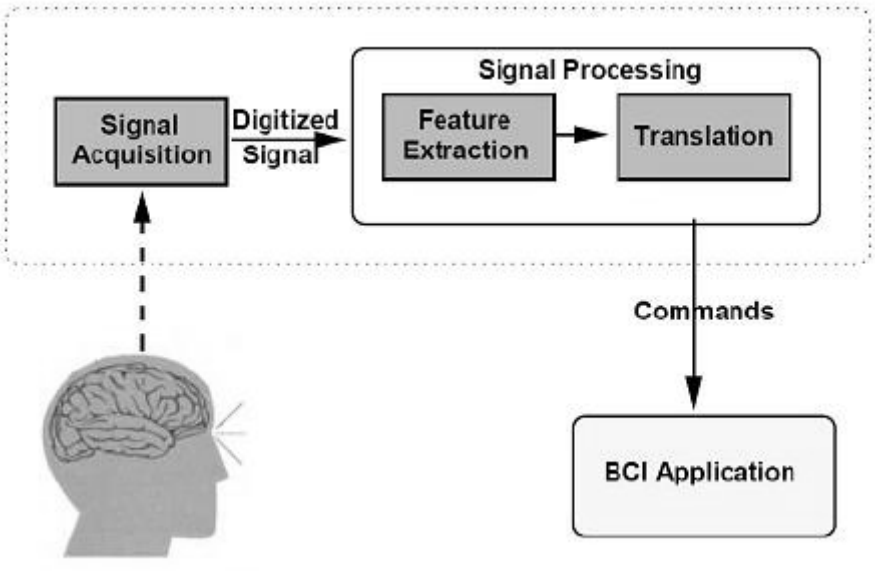

Figura 8. Interfaz Cerebro Ordenador propuesta por Jaques Vidal.

La idea de Jacques Vidal consistía en medir la actividad cerebral por medio de electrodos y digitalizar las señales. Extraer características de las señales para “traducirlas” y ejecutar comandos con ellas. En teoría resulta bastante lógico. Pero hay un problema. No sabemos cómo funciona el cerebro.

1.1.6 El problema

A pesar de entender cómo funciona la sinapsis y de que hemos observado y comprobado que hay ciertas áreas del cerebro que tienen funciones específicas. No sabemos aún cómo se pasa de una señal eléctrica a recuerdos, imaginación o consciencia. Pero se puede usar las áreas que ya sabemos cómo funcionan.

Si bien esto puede ayudar a que médicamente se puedan identificar patologías, aún no podemos obtener datos específicos de esas áreas. Podemos ver actividad en el lóbulo parietal mientras nos movemos, pero no podemos describir con precisión, a partir de la actividad neuronal si se moverá una pierna o un dedo del pie, en qué dirección o con cuánta fuerza será el movimiento.

El colocar un electrodo y comenzar a tomar mediciones es el equivalente a colocar un micrófono en un estadio de fútbol lleno de personas y escuchar las voces de todos los aficionados a la vez [2]. Sabemos que hay actividad en determinada sección del cerebro, pero no podemos saber qué neuronas están siendo activadas para el movimiento de esa pierna y aún peor. Al mismo tiempo que la pierna se mueve nuestros pulmones, estómago, tímpanos y otros muchos órganos están comunicándose con el cerebro.

Si a esto le sumamos las texturas percibidas por la piel, luz y otros estímulos externos, las emociones, pensamientos, estados de ánimo y los nutrientes que están o no actuando en nosotros, se tiene un sistema complejo con infinidad de variables involucradas. En la actualidad se están realizando esfuerzos para poder medir la actividad de las neuronas de manera individual [5]. Pero mientras se esperan los resultados de esos estudios se han hecho intentos por extraer información relevante entre la tormenta de sinapsis que ofrece el electroencefalograma.

1.1.7 El objetivo

Este proyecto busca clasificar señales de ondas cerebrales según la intención de movimiento. En 2013, el Dr. Yukiyasu Kamitani del Advanced Telecommunications Research Institute International en Kyoto [2]. Se dio a la tarea de crear un diccionario para interpretar sueños. A pesar de basarse en solo tres individuos el estudio fue significativo. Aunque no lograron interpretar si la persona estaba soñando con un automóvil rojo o verde, una honda o un hyundai. O si estaban soñando con una persona rubia o morena. Lograron categorizar las mediciones y clasificarlas, con esto lograron saber si estos individuos habían soñado con un hombre o una mujer, con autos, libros, muebles, computadoras o comida. Si bien el método está muy lejos de ser perfecto, resulta asombroso lo que se logró interpretar.

Se han intentado realizar una amplia variedad de clasificaciones con diversos objetivos como detectar enfermedades o “leer la mente”, entre ellas el clasificar los movimientos imaginados un objetivo similar al de este proyecto [7]. En este estudio hecho por el Laboratory of Brain-Computer Interfaces de la Graz University of Technology se buscó clasificar la intención de movimiento usando 22 electrodos de los cuales 6 son mencionados como los principales, esto servirá de base al proyecto.

Se han intentado clasificar y entender las ondas cerebrales por diversas maneras, modelos de machine learning, clasificación estadística, extracción de características e iterando sobre diversos algoritmos. Varios de ellos han logrado clasificar el comportamiento del cerebro en determinados perfiles de acuerdo al objetivo planteado aún carecemos de un intérprete cerebral que decodifique cada activación neuronal. Un método para interpretar los sueños no nos dirá que parte del cuerpo se está moviendo, y un método para identificar movimiento muscular no servirá para entender que se está escuchando.

1.1.7.1 Los comandos

Por lo tanto, en este momento, entre más específica sea la tarea que deseemos clasificar y más general la respuesta que deseemos recibir, más sencillo será alcanzar el objetivo. Es decir, en este momento podemos descubrir fácilmente si el cerebro está escuchando música, pero será más complicado tratar de descubrir el compás de la melodía. Teniendo esto en mente el objetivo del proyecto se limitará a clasificar los movimientos con cuatro intenciones: derecha, izquierda, adelante y sin intención de movimiento.

Figura 9. Control de un dron.

Buscando reemplazar casi por completo uno de los joysticks (véase la Figura 9) que se utilizan para controlar drones y obteniendo el control total en un entorno bidimensional.

1.1.7.2 El origen: “El lóbulo parietal”

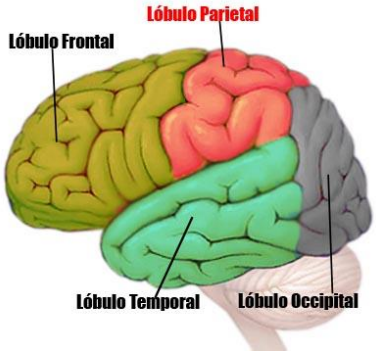

Figura 10. Lóbulos del cerebro.

El lóbulo parietal (véase la Figura 10) es el sector del cerebro donde se genera la intención del movimiento consciente [6]. La intención de movimiento es como un correo electrónico que aún no ha sido leído por el sistema nervioso para ser ejecutado por los nervios. El movimiento del cuerpo comienza en el cerebro, se calcula en una manera que aún no tenemos clara con qué intensidad y duración se debe de emitir una señal eléctrica y después se envía esa señal al nervio deseado para contraerlo [14].

A pesar de sonar sencillo a todo ser humano le toma años de práctica lograr un control sobre su cuerpo para conseguir caminar. El cerebro aprende por medio de repetición con qué intensidad mandar la señal a los nervios para lograr la presión, movimiento y velocidad deseados. Y aún más años de práctica para tareas complejas como escribir. Sin embargo, se trate de levantar un vaso o de realizar una cirugía a corazón abierto todo movimiento surge como una idea en el cerebro y se ejecuta como una señal eléctrica que contrae los nervios y acciona las fibras musculares.

1.2 Planteamiento del problema

No hay aún, un método 100% fiable para clasificar la información contenida en las mediciones de ondas cerebrales. Además de que el costo del hardware para obtener estas mediciones resulta prohibitivo.

Este proyecto busca experimentar el uso de redes convolucionales para clasificar las mediciones de ondas cerebrales y clasificarlas según 4 movimientos. Controlando un dron en un plano bidimensional a partir de esos 4 comandos. Todo esto a partir de un hardware accesible. No se busca clasificar de manera perfecta las señales, el objetivo es probar la efectividad de las redes convolucionales para este problema de clasificación.

1.3 Hipótesis del trabajo

Las redes convolucionales han demostrado en los últimos 5 años ser excelentes para problemas de clasificación donde abunda el “ruido”. Poder identificar un gato ya sea que este se encuentre en una ciudad, campo o este en distintas posiciones a partir de identificar patrones básicos que definen al gato, como orejas puntiagudas y bigotes.

La hipótesis a probar es, que al entrenar la red con señales de ondas cerebrales esta pueda aprender a identificar esos patrones básicos que representan movimiento y lograr “filtrarlos” (identificarlos a pesar de otras señales que introduzcan ruido a la medición).

1.4 Objetivos

1.4.1 Objetivo General

Implementar una interfaz cerebro-ordenador usando redes convolucionales, para ejecutar cuatro comandos a partir de mediciones de ondas cerebrales.

1.4.2 Objetivos Específicos

Codificar las señales de ondas cerebrales obtenidas en formato de imagen (matriz).

Implementar y entrenar una red convolucional que active 4 salidas posibles a partir de las imágenes obtenidas.

Vincular las salidas de la red neuronal con el control del dron en el simulador.

Calcular el porcentaje de movimientos correctos al controlar el movimiento bidimensional de un dron por medio de la interfaz.

2 Preparando la señal

Antes de hablar del método elegido para clasificar la señal debemos obtener la señal. Uno de los problemas al intentar tomar señales del cerebro es el equipo para hacerlo. El acceso a un equipo de electroencefalografía es costoso y limitado, pocos equipos por ciudad. Por lo que hacer uso de uno de estos de manera continua para obtener muestras y probar un modelo era inviable.

2.1 El dispositivo

Existe una variedad de marcas que se dedican a crear diademas con electrodos para proyectos de investigación y desarrollo. Una de las más utilizadas es la marca Emotiv cuyo modelo utilizado (y económico) es el insight. El problema con esta diadema es la rigidez de los electrodos. Estos no pueden cambiar de posición y solo tiene uno en cada parte del cerebro. Por lo que intentar tomar varias mediciones en el lóbulo parietal es imposible. Sumado a esto, Emotiv provee de su propio software, esto añade un costo de suscripción y una fuerte limitante en cuanto al manejo de las mediciones. A pesar de existir modelos con más electrodos este segundo punto valió para descartarlo como opción.

Otra opción planteada fue usar la diadema de nextmind. Sin embargo, esta presentaba aún más limitantes en el manejo de la señal, ya que está pensada 100% para el desarrollo de aplicaciones, haciendo uso de su propio modelo neuronal de clasificación.

Por lo que se optó por crear el sistema desde cero y tener el control total. Desde la posición de los electrodos hasta el manejo de la señal. A pesar del aumento de la complejidad esta opción también era significativamente más económica. El receptor se compone de hardware y software los cuales podemos clasificar en cuatro etapas: **Sensado, Amplificación, Filtrado y Visualización.**

2.2 Sensado

La parte de sensado consta de dos partes principales. El electrodo y el circuito de protección.

2.2.1 Electrodo

Existen diferentes tipos de electrodos clasificados en dos categorías principales. Electrodos secos y electrodos húmedos.

El nombre de electrodo húmedo viene de la necesidad de utilizar un gel conductivo entre el electrodo y el cuero cabelludo. Este tipo de electrodos está hecho de oro, plata, acero inoxidable. Por excelencia el oro es el material con la mejor conductividad eléctrica, pero también el más costoso. Este tipo de electrodos también presenta dificultades para su uso, ya que tiene que colocarse el gel mencionado antes de cada uso. Y si bien la precisión en las mediciones es la mejor, su poca practicidad los ha delegado a pruebas de laboratorio en ambientes controlados.

Figura 11. Electrodo seco.

El electrodo seco (véase la Figura 11) pese a tener una conductividad peor, usualmente hechos de acero inoxidable presenta la ventaja de ser económico y de fácil uso. Basta con colocarlo y tener un buen contacto con la superficie del cuero cabelludo, no es necesario aplicar gel ni rasurar el área de contacto. Este tipo de electrodos fueron los utilizados para este proyecto.

De acuerdo al estándar 10/20 que se utiliza para mediciones con EEG. Las posiciones utilizadas para los tres electrodos fueron Cz C3 y C4 (véase la Figura 12).

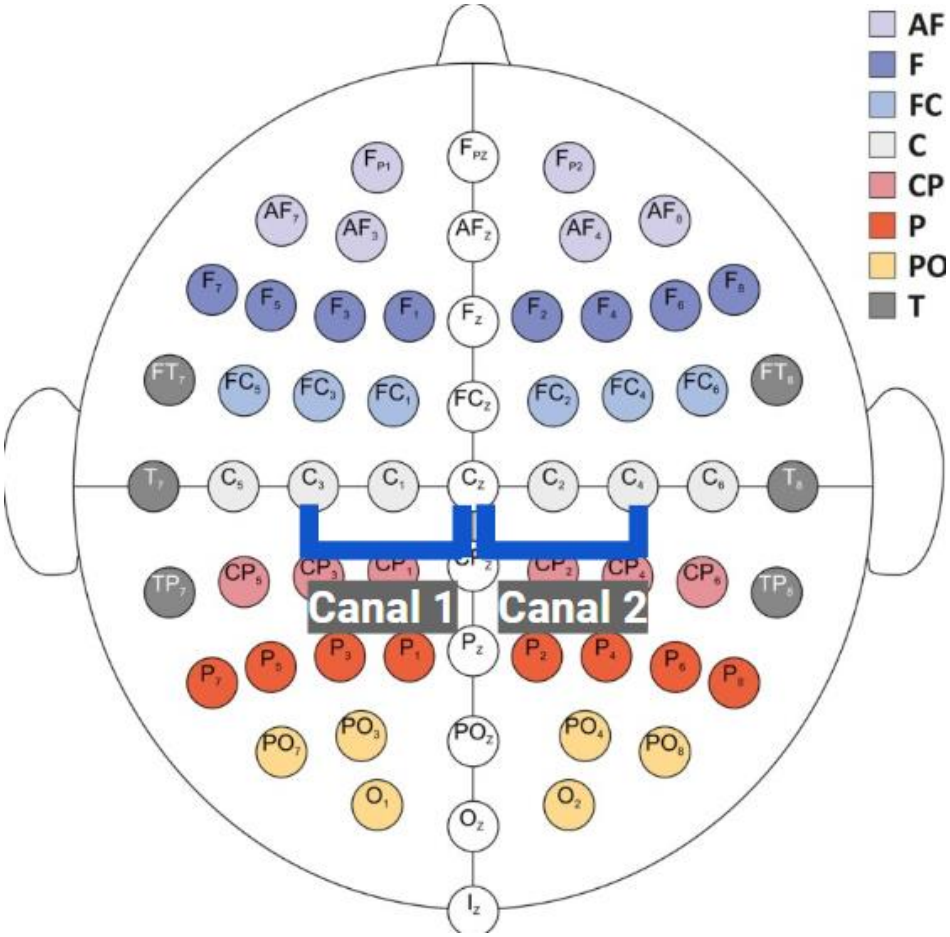

Figura 12. Posición de los electrodos según el estándar 10/20.

Se optó por utilizar dos canales formados por los tres electrodos en modo bipolar. La ventaja de este modo es que los electrodos provocan un cortocircuito en las señales que son iguales en ambos electrodos cancelándolas. Esto actúa también como un primer filtro para eliminar señales que se encuentran en el cuerpo o en ambos hemisferios del cerebro y dejar solo la actividad exclusiva del hemisferio correspondiente.

2.2.2 Protección

Una vez que se toma la señal por medio de los electrodos es necesario crear un circuito de protección que impida que las corrientes eléctricas de los circuitos siguientes puedan circular por el cuerpo del individuo a quien se realiza la medición. Para ello se utiliza el siguiente circuito formado por cuatro transistores, dos PNP y dos NPN. El circuito está diseñado para permitir el libre paso de corriente hacia la etapa de amplificación abriendo el paso por medio de los transistores e impidiendo con los mismos que el voltaje circule en sentido contrario (véase la Figura 13).

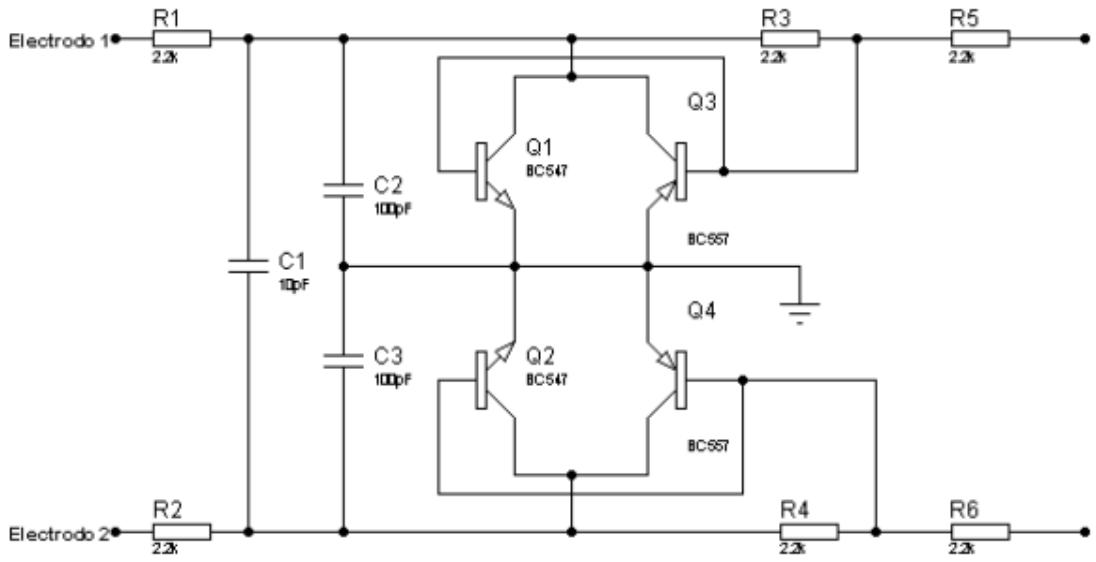

Figura 13. Circuito de protección.

2.2.3 Amplificación

Las bioseñales tienen amplitudes que rondan los 5mV. Trabajar con señales eléctricas tan pequeñas es complicado y casi siempre implica interferencia por ruido electromagnético. Para poder obtener una señal lo suficientemente limpia hay que amplificar y filtrar en varias etapas. La primera de ellas consiste en utilizar un amplificador de instrumentación el cual está formado internamente por tres amplificadores operacionales y puede trabajar con señales del orden de mV debido a su bajo voltaje de polarización.

El amplificador seleccionado para este proyecto fue el **AD620** debido a su bajo costo, disponibilidad y cumple con la característica de un bajo voltaje de polarización lo que lo vuelve capaz de trabajar con la amplitud de las bioseñales.

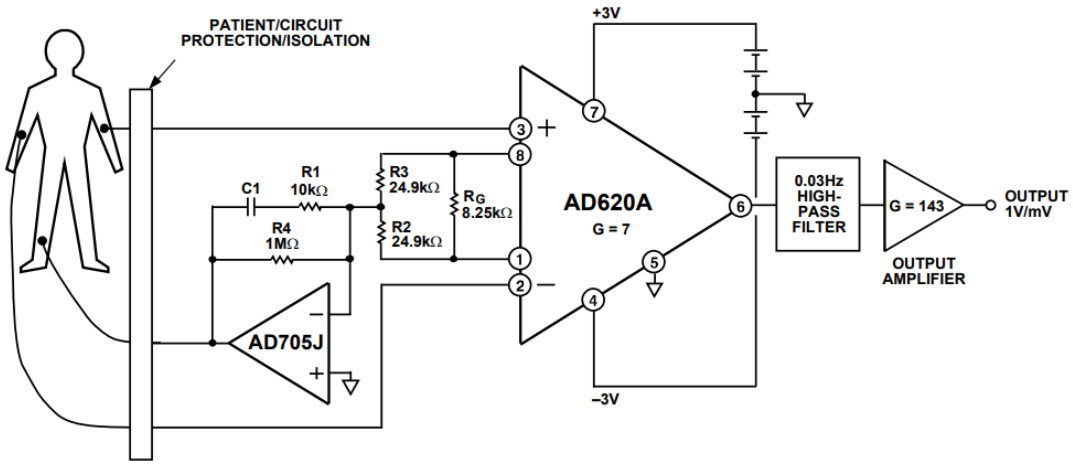

Figure 39. A Medical ECG Monitor Circuit

Figura 14. Circuito amplificador.

El circuito utilizado es el sugerido en la misma hoja de datos del amplificador (véase la Figura 14) con la modificación de sustituir la resistencia de 8.2Kohms por un potenciómetro de 10Kohms para poder controlar la ganancia a la salida del amplificador. El amplificador AD705J produce utiliza parte del voltaje que circula en el amplificador de instrumentación para mandar un voltaje diminuto al cuerpo de la persona el cual ayuda a ser tomado como referencia y eliminar ruidos producidos por las señales electromagnéticas que impactan en el cuerpo de la persona. Esto es necesario, ya que el cuerpo es un conductor de la electricidad, por tanto, las señales electromagnéticas inducen un potencial eléctrico en nuestro cuerpo y afectan las mediciones.

2.3 Filtrado

La etapa de filtrado consta de tres filtros y un desplazamiento que se aplican en distintas etapas.

2.3.1 Filtro pasa altos

Se adiciona un filtro pasa altos inmediatamente después del amplificador formado por una resistencia y un capacitor el cual discrimina frecuencias menores a 0.016Hz, como se mencionó anteriormente las señales buscadas oscilan entre 0.5 Hz y 60 Hz, por lo que señales mayores o menores sólo interferirá con las lecturas (véase la Figura 15).

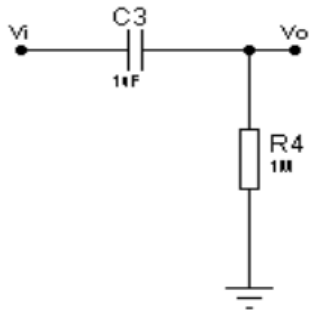

Figura 15. Filtro pasa altos.

2.3.2 Filtro Notch

Gran parte del ruido electromagnético que puede ensuciar la señal proviene de equipos eléctricos, como electrodomésticos, focos, etc. Estos equipos utilizan la corriente eléctrica suministrada por la compañía eléctrica la cual es transportada a una frecuencia de 60 Hz. Por lo que debemos eliminar las señales medidas que tengan esta frecuencia.

Para ello se implementa un filtro notch que actúa como elimina banda, evitando el paso de cualquier frecuencia de 60Hz (véase la Figura 16).

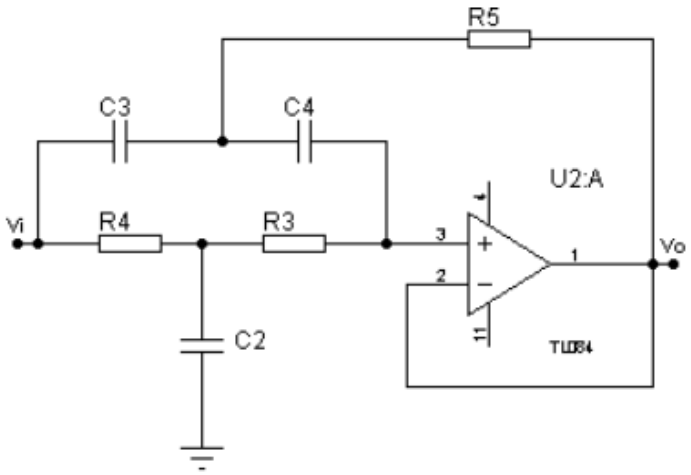

Figura 16. Filtro Notch.

2.3.3 Filtro pasa bajos

Ahora se busca eliminar las frecuencias mayores a lo buscado mediante un filtro pasa bajas. Este también amplifica la señal con una ganancia de 10. El filtro evita el paso de frecuencias mayores a 45 Hz en teoría, en la práctica el filtro actúa efectivamente alrededor de los 80 Hz y evitando el paso total de frecuencias mayores a 110 Hz (véase la Figura 17).

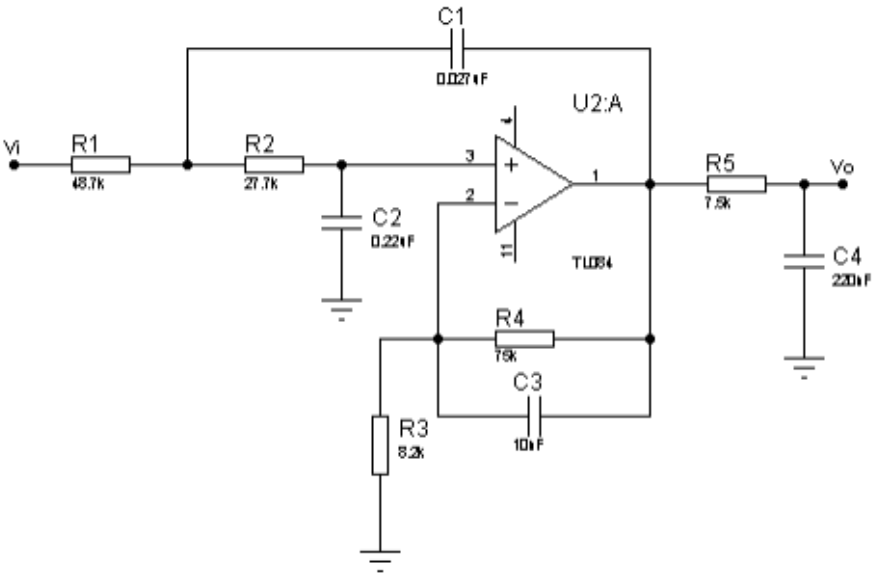

Figura 17. Filtro pasa bajos.

2.4 Desplazamiento

Las señales medidas oscilan entre voltajes positivos y negativos pero el convertidor de lecturas analógicas a digitales trabaja únicamente con voltajes positivos por lo que se realiza un desplazamiento de la señal o offset con ayuda de un amplificador operacional que además amplifica nuevamente la señal. El desplazamiento se logra por medio de las resistencias las cuales modifican el offset del amplificador y logran que las señales obtenidas se midan únicamente con voltajes positivos (véase la Figura 18).

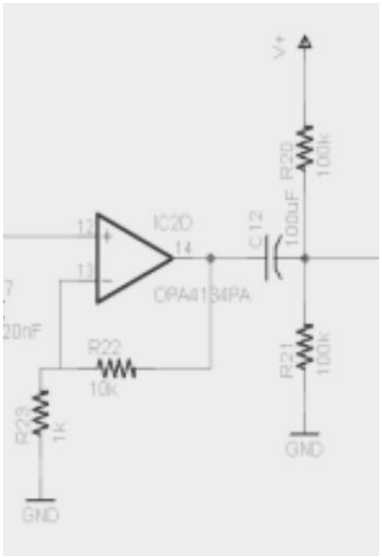

Figura 18. Circuito de desplazamiento.

La siguiente figura 19 ilustra el proceso completo de preparación y limpieza de la señal.

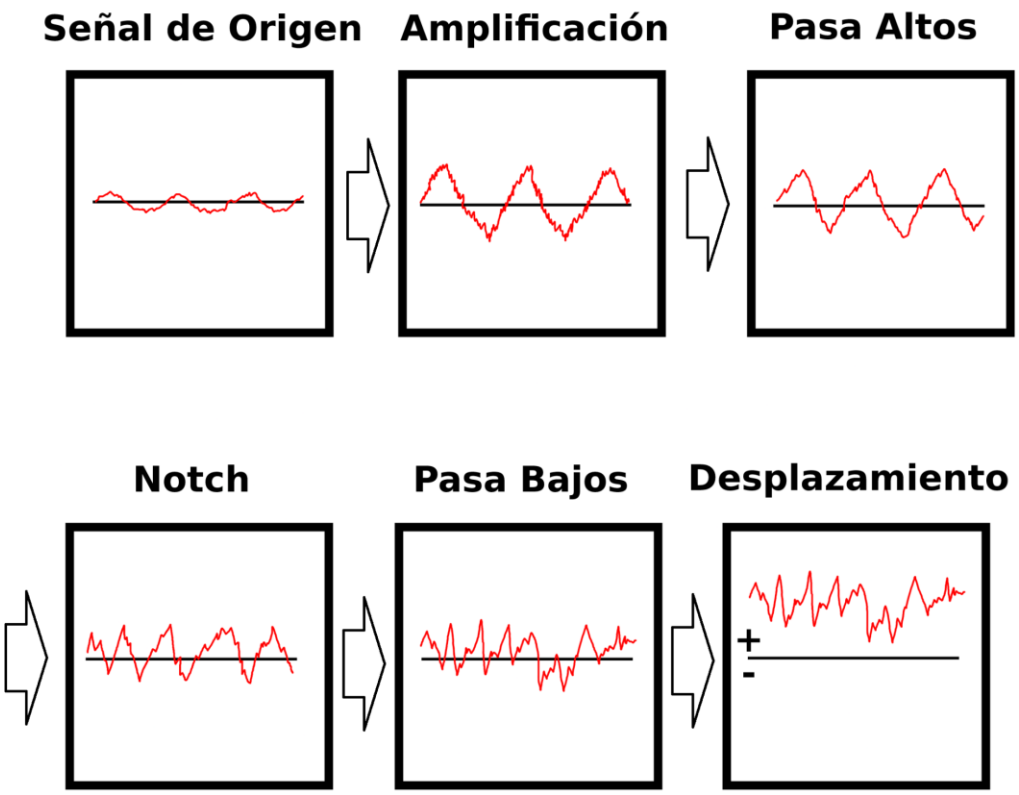

Figura 19. Proceso de amplificación y filtrado.

2.5 Visualización

Una vez que la señal está preparada llegó el momento de digitalizarla y graficarla para ser interpretada. Este proceso consta de tres etapas: **Digitalización, Filtrado Digital y Graficado.**

2.5.1 Digitalización

Para la visualización de datos primero es necesario convertir los datos analógicos a señales digitales con las que se pueda trabajar por medio de software. Para esto se utiliza un Arduino DUE (véase la Figura 20) el cual tiene la característica de utilizar un procesador ARM con 12 entradas analógicas y además una resolución de 12 bits. Esta resolución ayudará a obtener una mayor precisión en las lecturas distinguiendo entre picos más pequeños.

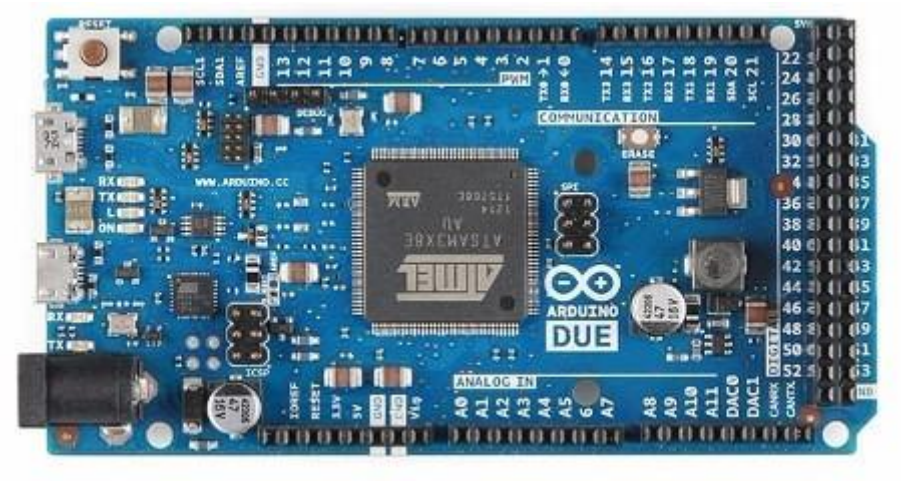

Figura 20. Arduino DUE.

Las mediciones son recibidas por los puertos A0 y A6. Ambos puertos configurador para tomar la lectura con una resolución de 12 bits. La medición se comunicará con una frecuencia de 19200 baudios por el puerto serial.

2.5.2 Filtrado digital

Se añade una etapa de filtrado digital, esta consta de un filtro notch que busca eliminar frecuencias de 50 Hz y un filtro notch de frecuencia normalizada según la frecuencia de muestreo, la cual es de 250 Hz. La señal cruda es guardada en variables “raw” para cada canal y después impresa en el puerto serial después de aplicar los respectivos filtros. El código completo puede verse en la figura 21.

```
#include <Filters.h>

#include <AH/Hardware/FilteredAnalog.hpp>
#include <AH/Timing/MillisMicrosTimer.hpp>
#include <Filters/Notch.hpp>

FilteredAnalog<12,      // Output precision in bits
               6,       // The amount of filtering
               uint32_t,// The int type for filter calculations
               analog_t // The int type for upscaled analog values
              >
  analog = A0;

FilteredAnalog<12,
               6,
               uint32_t,
               analog_t
              >
  analog2 = A6;

void setup() {
  Serial.begin(19200);
  while (!Serial);
  // Select the correct ADC resolution
  FilteredAnalog<>::setupADC();
  // Initialize the filter to whatever the value on the input is right
now
  // (otherwise, the filter is initialized to zero and you get
transients)
  analog.resetToCurrentValue();
}

// Sampling frequency
const double f_s = 250; // Hz
// Notch frequency (−∞ dB)
const double f_c = 50; // Hz
// Normalized notch frequency
const double f_n = 2 * f_c / f_s;

// Sample timer
Timer<micros> timer = std::round(19200 / f_s);

// Very simple Finite Impulse Response notch filter
auto filter1 = simpleNotchFIR(f_n);     // fundamental
auto filter2 = simpleNotchFIR(2 * f_n); // second harmonic

void loop() {
  if (timer) {
    auto raw = analogRead(A0);
    auto raw2 = analogRead(A6);
    Serial.print(filter2(filter1(raw)));
    Serial.print(", ");
    Serial.println(filter2(filter1(raw2)));
  }
}
```

Figura 21. Lectura y filtrado digital.

2.5.3 Graficado

La información se grafica usando el serial plotter del IDE de Arduino. En la figura 22 se puede observar en la línea verde que representa la señal cruda proveniente de los puertos del Arduino, y en amarillo la señal pasada por los dos filtros notch mencionados en la sección anterior.

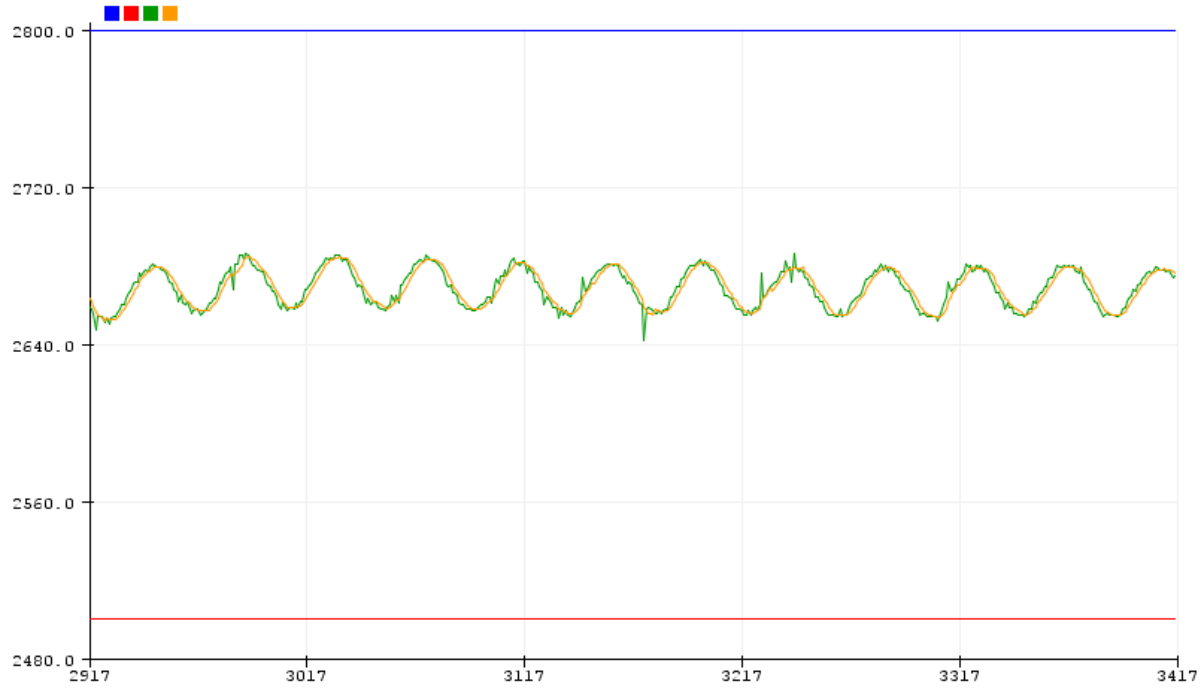

Figura 22. Señal antes y después del filtro.

La imagen de entrada es una señal oscilatoria generada por el mismo Arduino para verificar que todo el sistema estuviera amplificando y filtrando de manera correcta antes de la ingesta de la bioseñal.

Con este último paso el dispositivo está completo. Puede amplificar y mostrar una señal limpia que servirá para tomar las muestras que conformarán la base de datos para el entrenamiento del modelo.

3 El modelo

Para entrenar un modelo de redes convolucionales es necesario contar con un conjunto de datos, en específico un conjunto de imágenes de las cuales el modelo aprenderá. En la segunda parte de este capítulo hablaremos en detalle de este proceso de aprendizaje. Pero primero hablaremos del conjunto de datos.

3.1 El conjunto de Datos

A pesar de que existen conjuntos de datos libres que han sido recopilados con distintos propósitos. El problema con la transferencia de datos es replicar la manera de tomar la muestra. Ya sea por el dispositivo que fue utilizado para recopilar los datos. Por las personas que fueron voluntarias para las mediciones, o por las condiciones en donde se tomaron. Ante este problema se optó por crear el conjunto de datos desde cero.

3.1.1 Muestra

Con el dispositivo terminado, si colocamos los electrodos y comenzamos a tomar mediciones observaremos lo mismo que en la figura 23.

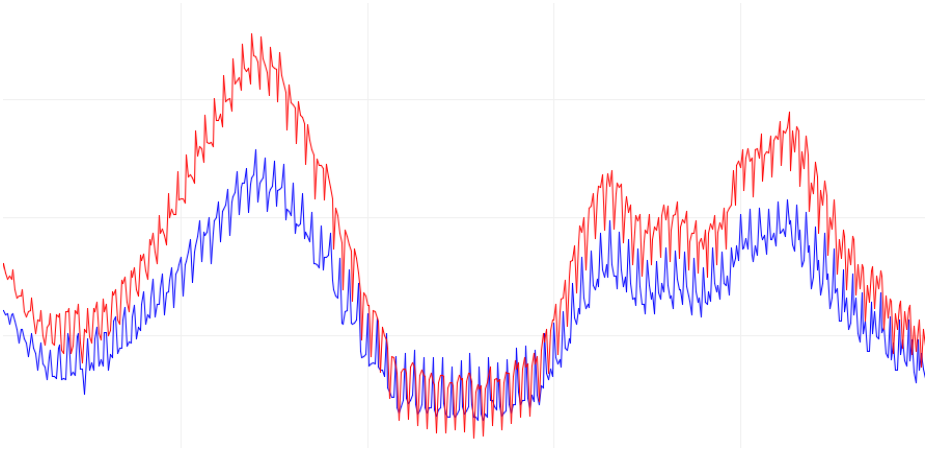

Figura 23. Lectura de ondas cerebrales.

Surgen varias dudas, ¿Cómo sabemos que nos está diciendo el cerebro?, ¿Cuándo debemos comenzar y terminar de medir?

La primera pregunta aún no tiene una respuesta concreta como mencionamos en el primer capítulo. Por lo que la idea será introducir en la red convolucional una imagen

que contenga la información necesaria para que pueda clasificarla. Siempre estarán ocurriendo activaciones neuronales en el cerebro por lo que debemos asegurarnos de que la indicación que buscamos se encuentre en la imagen, esperando que durante el entrenamiento la red encuentre esa diferencia entre todo el ruido y lo clasifique de manera correcta.

Lo que nos lleva a la segunda pregunta. Debemos estar seguros de que la señal de intención de movimiento se mide y se guarda en la imagen. Por suerte hay un método sencillo de asegurar que la muestra tomada contenga la señal deseada.

La señal P300 es un pico en las lecturas tomadas por electroencefalografía que se da 300 milisegundos después de un estímulo [8]. Por tanto, si comenzamos a tomar la muestra al mismo tiempo de la generación del estímulo y durante un segundo, garantizamos que la medición contenga la información de la intención de movimiento.

El proceso para tomar la muestra consta de tres pasos:

1. **Indicación previa del movimiento deseado:** Se muestra un mensaje con el movimiento deseado “derecha”, por ejemplo.
2. **Conteo:** Inicia un conteo de tres segundos para que la persona procese la instrucción.
3. **Activación de movimiento:** Se muestra una segunda vez el mensaje como indicación de activar el movimiento (recordando que no es necesario realizar el movimiento, simplemente tener la intención de hacerlo)

Se planteó la posibilidad de comenzar con la muestra al momento del primer mensaje. Pero la tensión que se generaba y un aumento en los errores (generar movimiento en otra dirección) complicaba el muestreo. Un conteo de tres segundos asegura que, aunque se dé la intención al momento del primer mensaje este no permanezca después de pasado el conteo, y disminuyendo la tensión y la posibilidad de error.

Una vez que se muestra el segundo mensaje comienzan a guardarse los datos de la medición en una lista. Se toman 96 muestras, este número está limitado por la frecuencia a la que procesa al procesador del Arduino y manda los datos por el puerto serial. 96 muestras es el máximo posible en un segundo.

Tomadas estas muestras se descartan las primeras 6. Esto debido al ruido generado por la apertura del puerto serial. Tomadas las 96 muestras son convertidas de formato bytes a datos de coma flotante (float64). Almacenadas en un dataframe con ayuda de la librería pandas junto con su respectiva etiqueta de comando (*Front, Right, Left, Quiet*). Finalmente (al terminar la toma de muestras) es guardado en un CSV para su futuro uso. El código utilizado para este proceso se adjunta en la figura 24.

```
import serial
from random import choice
from time import sleep
import pandas as pd
from os import system

instructions = ['quiet', 'front', 'left', 'right']

# Read serial port and save data in an array
#@execution_time
def take_measures():
    measures = []
    ser = serial.Serial('COM3', 19200)

    for _ in range(96):
        text = ser.readline()
        measures.append(text)

    for _ in range(6):
        measures.pop(0)

    format_measures = [element.decode('UTF-8') for element in
measures]

    cleaned_measures = [[float(s) for s in re.findall(r'-?\d+\.?\d*',
element)] for element in format_measures]

    return cleaned_measures

# Print a random instruction
def instruction():
    selection = choice(instructions)
    print(selection)

    for number in range(1,4):
        sleep(1)
        print(number)

    sleep(1)
    print(selection)
    return selection

# Save data
def save_data(data):
    df = pd.read_csv('./raw_dataset.csv')
    df.loc[len(df.index)] = data
```

```
    df.to_csv('./raw_dataset.csv', encoding='utf-8', index=False)


if __name__ == '__main__':
    cuantity = int(input('Ingrese la cantidad de muestras: '))
    for _ in range(cuantity):
        sleep(1)
        label = instruction()
        data = take_measures()
        save_data([label, data])

        for number in range(2,5,2):
            data2 = [[medition[0] + float(number), medition[0] -
float(number)] for medition in data]
            save_data([label, data2])

            data2 = [[medition[0] - float(number), medition[0] +
float(number)] for medition in data]
            save_data([label, data2])

        system('clear')
```

Figura 24. Recepción del puerto serial.

3.1.2 Transformación

Una vez con las muestras guardadas estas deben prepararse para su ingesta en la red. El primer paso es convertir los datos en imagen. Aunque con ayuda del plotter de arduino podemos visualizar la señal al guardar las mediciones por el puerto serial los datos se reciben de manera numérica tal y como se observa en la figura 25.

```
front,"[[2615.0, 2615.2], [2614.6, 2614.6], [2614.6, 2616.0], [2616.2, 2616.2],
[2605.4, 2596.0], [2594.8, 2594.6], [2583.4, 2583.0], [2590.6, 2588.6], [2578.0,
2579.8], [2583.0, 2577.6], [2579.4, 2589.6], [2590.4, 2596.4], [2603.8, 2604.2],
[2594.8, 2591.8], [2592.8, 2592.4], [2581.4, 2580.2], [2586.4, 2582.0], [2577.2,
2592.4], [2594.6, 2591.8], [2597.2, 2603.4], [2588.2, 2585.8], [2594.8, 2594.2],
[2581.0, 2579.8], [2588.8, 2586.0], [2575.6, 2576.6], [2580.4, 2577.0], [2577.4,
2587.2], [2588.8, 2588.0], [2592.8, 2594.6], [2583.4, 2580.6], [2588.2, 2587.4],
[2576.0, 2576.8], [2584.6, 2581.8], [2576.8, 2581.0], [2583.2, 2579.2], [2582.4,
2588.6], [2585.0, 2583.8], [2591.2, 2591.2], [2578.8, 2577.2], [2585.6, 2585.0],
[2574.8, 2575.8], [2580.2, 2574.0], [2572.6, 2581.2], [2583.2, 2581.4], [2588.6,
2591.0], [2581.2, 2578.2], [2586.0, 2585.2], [2573.4, 2573.6], [2581.6, 2579.2],
[2571.4, 2574.8], [2577.8, 2573.0], [2576.8, 2585.8], [2584.6, 2583.6], [2591.4,
2590.8], [2578.0, 2575.2], [2583.4, 2583.0], [2572.4, 2574.0], [2579.4, 2574.0],
[2571.6, 2578.8], [2581.0, 2579.0], [2585.4, 2589.6], [2581.2, 2577.4], [2585.4,
2584.8], [2572.6, 2572.2], [2581.2, 2578.8], [2569.0, 2570.4], [2574.4, 2569.8],
[2572.6, 2587.8], [2588.0, 2586.8], [2594.0, 2593.6], [2576.4, 2573.4], [2581.6,
2581.2], [2570.6, 2572.0], [2578.2, 2573.2], [2569.2, 2577.4], [2579.4, 2577.0],
[2583.2, 2588.6], [2579.8, 2577.0], [2584.6, 2583.8], [2571.4, 2571.6], [2580.0,
2578.4], [2568.6, 2570.0], [2573.0, 2569.4], [2571.0, 2580.8], [2584.2, 2583.6],
[2589.8, 2590.2], [2578.4, 2574.4], [2582.2, 2581.4], [2571.0, 2571.8], [2577.8,
2574.2]]"
```

Figura 25. Entrada del puerto serial.

Una lista de pares, cada par con la información referente a un canal. Con ayuda de matplotlib graficamos la señal en un gráfico de líneas donde el eje x representa el tiempo y el eje y la medición en milivolts (recordando que esta fue amplificada y desplazada por lo que el valor sólo representa su posición con respecto a si misma en el tiempo y no un valor real en milivolts). Por lo que vemos la primera transformación de la siguiente manera (véase la Figura 26).

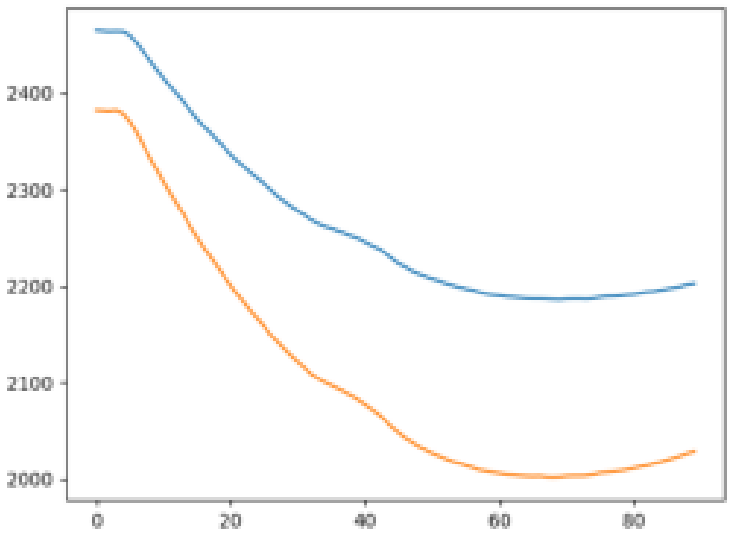

Figura 26. Datos Graficados.

El siguiente paso es eliminar los bordes generados. y convertir la imagen de un formato rgba a rgb (cuatro componentes a solo tres) (véanse las Figuras 27 y 28). Para estas modificaciones con la imagen hacemos uso de la librería skimage. Obteniendo la siguiente imagen que contiene tres componentes, cada uno su respectivo canal Red, Green o Blue.

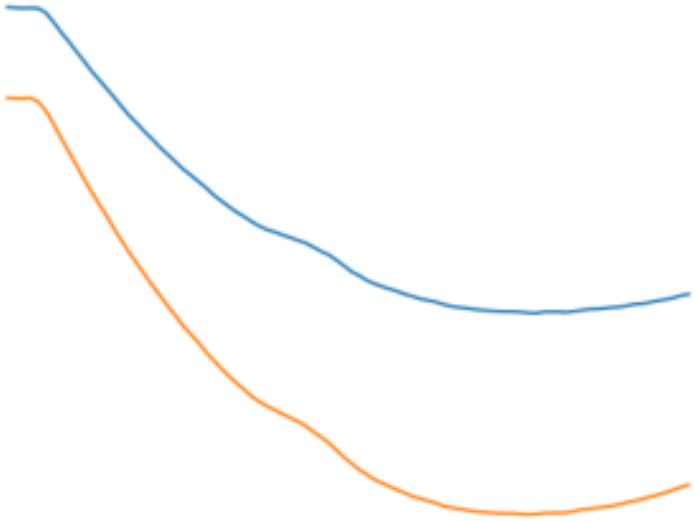

Figura 27. Imagen RGB.

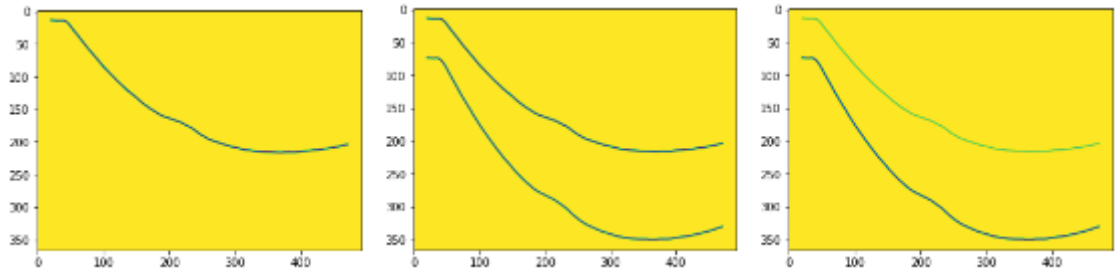

Figura 28. Canales RGB separados.

La transformación en RGB es una mera conversión para igual componentes con el formato final, HSV (véase la Figura 29). Este se compone de tres canales Hue, Saturation y Value (véase la Figura 30). Hue representa el color, Saturation la cantidad de blanco y Value la cantidad de negro. Cuando se trata de modelos de visión por computadora era común usar el valor de Saturation ya que es más robusto frente a cambios de luz externa a la imagen, variando menos que los valores RGB.

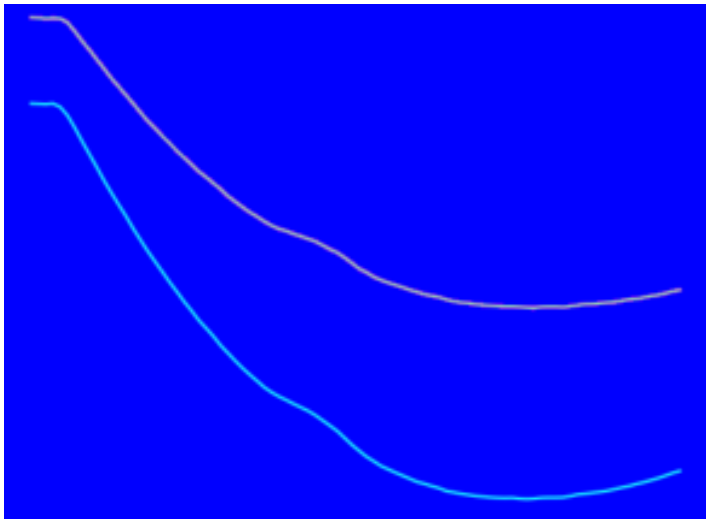

Figura 29. Imagen HSV.

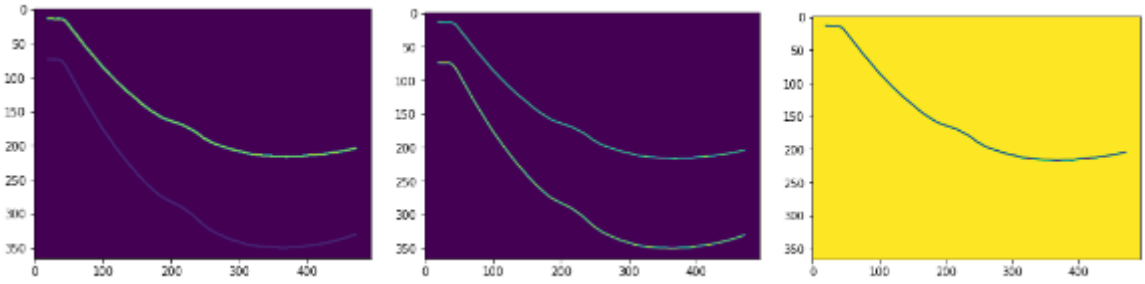

Figura 30. Canales HSV separados.

A pesar de que para el caso de las imágenes que estamos usando esto no debería afectar ya que no hay iluminación se optó por seguir el estándar. El código utilizado para este proceso se muestra en la figura 31.

```
fig, ax = plt.subplots()
    ax.plot([a for a in range(90)], [b[0] for b in test])
    ax.plot([a for a in range(90)], [b[1] for b in test])
    plt.savefig('work_image.png')
    plt.close('all')

    im = io.imread('./work_image.png')
    im = img_as_ubyte(im)

    img = im[60:426, 82:575]
    img = rgba2rgb(img)
    img = rgb2hsv(img)
    img = img[:,:,1]
    img = img_as_ubyte(img)
```

Figura 31. Procesamiento de la imagen.

3.1.3 Resultado

Después de aplicar el proceso de transformación cada muestra se ve tal y como en la figura 32.

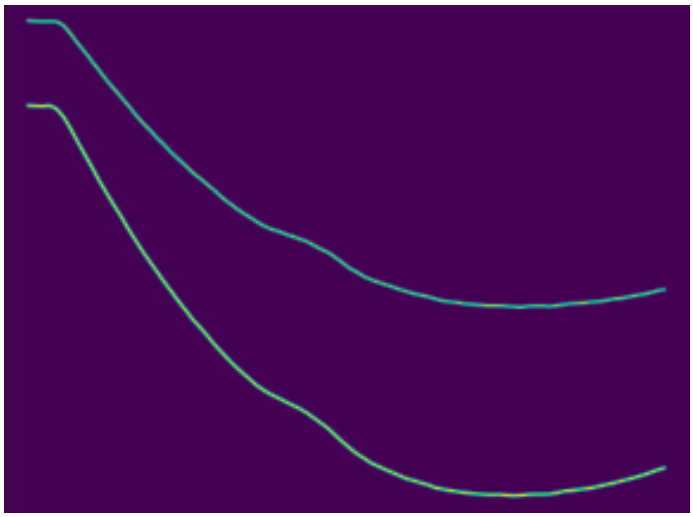

Figura 32. Imagen resultante.

La última línea del código mostrado en la página anterior realiza la conversión al formato ubyte. Esta imagen será ingresada a la red convolucional como una matriz de valores.

Una práctica común para ampliar el conjunto de entrenamiento es desplazar un poco la imagen o rotarla para aumentar la cantidad de ejemplos que serán mostrados a los modelos. Este proceso conocido como *Data Augmentation* queda más claro con la figura 33.

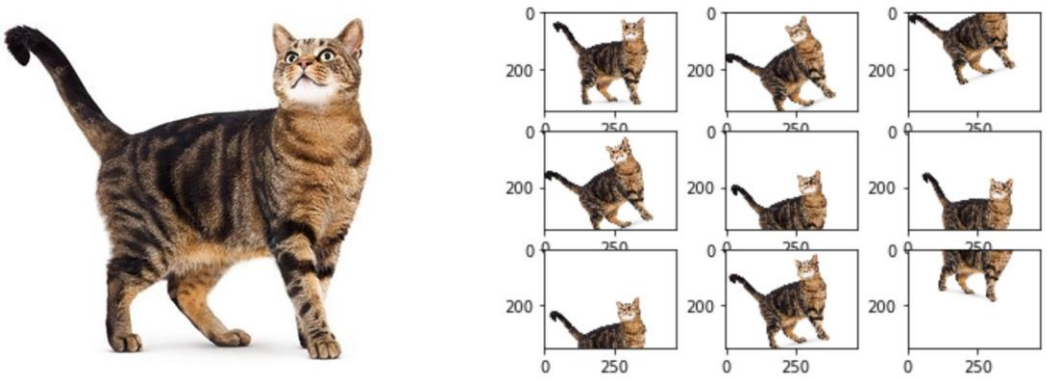

Figura 33. Ejemplo de Data Augmentation.

Esta técnica solo presenta una ventaja significativa con conjuntos de datos pequeños [9]. Lo que nos viene perfecto para aumentar la precisión del modelo. Para aplicar la técnica desplazamos los resultados de las mediciones en el eje y. Conservando su forma, picos y relación entre canales, pero simulando una variación en la impedancia del electrodo.

Una de las partes más importantes a la hora de crear el modelo es asegurarnos de que la precisión del mismo sea real. Por lo que procedemos a dividir nuestro conjunto de datos en dos partes. El conjunto de entrenamiento y el conjunto de prueba.

El conjunto de entrenamiento consta de 400 imágenes que serán “vistas” por la red convolucional. Con estas imágenes, se darán y se ajustarán los pesos del modelo.

El conjunto de prueba se compone de las 100 imágenes restantes. Estas imágenes no serán vistas por la red y validará si la red realmente aprendió a reconocer el patrón buscado. O simplemente “memorizo” las etiquetas para las imágenes que le fueron mostradas.

Los pasos finales después de dividir el conjunto son transformarlo a datos de coma flotante (float32). Y dividir los valores de cada píxel entre 255. Esto se hace con la intención de usar valores entre el 0 y 1. Que son fácilmente comprendidos por los modelos de redes, esto se debe a que son valores porcentuales donde la red puede “comprender” cual es el valor máximo y saber cuáles valores tienen mayor peso que otros. O en otras palabras, asignar una escala perfectamente comparable para todos los valores.

El 255 viene de los valores que toma la imagen para asignar color al píxel. Recordando que tomamos el canal de saturación, los valores van de 0 a 255 acercándose más al blanco o la ausencia de este color.

El último paso de esta etapa consta de declarar la forma que tendrá la imagen en el conjunto de datos, para nuestro caso es [366,493,1]. Esto implica una matriz de 366 unidades de alto por 493 de ancho y una sola dimensión. Estas mismas dimensiones serán los parámetros de la capa de entrada de la red convolucional. Esto será visto a detalle en la siguiente sección.

Las etiquetas son guardadas en conjuntos distintos de forma categórica, esto es, con valores enteros de acuerdo al siguiente diccionario ('quiet': 0, 'right': 1 , 'front': 2, 'left': 3). De esta manera terminamos con cuatro conjuntos de datos.

1. **train_images**: Imágenes de entrenamiento
2. **test_images**: Imágenes de prueba
3. **train_labels**: Etiquetas correspondientes a las imágenes de entrenamiento
4. **test_labels:** Etiquetas correspondientes a las imágenes de prueba

El código completo utilizado para la creación del *dataset* se muestra en la figura 34.

```
import pandas as pd
from skimage import io
import matplotlib.pyplot as plt

from skimage.color import rgb2hsv
from skimage.color import rgba2rgb

from skimage.util import img_as_ubyte


label_dict = {'quiet': 0, 'right': 1 , 'front': 2, 'left': 3}

train_labels = []
train_images = []
test_labels = []
test_images = []

for position in range(500):
    test = eval(df['data'][position])
    label = df['label'][position]

    fig, ax = plt.subplots()
    ax.plot([a for a in range(90)], [b[0] for b in test])
    ax.plot([a for a in range(90)], [b[1] for b in test])
    plt.savefig('work_image.png')
    plt.close('all')

    im = io.imread('./work_image.png')
    im = img_as_ubyte(im)

    img = im[60:426, 82:575]
    img = rgba2rgb(img)
    img = rgb2hsv(img)
    img = img[:,:,1]
    img = img_as_ubyte(img)

    if position < train_limit:
        train_labels.append(label_dict[label])
        train_images.append(img)
```

```
    else:
        test_labels.append(label_dict[label])
        test_images.append(img)

train_images = np.array(train_images)
test_images = np.array(test_images)
train_images = train_images.astype('float32') / 255
test_images = test_images.astype('float32') / 255

train_images = train_images.reshape(train_images.shape[0],366,493,1)
test_images = test_images.reshape(test_images.shape[0],366,493,1)

train_labels = tf.keras.utils.to_categorical(train_labels, 4)
test_labels = tf.keras.utils.to_categorical(test_labels, 4)
```

Figura 34. Preparación del conjunto de datos.

3.2 Arquitectura

La arquitectura pasó por muchas iteraciones buscando dos objetivos:

1. Conseguir la mayor precisión posible
2. Utilizar la cantidad necesaria (menor) de capas

El proceso para alcanzar un punto óptimo tuvo siguió procesos distintos para cada etapa, se hablará con más detalle del funcionamiento de cada capa en las secciones siguientes, pero primero dejamos en claro el proceso seguido para buscar el punto óptimo y la configuración final:

- **Capas:** El proceso consistió en iniciar con una sola capa convolucional, una capa flaten, seguida de una capa convolucional y una capa *softmax* de salida. A partir de este proceso se añadieron distintas capas repitiendo el entrenamiento hasta llegar al punto donde la precisión no mostraba un aumento significativo. Desde ese punto se regresó al punto donde la precisión era lo más alta, pero con la menor cantidad de capas. La configuración final consta de tres sets de capas convolucionales (Convolucional, *MaxPool*, *Dropout*), una capa *flatten* y tres sets de capas neuronales (Densas en orden de 128, 64, 32, *Dropout*) con una última capa de salida densa con activación *softmax*

- **Capa convolucional:** Se itero con el uso de kernels de distintos tamaños. comenzando de 3x3 hasta 16x16, dando los resultados óptimos el kernel de 9x9 para la primera capa, 5x5 para la segunda y 3x3 para la última capa convolucional.
- **Capa *MaxPool*:** El proceso fue el mismo que con las capas convolucionales quedando con kernels de 6x6, 5x5 y 2x2.
- **Capa *Dropout*:** El *dropout* se itero en aumentos de 0.01 hasta llegar a 0.1 y desde ese punto en incrementos de 0.1. El punto óptimo para las capas convolucionales fue de 0.2 y para las capas neuronales de 0.4.
- **Capa *Flatten*:** Actúa como aplanado de las capas convolucionales a un vector listo para entrar a la red neuronal. No tuvo una iteración.
- ***Softmax*:** Salida con cuatro neuronas activadas por la función *softmax*. No tuvo iteración.

3.2.1 Capa Convolucional

El propósito de una capa convolucional es resaltar características de la imagen [18]. Esto se logra aplicando un filtro llamado Kernel. El kernel es una matriz con un determinado orden de valores. Esta matriz será aplicada mediante un producto escalar a una fracción de la imagen [19]. Esto queda más claro con el siguiente ejemplo.

Tomemos como muestra dos kernels. El vertical y el horizontal, sus nombres están dados por su propósito, resaltar los patrones verticales y horizontales respectivamente. Si examinamos los valores que los componen, se verán de la siguiente manera. Un kernel de 3x3 vertical contendría en su primera columna valores -1, en la segunda 0 y en la tercera 1. El kernel de 3x3 horizontal tendría los mismos valores, pero acomodados por filas (véase la Figura 35).

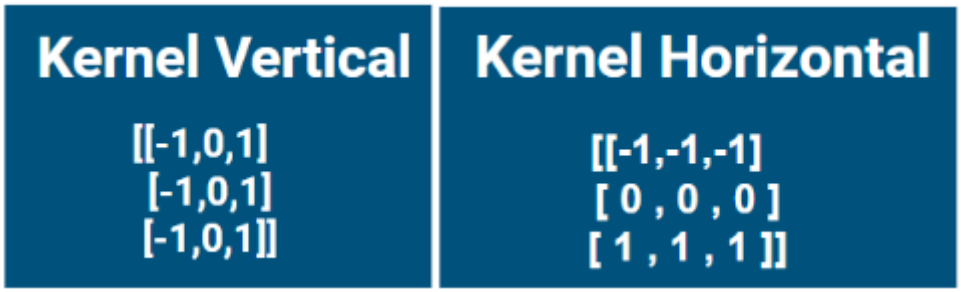

Figura 35. Kernels vertical y horizontal.

Si aplicamos el producto escalar a la primera imagen utilizando estos kernels, obtendremos dos resultados (véase la figura 36).

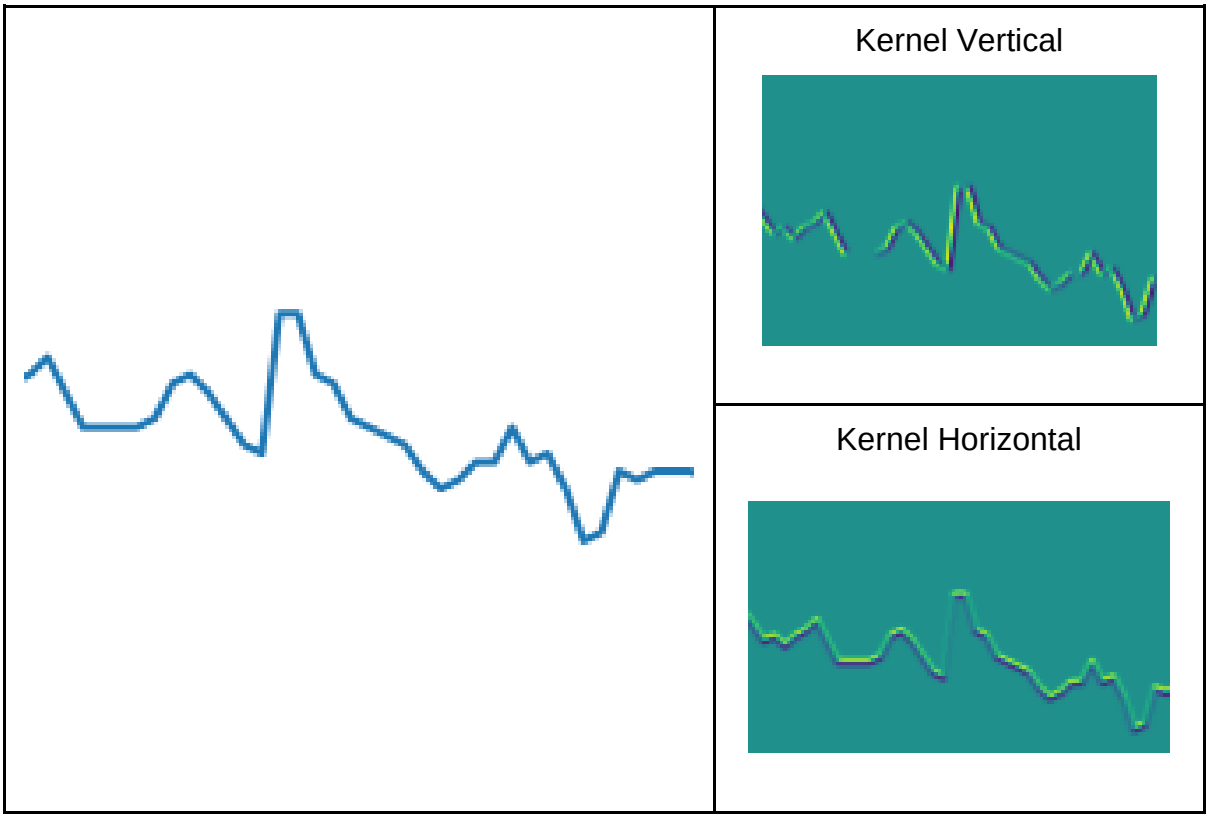

Figura 36. Resultados de kernels.

El kernel vertical resalta las líneas verticales y el kernel horizontal las líneas horizontales. Las líneas en diagonal serán parcialmente resaltadas dependiendo de su inclinación y si encontramos una línea estrictamente horizontal o vertical esta será eliminada dependiendo del filtro, tal y como podemos observar en la imagen resultante del producto escalar vertical. donde una línea horizontal ha sido borrada.

El tamaño de la matriz puede incrementar o disminuir, pero las columnas conservarán los valores dependiendo del propósito del kernel. Existen muchos tipos de kernels y se pueden crear dependiendo de los patrones que se quieran resaltar. Los kernels son usados en programas de edición de fotografía donde también sirven para enfocar, desenfocar o lograr diversos efectos sobre la imagen.

Dado que cada kernel solo sirve para resaltar una sola característica, es necesario aplicar varios kernels a la imagen en cada capa de convolución para resaltar todas las características posibles. Para nuestro caso en particular se aplican 64 kernels a la imagen, creando 64 variantes de esta. A la siguiente capa se aplican 32 y en la última capa convolucional se aplican 16. Este proceso de reducción va descartando las variantes con valores menores, variantes donde no se extrajeron características para conservar sólo las más relevantes.

Por ejemplo, para el caso de una imagen que contenga únicamente líneas verticales, no tendría sentido conservar la variante a la que se aplicó el kernel horizontal, ya que la imagen estaría vacía. Este proceso se asemeja a la manera en la que percibimos las formas a muy bajo nivel, aprendiendo a distinguir entre una casa y un automóvil porque sería de esperar que el automóvil tenga círculos (ruedas) y la casa líneas rectas (paredes).

3.2.2 Capa MaxPool

La capa de *Maxpool* reduce la imagen condensando en sus principales características. Dependiendo del tamaño del Pool dividirá la imagen en secciones de ese tamaño extrayendo el píxel que tenga mayor valor.

Por ejemplo, en la figura 37 tenemos una imagen de 4x4 donde se aplica una capa de *MaxPool* de 2x2, la imagen será dividida en secciones de 2x2 y se extraerá el píxel que contenga el valor máximo de su sección. Así, para la primera sección (roja), el valor máximo es 20, solo este pixel pasará a la siguiente etapa. Para la sección amarilla el valor máximo es 30 y este valor es el que pasa. 112 y 37 son los valores máximos de las secciones restantes.

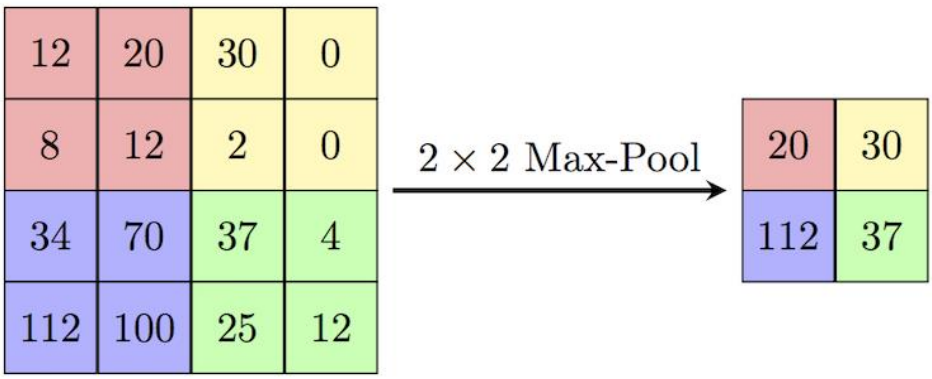

Figura 37. Ejemplo de capa MaxPool.

Después de aplicar el *MaxPool* la imagen resultante será reducida en tamaño dependiendo del tamaño seleccionado para el Pool. Para este caso pasamos de una imagen de 16 pixeles a una imagen de cuatro píxeles donde la información está condensada, conservando solo el dato más relevante de su respectiva sección.

Pesé a que existen otras opciones como el *AveragePool*, el más utilizado es el *MaxPool*, dado que después de ser aplicado conserva la característica principal. *AveragePool* presenta la desventaja de difuminar las características entre el resto de la imagen [17]. Sería el equivalente a identificar formas con miopismo.

La configuración utilizada para este proyecto fueron tres capas de *MaxPool* con tamaños de seis, cinco y dos.

3.2.3 Capa Dropout

La capa de *Dropout* cumple el propósito de evitar el sobreentrenamiento [16]. La manera más fácil de explicar el sobreentrenamiento es observando la figura 38.

Una red sobreentrenada se ajustará excesivamente al conjunto de datos de entrenamiento (línea verde). Y aunque obtenga una alta precisión, evitará que pueda clasificar de manera efectiva casos generales (Línea negra).

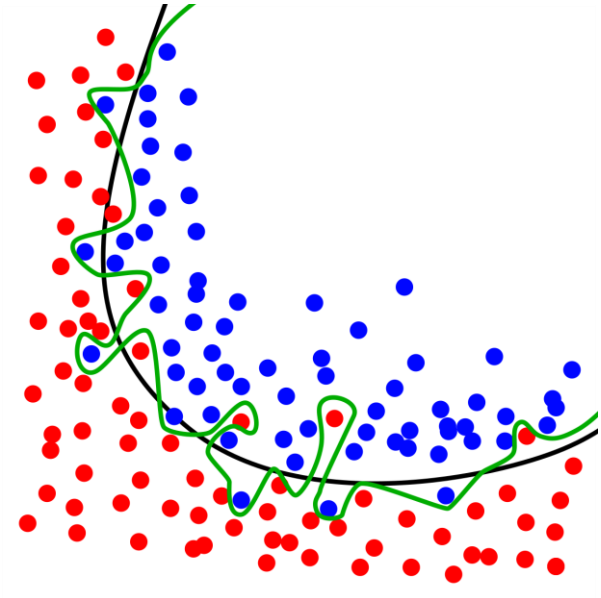

Figura 38. Sobreentrenamiento.

Para evitar este sobreajuste se utilizan capas de *Dropout*, donde, durante el proceso de entrenamiento se apagan un porcentaje de neuronas (o kernels) para que estas no “aprendan” en esa ronda de entrenamiento. La salida de las neuronas será mandada a cero simulando no estar activadas y dejando que la red trate de resolver el problema con las neuronas disponibles (véase la Figura 39).

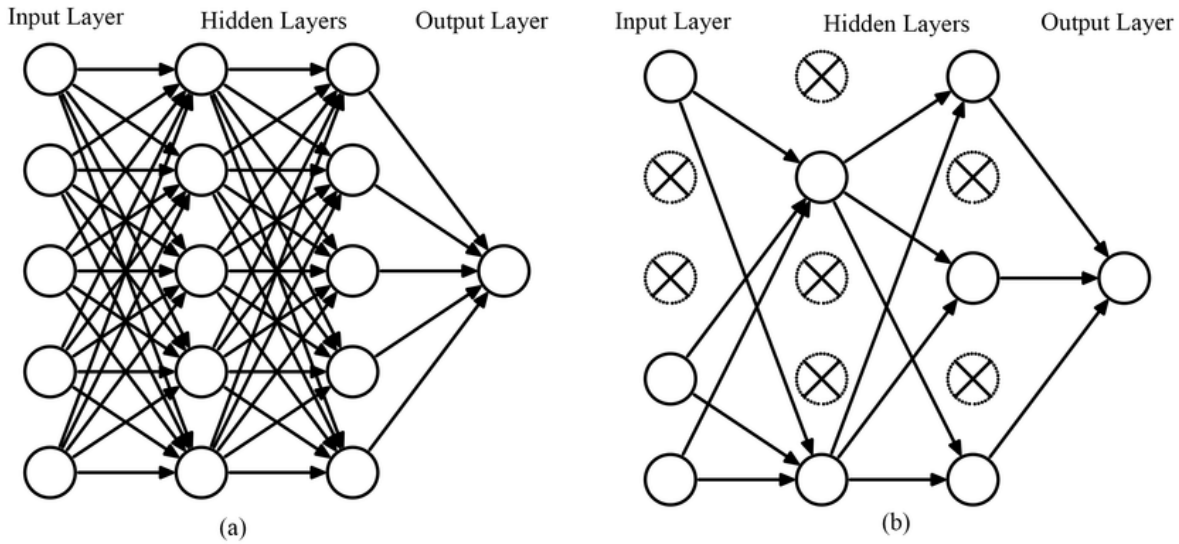

Figura 39. Red neuronal aplicando Dropout.

Este proceso podría representarse con una empresa. Tomemos por ejemplo una Empresa de Pan, donde aparte de panaderos debe de haber ingenieros, contadores, marketing, administradores, etc. Todos trabajan con el objetivo de vender pan, pero lo hacen desde distintas áreas de especialización. Al detener las neuronas en las

etapas de aprendizaje se imita el efecto de especializar para ciertos casos específicos. De esta manera en lugar de terminar con una red especializada, termina con grupos de neuronas especializadas, cada uno en ciertas características o tipos de imagen. Si bien lo ideal sería que esta especialización se repartierá uniformemente durante el entrenamiento, al tratarse de un proceso aleatorio no hay manera de controlarlo.

No todo son ventajas, ya que algo a tener en cuenta es que, si bien se evita sobreentrenar la red, el entrenamiento se entorpece, requiriendo de mayor número de iteraciones para alcanzar la misma presión que sin las capas de *Dropout*.

3.2.4 Capa Flatten

La capa *flatten* es un puente entre las capas convolucionales o pensadas en imagen (matrices) y las capas neuronales que trabajan con valores únicos. Su funcionamiento consiste en tomar los valores de cada píxel y formar un vector que será la entrada de una neurona. Una neurona por cada píxel (véase la Figura 40).

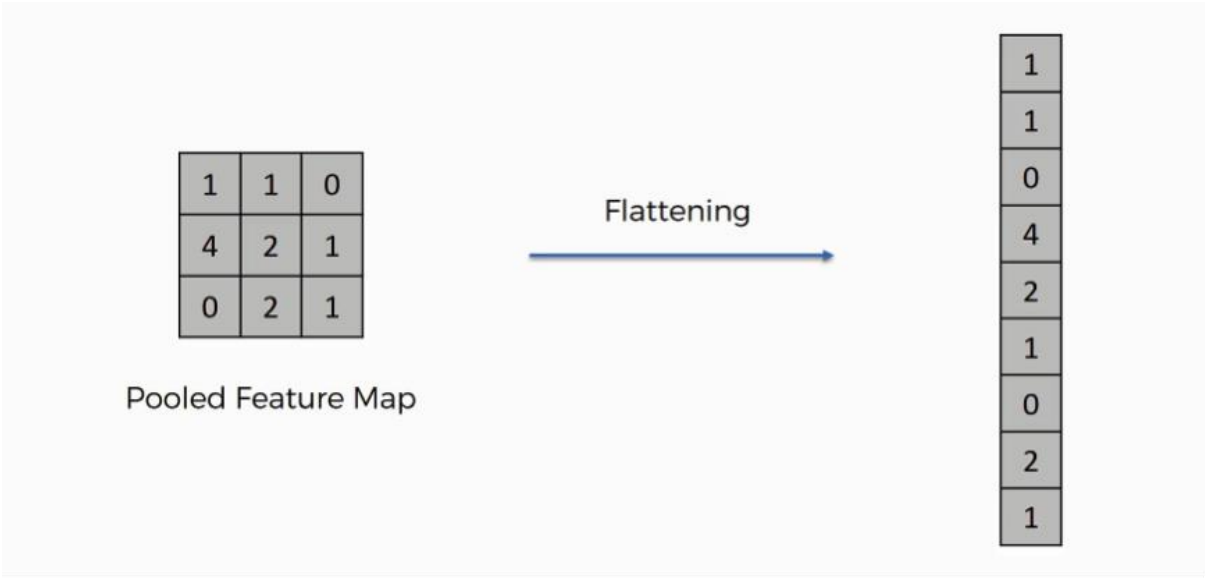

Figura 40. Ejemplo capa Flatten.

En este punto lo ideal sería que la imagen se haya comprimido lo suficiente para contener la información de todos los patrones presentes, pero que aún sea lo suficientemente grande para que no haya pérdida de ellos. Una imagen final demasiado grande podría ser aún demasiado difusa para ser “entendida” y una

imagen demasiado pequeña podría haber perdido información relevante para su correcta clasificación.

3.2.5 Red Neuronal

Las redes neuronales imitan el funcionamiento de las neuronas de nuestro cerebro y su estructura. Una neurona está formada por dendritas que reciben información, un núcleo que la procesa y un axón que las retransmite conectándose con las dendritas de más neuronas.

De manera similar una neurona artificial se compone de entradas que reciben información, esas entradas son “procesadas” por una suma ponderada y después pasan por una función de activación que las retransmite como entradas a otras neuronas (véase la Figura 41).

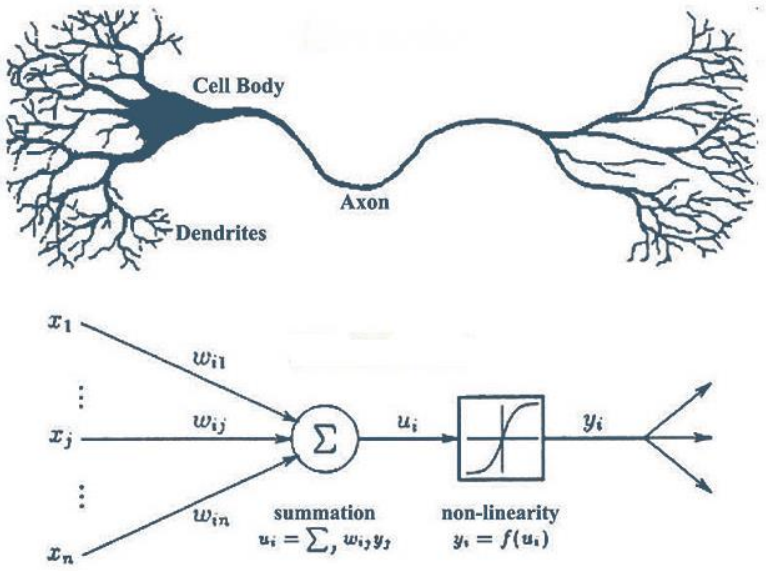

Figura 41. Similitud entre neurona real y artificial.

El proceso de entrenamiento de las redes neuronales se basa en cambiar los pesos que multiplican a las entradas de las neuronas. Esto quedará más claro al observar las figuras 42, 43, 44 y 45. Supongamos que tenemos que realizar una clasificación donde los resultados se comportan idénticos a una compuerta AND Solamente cuando los dos valores sean verdaderos la respuesta dará como resultado 1. Una neurona podría trazar una línea para dividir los resultados correctos.

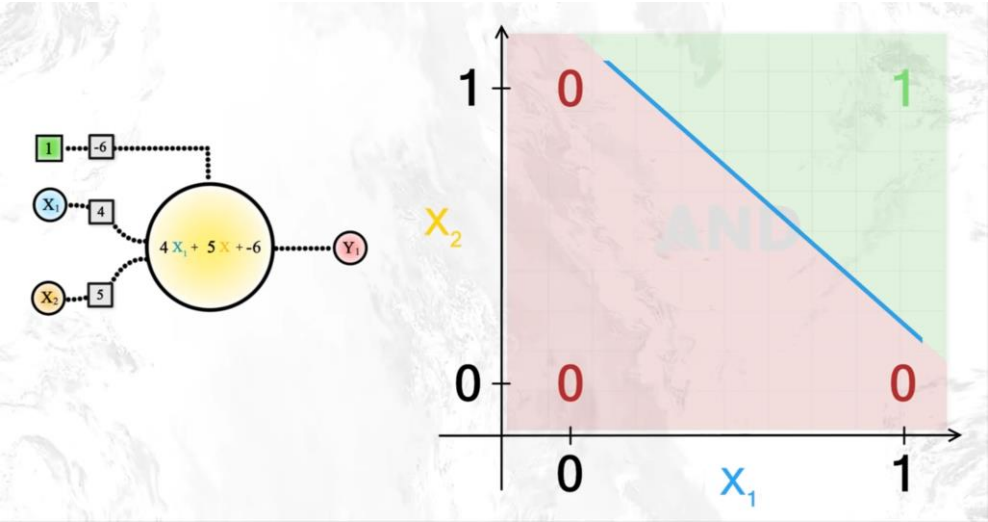

Figura 42. Neurona trabajando como compuerta AND.

Esta neurona incluso podría ajustarse y encontrar más resultados, como $4x_1 + 5x_2 - 7$. O encontrar la solución a otro tipo de problemas como el comportamiento de una compuerta OR.

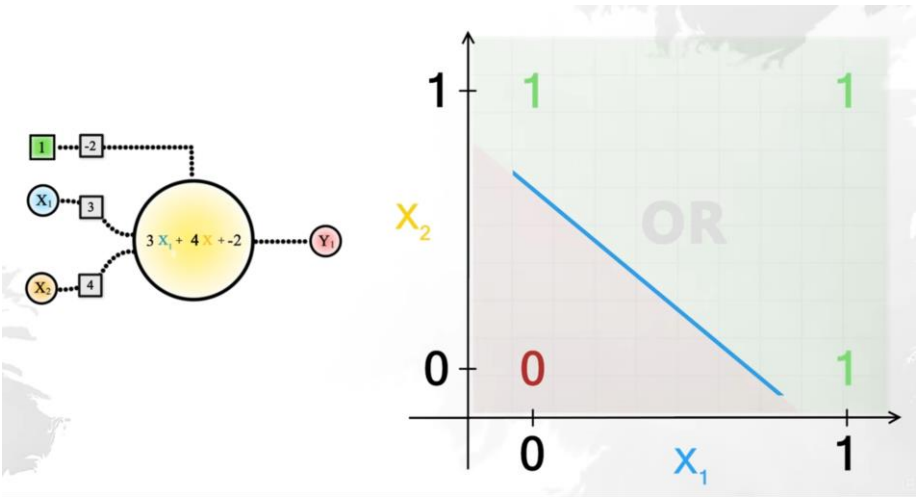

Figura 43. Neurona trabajando como compuerta OR.

Si bien, los problemas que puede resolver una neurona de manera independiente son lineales y existen maneras más simples de darles solución. El verdadero potencial se demuestra al enfrentarse a problemas más complejos. Por ejemplo, una compuerta XOR.

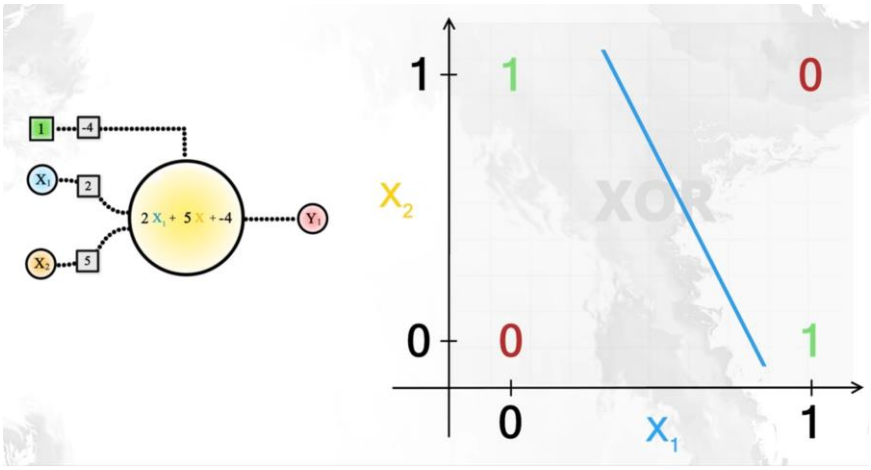

Figura 44. Una neurona sin poder resolver el comportamiento de una compuerta XOR.

Este problema imposible de solucionar con una división lineal se resuelve agregando una segunda neurona.

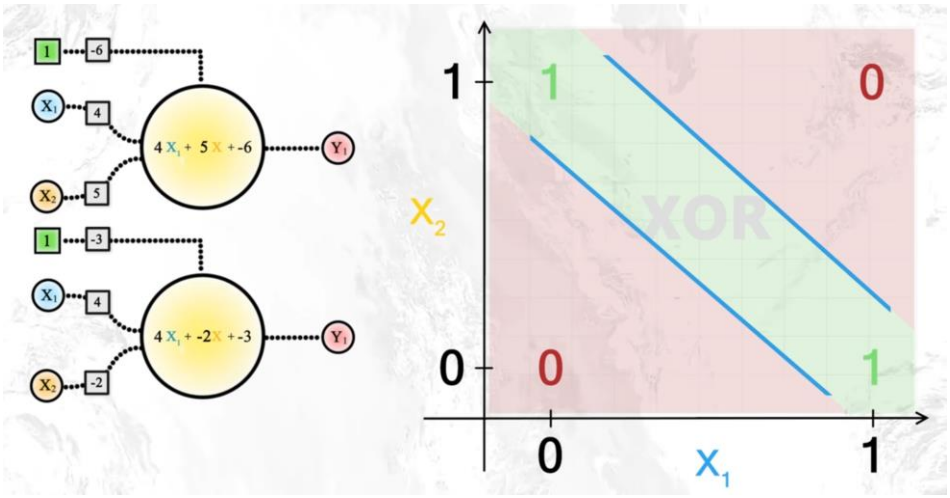

Figura 45. Dos neuronas actuando como compuerta XOR.

Esto no significa que añadiendo neuronas podamos resolver todos los problemas del mundo, pero implica que existe la posibilidad de resolver problemas cada vez más complejos (al menos de clasificación) añadiendo una misma estructura. Es mucho más sencillo crear 1,000,000 de neuronas que logren resolver el problema que intentar encontrar una ecuación que describa la solución. Repitiendo claro, que esto no se puede aplicar a todos los problemas.

Una red neuronal puede estar formada por miles de neuronas o por sólo unas pocas. Estas se clasifican en capas, las cuales son las capas de entrada que reciben la información. Capas ocultas, la cual puede ser una o varias capas formadas por múltiples neuronas, y la capa de salida. Que dependiendo del problema y la solución que queramos obtener puede ser una o varias (véase la Figura 46).

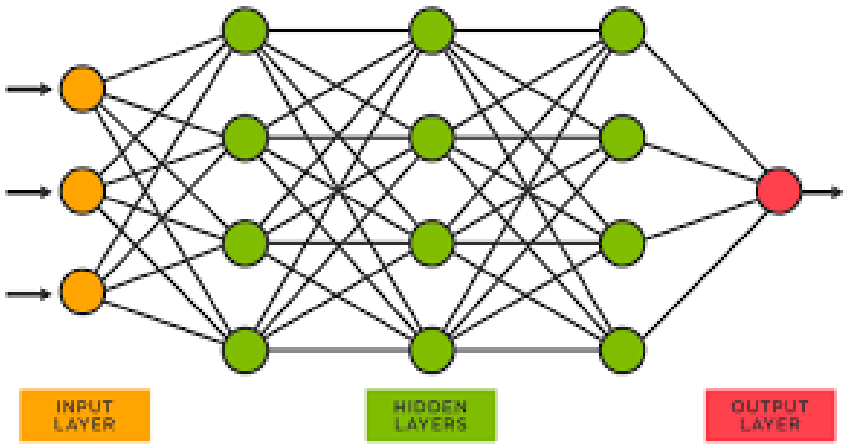

Figura 46. Clasificación de capas de una red neuronal.

El proceso de entrenamiento de una red neuronal se da con el algoritmo de backpropagation. A partir de un conjunto de entrenamiento, se ingresa un valor a la red y esta busca realizar la clasificación. Los valores están etiquetados con las respuestas correctas. En caso de que la red se equivoque se genera un error, este será distribuido hacia atrás (de ahí el nombre del algoritmo) modificando los pesos de cada una de las neuronas.

La idea de este proceso es mostrar el mismo conjunto de valores de entrenamiento a la red para que ajuste sus pesos, reduciendo o aumentando. Hasta que logre clasificar los valores ingresados con una precisión adecuada.

Para nuestro caso en específico la red se compone de tres capas ocultas densas (siendo la capa *flatten* la que actúa como capa de entrada y conexión con la red convolucional) cada una con una capa de *dropout* para evitar el sobreentrenamiento y una capa de salida densa con función de activación *softmax* que se explica a continuación.

3.2.6 Softmax

Durante todo el proceso anterior, cada una de las neuronas funcionaba con la función de activación relu.

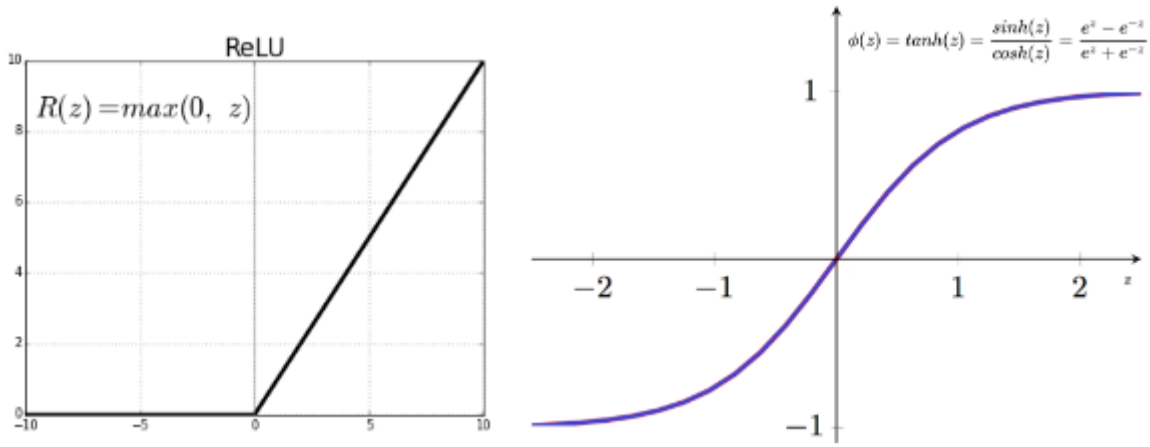

Figura 47. Función ReLU y Softmax graficadas.

Una función lineal bastante sencilla cuyos valores van entre uno y cero, descartando los valores negativos. Sin embargo, para la capa de salida hacemos uso de la función *softmax* (véase la Figura 47).

$$\sigma(\vec{z})_i = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}}$$

Ecuación 1. Función Softmax.

La función *softmax* (véase la Ecuación 1) es una función sigmoidea, perfecta para problemas de clasificación múltiple. Porque devuelve la distribución de probabilidad de las clases del modelo [13].

Es decir, para nuestro modelo con cuatro salidas (4 clasificaciones, izquierda, derecha, adelante, sin movimiento). La función *softmax* devolverá la posibilidad de que la entrada corresponda con la respectiva salida.

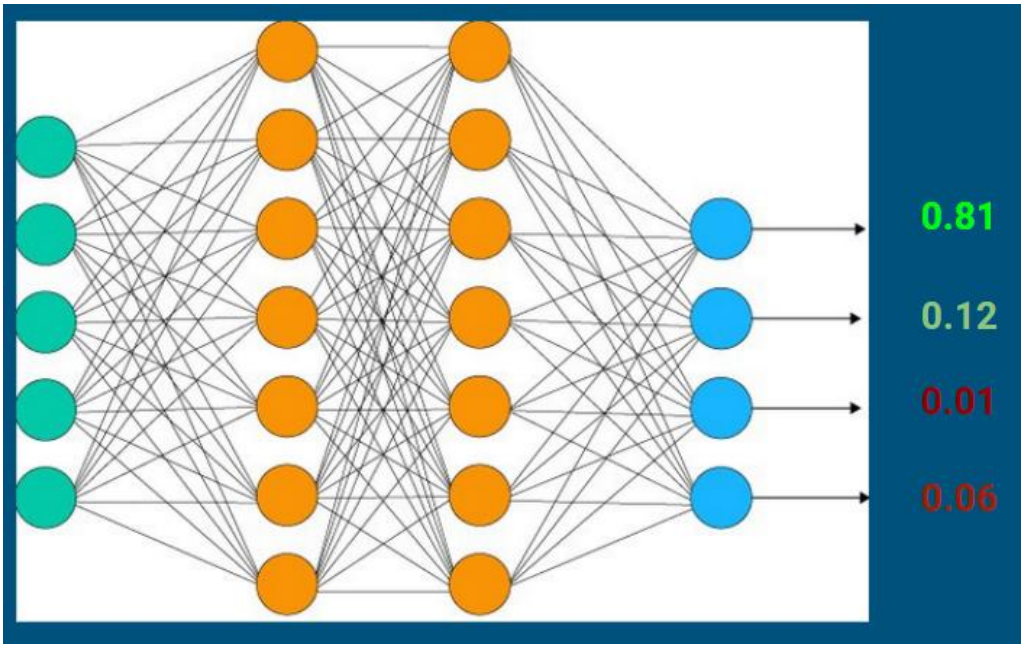

Figura 48. Función Softmax aplicada.

Esperaríamos que la salida con el porcentaje más alto nos indique la clasificación correcta (véase la Figura 48). Para nuestro caso particular, aunque una segunda salida esté relacionada, solo tomaremos el porcentaje más alto para clasificar la salida.

3.2.7 Entrenamiento

Si bien entendemos cómo las redes neuronales se ajustan para encontrar la respuesta. No sabemos por qué o cómo funciona la configuración fina o los ajustes que hacen. Sin embargo, hay ciertos mecanismos que nos ayudan a darle un pequeño empujón a la red y facilitar el proceso de entrenamiento.

3.2.7.1 Categorical Crossentropy

Categorical Crossentropy es una función de pérdida. Su objetivo es decirnos qué tan diferentes son los resultados obtenidos de los reales [12] y está dado por la sumatoria del resultado esperado multiplicado por el logaritmo del valor obtenido (véase la Ecuación 2).

$$\text{Loss} = -\sum_{i=1}^{\text{output size}} y_i \cdot \log \hat{y}_i$$

Ecuación 2. Categorical Cossentropy.

Podemos visualizarlo de la siguiente manera. La posibilidad de que un objetivo se cumpla, si vemos los resultados correctos en una clasificación múltiple es de 1.

$$[derecha, izquierda, adelante, sin\ movimiento]$$

$$[0,1,0,0]$$

Cuando vemos los resultados obtenidos veremos la posibilidad de que cada una de las salidas sea la correcta (o que esta ocurre si lo vemos desde la probabilidad).

$$[derecha, izquierda, adelante, sin\ movimiento]$$

$$[0.81, 0.12, 0.1, 0.6]$$

Al aplicar la función de pérdida le estamos diciendo a la red que tan desviada está del resultado correcto. Lo que ayuda saber qué ajustes debe de hacer para aumentar la probabilidad del resultado correcto y disminuir los incorrectos.

3.2.7.2 RMSProp

Root Mean Square Propagation es un optimizador. Su función es mejorar los cambios que hace la red a sus pesos para encontrar la respuesta correcta. Las funciones de optimización tienen como objetivo el descenso de gradiente. Es decir, llegar a la convergencia en la menor cantidad de pasos (véase la Figura 49).

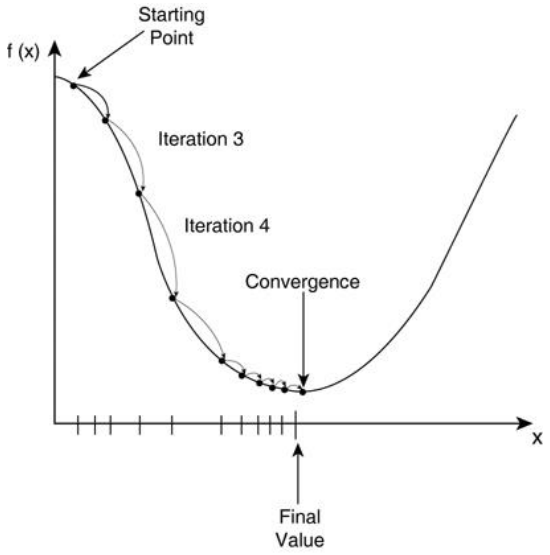

Figura 49. Descenso del gradiente aplicando el optimizador RMSProp.

La ventaja que presenta RMSProp frente a un descenso de gradiente normal o su variante Rprop es que permite hacer ajustes en el “paso” (ajuste del peso de la neurona) de manera independiente para cada neurona y está dado de la siguiente manera (Véase Ecuación 3).

$$For\ each\ Parameter\ w^j$$
$$(j\ subscript\ dropped\ for\ clarity)$$

$$\nu_t = \rho\nu_{t-1} + (1-\rho) * g_t^2$$
$$\Delta\omega_t = -\frac{\eta}{\sqrt{\nu_t + \epsilon}} * g_t$$
$$\omega_{t+1} = \omega_t + \Delta\omega_t$$

$$\eta : Initial\ Learning\ rate$$
$$\nu_t : Exponential\ Average\ of\ squares\ of\ gradients$$
$$g_t : Gradient\ at\ time\ t\ along\ \omega^j$$

Ecuación 3. Ecuaciones para aplicar RMSProp.

De acuerdo a la explicación de Ayoosh Kathuria [10]. En la primera ecuación, calculamos un promedio exponencial del cuadrado del gradiente. Esto se hace de manera independiente para cada peso y el propósito es dirigirse hasta el punto bajo del gradiente (lo cual puede variar de neurona a neurona).

Luego, en la segunda ecuación, decidimos nuestro tamaño de paso. Nos movemos en la dirección del gradiente, pero nuestro tamaño de paso se ve afectado por el promedio exponencial. Elegimos un *learning rate*, y luego la dividimos por el promedio. Esto nos ayudará a evitar rebotes entre las crestas y avanzar hacia los mínimos.

La tercera ecuación es solo el paso de actualización. El hiperparámetro p generalmente se elige para que sea 0.9. La épsilon es la ecuación 2, es para garantizar que no terminemos dividiendo por cero, y generalmente se elige para que sea 1e-10.

RMSProp presenta la ventaja de reducir el paso entre más nos acerquemos a la convergencia. Evitando saltos innecesarios y reduciendo el tiempo del entrenamiento.

3.2.7.3 Épocas y Batch Size

Las épocas y el batch size son parámetros que están relacionados con el número de iteraciones que realiza la red para entrenarse.

El Batch Size se refiere al número de ejemplos de entrenamiento que serán mostrados a la red en una iteración [20]. La época por su parte es la cantidad de veces que el conjunto de datos de entrenamiento es mostrado por completo [21]. Esto quiere decir que en una época la red ve todos los ejemplos una vez, estos son mostrados en iteraciones de acuerdo al Batch Size. El Batch Size toma especial importancia cuando la cantidad de datos es demasiado grande. No es nuestro caso por lo que durante las iteraciones siempre se mantuvo en 64.

Las épocas por su parte fueron cambiando en un rango que fue de 1 hasta 150 y dando los mejores resultados entre 20 y 150.

El código completo del modelo junto con las representaciones gráficas de la arquitectura se puede observar en las Figuras 50, 51,52 y 53.

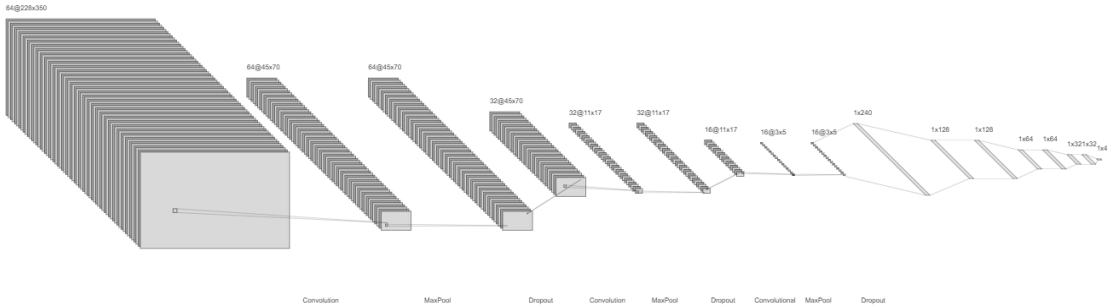

Figura 50. Arquitectura.

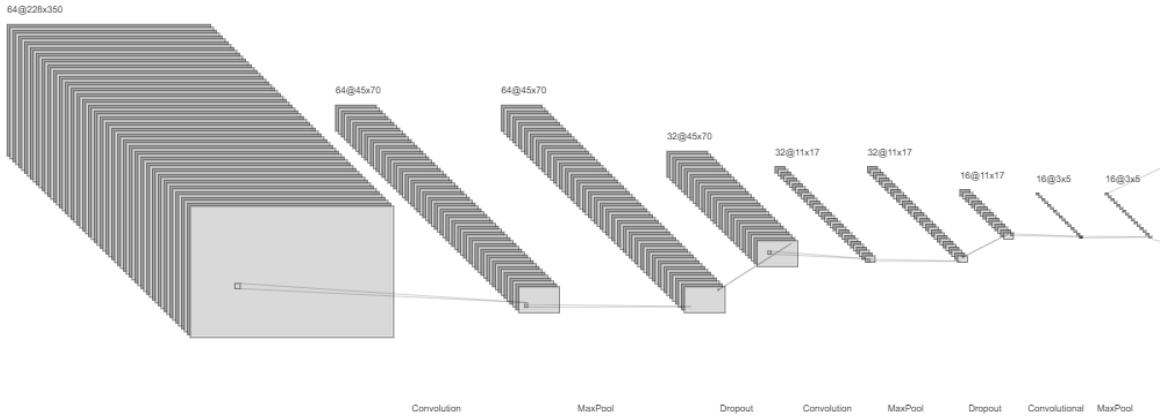

Figura 51. Sección convolucional.

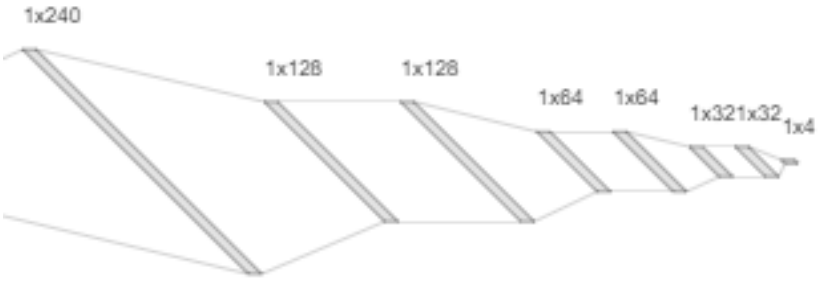

Figura 52. Sección neuronal.

```
import tensorflow as tf
from tensorflow.keras.layers import Conv2D, Dropout, MaxPooling2D,
Flatten, Dense
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import matplotlib.pyplot as plt
from skimage import io
from skimage.util import img_as_ubyte
from skimage.color import rgba2rgb
from skimage.color import rgb2hsv

df = pd.read_csv('./raw_dataset.csv')

train_limit = 400

label_dict = {'quiet': 0, 'right': 1 , 'front': 2, 'left': 3}

train_labels = []
train_images = []
test_labels = []
test_images = []

for position in range(500):
    test = eval(df['data'][position])
    label = df['label'][position]

    fig, ax = plt.subplots()
    ax.plot([a for a in range(90)], [b[0] for b in test])
    ax.plot([a for a in range(90)], [b[1] for b in test])
    plt.savefig('work_image.png')
    plt.close('all')

    im = io.imread('./work_image.png')
    im = img_as_ubyte(im)

    img = im[60:426, 82:575]
    img = rgba2rgb(img)
    img = rgb2hsv(img)
    img = img[:,:,1]
    img = img_as_ubyte(img)

    if position < train_limit:
        train_labels.append(label_dict[label])
        train_images.append(img)
    else:
        test_labels.append(label_dict[label])
        test_images.append(img)

train_images = np.array(train_images)
test_images = np.array(test_images)
train_images = train_images.astype('float32') / 255
test_images = test_images.astype('float32') / 255

train_images = train_images.reshape(train_images.shape[0],366,493,1)
test_images = test_images.reshape(test_images.shape[0],366,493,1)
```

```
train_labels = tf.keras.utils.to_categorical(train_labels, 4)
test_labels = tf.keras.utils.to_categorical(test_labels, 4)

# Model
model = tf.keras.Sequential()
model.add(Conv2D(filters = 64, kernel_size = 9, padding = 'same',
activation ='relu', input_shape = (366,493,1)))
model.add(MaxPooling2D(pool_size = 6))
model.add(Dropout(0.2))
model.add(Conv2D(filters = 32, kernel_size = 5, padding = 'same',
activation = 'relu'))
model.add(MaxPooling2D(pool_size = 5))
model.add(Dropout(0.2))
model.add(Conv2D(filters = 16, kernel_size = 3, padding = 'same',
activation = 'relu'))
model.add(MaxPooling2D(pool_size = 2))
model.add(Dropout(0.2))
model.add(Flatten())
model.add(Dense(128, activation = 'relu'))
model.add(Dropout(0.4))
model.add(Dense(64, activation = 'relu'))
model.add(Dropout(0.4))
model.add(Dense(32, activation = 'relu'))
model.add(Dropout(0.4))
model.add(Dense(4,activation = 'softmax'))
# model.summary()

# Compile model
model.compile(loss = 'categorical_crossentropy',
              optimizer = 'rmsprop',
              metrics = ['accuracy'])


model.fit(train_images, train_labels, batch_size = 64, epochs = 20)

print(model.evaluate(test_images, test_labels, verbose = 0))

model.save('./models/model_x_x.h5')
```

Figura 53. Red convolucional.

4 La interfaz

Podemos resumir el funcionamiento completo de la interfaz en 8 pasos (véase la Figura 54).

1. La medición de la señal
2. Amplificación y Filtrado Analógico
3. Digitalización y Filtrado Digital
4. Lectura Serial y Transformación
5. Graficado
6. Clasificación por la red
7. Selección de probabilidad máxima
8. Ejecución de comando

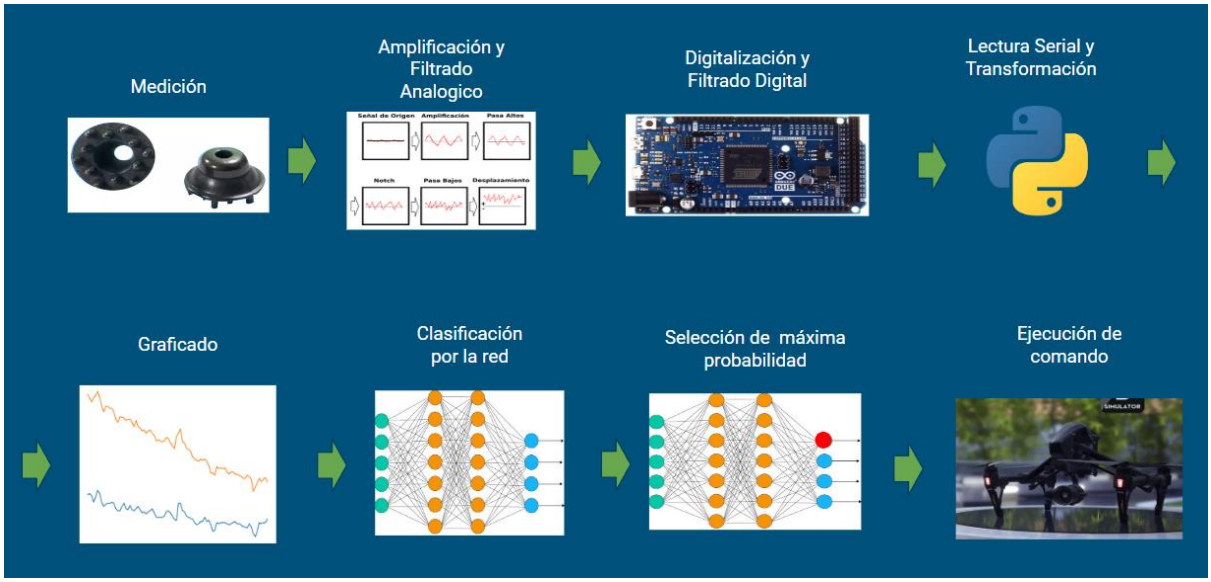

Figura 54. Procesos de la interfaz.

Los puntos uno a tres se abordan en el segundo capítulo. Los puntos cuatro a siete en el tercer capítulo. Y ahora procederemos a describir el funcionamiento de la ejecución del comando.

4.1 DJI Flight Simulator

DJI es una empresa de origen chino, dedicada a la fabricación de drones. Como parte de sus productos, ofrecen un simulador (véase la Figura 55) para aprender a volar sus drones sin el riesgo de dañar uno, con la ventaja de poder conectar vía bluetooth sus controles con el simulador. Aunque para acceder al software completo se requiere de haber adquirido uno de sus productos, el simulador permite el acceso gratuito a contenido limitado. Lo cual vino perfecto a este proyecto para realizar la simulación sin el riesgo de dañar un dron. El simulador provee de un entorno virtual donde se puede volar el dron y cuenta también con simulación de impactos

Figura 55. Simulador DJI.

Si bien se puede conectar el control real del dron al simulador, es posible usar el teclado para controlarlo. La simulación de los joysticks (el objetivo a reemplazar) se lleva a cabo con las teclas de la figura 56.

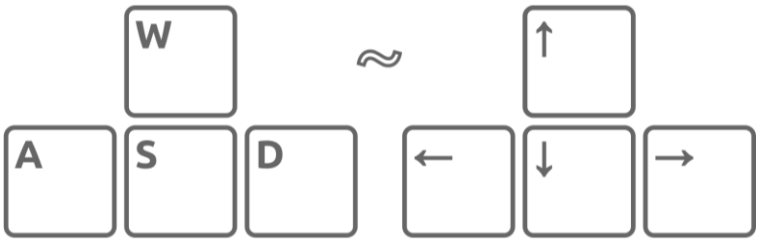

Figura 56. Teclas utilizadas para simular los joysticks del control.

Las teclas objetivo a reemplazar son:

- D - Derecha
- ↑ - Adelante
- A - Izquierda

4.2 La conexión

El proceso inicia haciendo uso de una rudimentaria diadema impresa en 3D (véase la Figura 57) donde se colocan los electrodos secos.

Figura 57. Prototipo de diadema.

Los electrodos comienzan a sensar y a mandar la señal al Arduino. Mismo que filtra y digitaliza la señal después de pasar por las ya mencionadas etapas de amplificación y filtrado. La señal es mandada por el puerto serial y recibida por un script en python que se encarga de recibir los datos, transformarlos, generar la imagen y con ella realizar una predicción con ayuda del modelo entrenado.

Del modelo solo se toma la neurona con mayor probabilidad y dependiendo de cuál haya sido se ejecuta interpreta el comando (véase la Figura 58).

```
{'quiet': 0, 'right': 1 , 'front': 2, 'left': 3}
```

Figura 58. Equivalencia de comandos.

Para ejecutar ese comando en el simulador, utilizamos la librería pyautogui. La cual nos permite simular pulsaciones del teclado o clicks del mouse. Con el simulador abierto el comando es ejecutado, simulando que la tecla se ha presionado (véase la Figura 59).

Figura 59. Dron en el simulador.

El código completo para este proceso se muestra en la figura 60.

```
from numpy import matrix
import serial
import pyautogui
import pandas as np
import tensorflow as tf
import matplotlib.pyplot as plt
from skimage import io
from time import sleep
from skimage.color import rgb2hsv
from skimage.color import rgba2rgb
from skimage.util import img_as_ubyte

label_dict = {'quiet': 0, 'right': 1 , 'front': 2, 'left': 3}

new_model = tf.keras.models.load_model('./models/model_340_137.h5')


def get_key(val):
    for key, value in label_dict.items():
         if val == value:
             return key

def take_measures():
    measures = []
    ser = serial.Serial('COM3', 19200)

    for _ in range(96):
        text = ser.readline()
        measures.append(text)

    for _ in range(6):
        measures.pop(0)

    format_measures = [element.decode('UTF-8') for element in
measures]

    cleaned_measures = [[float(s) for s in re.findall(r'-?\d+\.?\d*',
```

```
element)] for element in format_measures]

    return cleaned_measures


def create_img(measures):
    fig, ax = plt.subplots()
    ax.plot([a for a in range(90)], [b[0] for b in measures])
    ax.plot([a for a in range(90)], [b[1] for b in measures])
    plt.savefig('work_image.png')
    plt.close('all')


def convert_img_to_matrix():
    im = io.imread('./work_image.png')
    im = img_as_ubyte(im)

    img = im[60:426, 82:575]
    img = rgba2rgb(img)
    img = rgb2hsv(img)
    img = img[:,:,1]
    img = img_as_ubyte(img)
    img = img.astype('float32') / 255
    img = img.reshape(1,366,493,1)
    return img


def predict(img):
    prediction = new_model.predict(img)
    return(prediction[0].argmax())


sleep(10)
while True:
    create_img(take_measures())
    prediction = predict(convert_img_to_matrix())
    print(get_key(prediction))

    if prediction == 0:
        sleep(.5)
    elif prediction == 1:
        pyautogui.keyDown('d')
        sleep(.25)
        pyautogui.keyUp('d')
    elif prediction == 2:
        pyautogui.keyDown('up')
        sleep(1)
        pyautogui.keyUp('up')
    elif prediction == 3:
        pyautogui.keyDown('a')
        sleep(.25)
        pyautogui.keyUp('a')
```

Figura 60. Comunicación con el simulador.

4.3 Resultados

Después de iterar repetidamente, el mejor resultado fue con una precisión del **86%** con una pérdida de **0.5884.** A continuación, la tabla 1 compara trabajos similares para dar contexto de los resultados.

Trabajo	Diseño E Implementación de Interfaz Cerebro-Computacional para el Control de Vuelo de Vehículo Aéreo no tripulado usando señales cerebrales a través de Headset Electroencefalografo [11]	Seperability of four-class motor imagery data using independent components analysis	Este trabajo
Autor	Marco Antonio Mamani Musaja	Robert Leeb	
Equipo de Medición	Emotiv Insight	Equipo Médico EEG	Arduino Due
Tipo de electrodo	Semisecos Polímero	Chapa de Oro	Secos
Método de Clasificación	Clasificador Cognitivo Emotiv	Extracción de Características	Redes Convolucionales
Resultado Teórico	28% ~ 32%	63.3% ~ 69.2%	32% ~ 87%
Resultado Práctico	27%	33% ~ 84%	~ 30%

Tabla 1. Comparación de resultados obtenidos.

Si bien, la precisión alcanzada por el modelo puede parecer muy alta, hay una clara diferencia entre los resultados teóricos y los resultados prácticos. Esto puede verse atribuido al tiempo de muestreo. Ya qué es distinto el funcionamiento del cerebro en un día y una hora específicos. Al entorno, sensaciones y estado emocional de la persona al momento de las pruebas prácticas.

Los resultados obtenidos son mejores o comparables con los de otros estudios presentados, además se debe tener en consideración que el equipo de medición utilizado es mucho más económico e incluso rústico. Consecuentemente, el método de clasificación por redes convolucionales es mejor que la extracción de características o el clasificador cognitivo proveído por emotiv.

5 Conclusiones y Trabajo Futuro

En este trabajo de tesis de maestría se presentó una interfaz de control cerebro-computadora de un dron mediante redes convolucionales. Después de observar los resultados obtenidos, se concluyó que las redes convolucionales son una excelente opción para filtrar el ruido proveniente de todas las redes neuronales de nuestro cerebro y lograr destacar las señales relevantes. Aunque los resultados prácticos están lejos de ser los deseados para aplicaciones reales (donde los fallos pueden traer grandes consecuencias) su similitud en precisión teórica con otras técnicas exploradas en estudios previos es similar y a veces mejor. Aunado a esto se destaca la variable de un equipo de medición considerablemente económico con respecto a estos estudios mencionados. Estas afirmaciones se hacen al comparar los trabajos obtenidos con trabajos de investigación similares (Véase la Tabla 10)

Si bien las redes convolucionales demostraron ser un excelente método para clasificar patrones complejos, presentan dos dificultades. En primer lugar, sólo puede clasificar correctamente información similar a la que ha visto previamente y para lograr inferencias adecuadas necesita una base de entrenamiento amplia. Esto nos lleva al segundo problema, donde muchas veces obtener la información de entrenamiento no es tan sencillo. Así que entre más amplia sea la base de datos, mejores resultados se tendrán. Si tomamos muestras de los comandos con todas las emociones posibles (felicidad, enojo, tristeza, etc.), en distintos entornos (exterior, interior, movimiento, etc.) y para distintas personas, seguramente logremos obtener un modelo con excelente precisión en todas las situaciones.

Por tanto, el método demuestra ser favorable al lograr resultados similares o superiores a otras técnicas, además de tomar en cuenta la considerable diferencia de usar un hardware mucho más accesible.

Una de las limitantes al trabajar con ondas cerebrales, es el costo de los equipos. Al ver que esta limitante puede ser vencida con una heurística como las redes convolucionales, el trabajo futuro se centrará en tres líneas:

1. Explorar el impacto de un conjunto de datos de entrenamiento amplio con variaciones de todos los entornos, emociones, estímulos que ayude a la red a distinguir las señales correctas de entre todo el ruido posible.
2. Agregar al conjunto de datos de entrenamiento los datos de más personas, para establecer si el ruido se da de la misma manera en distintos cerebros y cómo esto puede afectar o ayudar a obtener mejores resultados.
3. *Stable Diffusion* [23] junto con *Dreambooth* [22] permiten transformar texto en imágenes con la adición de poder realizar un *fine tuning* al modelo (*Stable Difussion*) y con pocos ejemplos, (entre 5 y 6) este aprenda a identificar nuestro rostro para generar imágenes completamente nuevas a partir de ellos.

   Se podría construir un modelo robusto que clasifique ondas cerebrales y después realizar un *fine tuning* con pocos ejemplos esperando lograr resultados similares a los logros de estos dos modelos.

Referencias

[1] Vidal, J.J., Buck, M.D., Olch, R.H., & Ramchandani, T.D. (1974). Biocybernetic Control in Man-Machine Interaction: Final Technical Report 1973 - 1974.

[2] Walker, M. (2018). Why we sleep. Penguin Books.

[3] Duhigg, C. (2013). *The power of habit*. Random House Books.

[4] Palacios, L. (2002). Breve historia de la electroencefalografía. Acta Neurológica Colombia.

[5] Musk E, Neuralink (2019). An Integrated Brain-Machine Interface Platform With Thousands of Channels. Journal of Medical Internet Research.

[6] Desmurget Michel (2012). Conscious motor intention emerges in the inferior parietal lobule. Current Opinion in Neurobiology.

[7] M Naeem (2006). Seperability of four-class motor imagery data using independent components analysis. Journal of Neural Engineering. Graz University of Technology.

[8] Cortés A. (2014). Prototipo para la adquisición y visualización de señales de eeg para la onda P300. Universidad De San Buenaventura Facultad de Ingeniería.

[9] Xie Q., Dai Z., Hovy E., Luong M. Le Q. (2019). Unsupervised Data Augmentation for Consistency Training. Developers Corner. Cornell University.

[10] Kathuria A. Intro to optimization in deep learning: Momentum, RMSProp and Adam. PaperspaceBlog.

[11] Mamani M. (2018). Diseño e implementación de interfaz cerebro computacional para el control de vuelo de vehículo aéreo no tripulado usando señales cerebrales a través de Headset Electroencefalógrafo. Universidad Nacional de San Agustín de Arequipa.

[12] Zhang Z. (2018) Generalized Cross Entropy Loss for Training Deep Neural Networks with Noisy Labels. Meinig School of Biomedical Engineering. Cornell University

[13] Bridle J. (1990) Training Stochastic Model Recognition Algorithms as Networks can lead to Maximum Mutual Information Estimation of Parameters

[14] Brownsett S. (2009) The Contribution of the Parietal Lobes to Speaking and Writing. Division of Neuroscience and Mental Health and Medical Research Council Clinical Sciences Centre, Imperial College, Hammersmith Campus.

[15] Koudelková Z. (2018) Introduction to the identification of brain waves based on their frequency. Faculty of Applied Informatics, Tomas Bata University.

[16] Srivstava N. (2014) Dropout: A Simple Way to Prevent Neural Networks from Overfitting. Journal of Machine Learning Research.

[17] Bieder F. (2021) Comparison of Methods Generalizing Max- and Average-Pooling. Cornell University.

[18] O'shea Keiron (2015) An Introduction to Convolutional Neural Networks. Cornell University.

[19] Mairal J. (2014) Convolutional Kernel Networks. Cornell University.

[20] Devarakonda A. (2018) AdaBatch: Adaptative Batch Sizes for Training Deep Neural Networks. Computer Science Division. University of California.

[21] Rao S. (2020) Significance Of Epochs On Training A Neural Network. International Journal of Scientific & Technology Research.

[22] Ruiz N. (2022) DreamBooth: Fine Tuning Text-to-Image Diffusion Models for Subject-Driven Generation. Google Research.

[23] Ho J. (2022) Classifier-Free Diffusion Guidance. Brain Team. Google Research.