Universidad Autónoma de Querétaro

Facultad de Ingeniería

Ingeniería Física

IDENTIFICACIÓN DE MEZCLAS BINARIAS USANDO REDES NEURONALES ARTIFICIALES

Tesis

Que como parte de los requisitos para obtener el grado de

Ingeniero Físico

Presenta:

Evelyn Zuñiga Cornejo

Dirigido por:

Dr. Jorge Luis Domínguez Juárez

Co-dirigido por:

Dr. Mario Alan Quiroz Juárez

Universidad Autónoma de Querétaro
Facultad de Ingeniería
Licenciatura en Ingeniería Física

Identificación de mezclas bioquímicas binarias usando redes neuronales artificiales

TESIS

Que como parte de los requisitos para obtener el grado de
INGENIERO FÍSICO

Presenta:
Evelyn Zuñiga Cornejo

Dirigido por:
Dr. Jorge Luis Domínguez Juárez

Co-dirigido por:
Dr. Mario Alan Quiróz Juárez

Dr. Jorge Luis Domínguez Juárez Presidente	Firma
Dr. Mario Alan Quiróz Juárez Secretario	Firma
Dra. María Lucero Gómez Herrera Vocal	Firma
MC. Jesús Valdés Hernández Sinodal	Firma

Centro Universitario, Santiago de Querétaro, Qro.

Dedicado a

Todas las personas que me apoyaron durante el proceso.
Pero sobre todo, a mí misma,
por creer en mis sueños,
por superar las dificultades que se presentaron,
y por confiar en mi capacidad para lograrlo.

Agradecimientos

Agradezco a todos mis amigos y familiares que me apoyaron durante este largo viaje que fueron mis estudios universitarios, así como a los docentes e instituciones involucrados en mi formación. Particularmente expreso mi agradecimiento a los siguientes:

- A la Dirección General de Asuntos del Personal Académico (DGAPA) UNAM por el apoyo otorgado a través del Programa de Apoyo a Proyectos de Investigación e Innovación Tecnológica bajo el proyecto TA101023.
- Al Consejo Nacional de Humanidades, Ciencias y Tecnologías por el apoyo otorgado a través del proyecto CF-2023-I-1496.
- A la Universidad Autónoma de Querétaro y a la Facultad de Ingeniería, por brindarme los medios para estudiar una ciencia tan maravillosa como es la física. A cada docente de la carrera de Ingeniería Física por compartir sus conocimientos y motivarme a seguir aprendiendo.
- A mis asesores de tesis por el apoyo durante este trabajo. Al Dr. Mario Alan Quiroz Juárez, quien me brindó la oportunidad de trabajar en este tema con él, me permitió asistir a un congreso nacional de física, y me guió durante todo el proceso para poder concluir este trabajo. Al Dr. Jorge Luis Domínguez Juárez, quien prestó su laboratorio, su conocimiento y su trabajo para poder obtener las imágenes utilizadas.
- A mis padres, Liliana Cornejo y Antonio Zuñiga, por las oportunidades que me brindaron, por su amor y su apoyo durante todos mis estudios. A mi madre, quien siempre cuidó de mí y se aseguró de que comiera adecuadamente y llegara con bien a mi destino. A mi padre, quien pagó mis estudios y me apoyó en mi elección de carrera.
- A Luz Benítez, una segunda madre para mi, por su apoyo en mi desarrollo personal y profesional y por confiar en mi y en mis decisiones.
- A Jordi Ancona, mi mejor amigo de la carrera, quien hizo mis clases más divertidas y siempre resolvió mis dudas.
- A Eduardo Castellanos, quien se desveló acompañándome en muchas ocasiones, me ayudó con más de una tarea y me apoyó en más de un proyecto durante gran parte de mi formación.

Resumen

La presencia y cuantificación de alcoholes en agua es un aspecto fundamental en diversos procesos de manufactura, control de calidad de productos comerciales, y en procesos químicos orientados a la generación de energías renovables en medios acuosos. En particular, los alcoholes con bajo peso molecular, como el metanol, etanol y propanol, son completamente miscibles en agua, lo que permite la formación de mezclas en cualquier proporción. Si bien existen métodos que permiten cuantificar estas proporciones, muchos de ellos son invasivos, lo que imposibilita la recuperación y reutilización de la muestra. Además, otros métodos requieren instalaciones y equipos especializados de alto costo, lo cual puede restringir su disponibilidad y aplicabilidad en diferentes contextos. En respuesta a estas limitaciones, esta tesis propone el desarrollo de un método no invasivo para la identificación de mezclas bioquímicas binarias de agua y alcohol en distintas concentraciones, utilizando únicamente la información proveniente de los patrones de difracción generados mediante espectroscopía fototérmica. Dado el notable desempeño de las redes neuronales artificiales en el reconocimiento y clasificación de patrones, el método propuesto se basa en una red neuronal con una única capa oculta de neuronas sigmoides y una capa de salida con neuronas softmax. Esta red neuronal es entrenada con imágenes adquiridas por una cámara CCD alineada en el montaje experimental de espectroscopia fototérmica. En una etapa siguiente y explotando la baja dimensionalidad de la red neuronal propuesta, se desarrolla una aplicación móvil que integra el algoritmo de identificación. Esta aplicación es desarrollada en el lenguaje de programación Kotlin para el sistema operativo Android. Bajo este enfoque no solo se facilita la identificación de mezclas agua-alcohol, sino que también se ofrece una solución accesible y portátil.

Palabras clave: Mezclas agua-alcohol, redes neuronales, identificación de alcoholes, aplicación móvil

Abstract

The presence and quantification of alcohols in water is a fundamental aspect of various manufacturing processes, quality control of commercial products, and chemical processes aimed at generating renewable energy in aqueous media. In particular, low-molecular weight alcohols, such as methanol, ethanol, and propanol, are completely miscible in water, allowing for the formation of mixtures in any proportion. While there are methods to quantify these proportions, these are invasive, making it impossible to recover and reuse the sample. Additionally, other methods require specialized and costly equipment, which may limit their availability and applicability in different contexts. In response to these limitations, this thesis proposes a non-invasive method for identifying binary biochemical mixtures of water and alcohol in various concentrations, using only the information obtained from diffraction patterns generated by photothermal spectroscopy. Given the remarkable performance of artificial neural networks in pattern recognition and classification, the proposed method is based on a neural network with a single hidden layer of sigmoid neurons and an output layer with softmax neurons. This neural network is trained with images acquired by a CCD camera aligned in the photothermal spectroscopy experimental setup. In a subsequent stage, and taking advantage of the low dimensionality of the proposed neural network, a mobile application is developed that integrates the identification algorithm. This application is developed in the Kotlin programming language for the Android operating system. This approach not only facilitates the identification of water-alcohol mixtures but also offers an accessible and portable solution.

Keywords: Water-alcohol mixtures, neural networks, alcohol identification, mobile application.

Índice general

Índice de tablas

Índice de figuras

Abreviaturas y siglas

ADALINE: Abreviatura en inglés para el tipo de red neuronal ADAptative LINear Element

APK: Siglas en inglés para Paquete de Aplicación Android (Android Application Package)

CCD: Siglas en inglés para Dispositivo de carga acoplada (charge-coupled device)

MATLAB: Abreviatura en inglés de laboratorio de matrices (Matrix laboratory)

RGB: Sigla del inglés de Rojo, Verde, Azul (Red, Green, Blue)

RNA: Redes neuronales artificiales

SCV: Siglas en inglés para Luz visible súpercontinua (Supercontinuum light)

Capítulo 1

Introducción

1.1. Antecedentes

1.1.1. Mezclas bioquímicas

Las mezclas son composiciones formadas por dos o más sustancias en las que cada una mantiene sus propiedades. En el campo de la bioquímica, que estudia la química de los organismos vivos, se investiga también la química de los alcoholes. Los alcoholes son derivados de los hidrocarburos que contienen uno o más grupos OH, y son capaces de formar mezclas binarias con el agua (Brown *et al.*, 2014). Las mezclas binarias se caracterizan por contener únicamente dos sustancias.

El metanol, etanol y propanol son alcoholes miscibles con agua en cualquier proporción de mezcla. Sin embargo, son difíciles de cuantificar en ambientes acuosos debido a que se encuentran en concentraciones muy bajas, tienen una estructura similar al agua y no absorben mucha luz. Estos alcoholes son introducidos en el medio ambiente a través de diversas fuentes antropogénicas y biogénicas. Las fuentes antropogénicas incluyen los combustibles alternativos, los aditivos de la gasolina, los aerosoles y los decapantes; mientras que las fuentes biogénicas de metanol incluyen la descomposición de residuos, gases volcánicos, vegetación, microbios e insectos (Magolan, 2005; Wakisaka y Matsuura, 2006).

1.1.2. Caracterización

Existen métodos para analizar alcoholes en fase gaseosa en la atmósfera y en fluidos biológicos a altas concentraciones. Sin embargo, estos métodos no son aplicables al agua debido a los altos límites de detección y a las reacciones de los reactivos con los alcoholes y el agua. Una vez que los alcoholes se han oxidado, estos son más sencillos de analizar (Magolan, 2005).

Debido a la pandemia mundial declarada en 2019 por causas del Coronavirus (COVID-19), la demanda de desinfectantes de manos a base de alcohol incrementó drásticamente. Para evaluar y asegurar una concentración de al menos 60 % de alcohol en dichos desinfectantes se realizaron diversos estudios con diferentes métodos de caracterización, algunos de ellos aplicables a la identificación de alcoholes como etanol y propanol en muestras acuosas. (Bedner *et al.*, 2021; Chakraborty *et al.*, 2023).

El Instituto Nacional de Estándares y Tecnología (NIST, National Institute of Standards and Technology) desarrolló y evaluó cuatro métodos de medición para conocer su cantidad de etanol e isopropanol (2-propanol) presentes. Los métodos analizados incluyen cromatografía de gases con detección por ionización de llama (GC-FID, gas chromatography with flame ionization detection), cromatografía líquida con detección por absorción ultravioleta (LC-UV, liquid chromatography with ultraviolet absorbance detection), espectroscopía de resonancia magnética nuclear cuantitativa (qNMR, quantitative nuclear magnetic resonance spectroscopy) y espectroscopía infrarroja por transformada de Fourier con reflectancia total atenuada (ATR-FTIR, attenuated total reflectance Fourier-transform infrared spectroscopy). Los métodos utilizados, excepto qNMR, fueron calibrados utilizando soluciones preparadas a partir de etanol puro de pureza conocida o soluciones de Etanol-Agua (Bedner *et al.*, 2021).

El método GC-FID separa los componentes volátiles de la muestra utilizando una columna cromatográfica y los detecta mediante ionización en una llama de hidrógeno. Este método analiza alcoholes en fase gaseosa y no es aplicable al agua, es decir, no la detecta (Magolan, 2005). El método GC-FID únicamente detecta la concentración de etanol. Este método presenta una alta precisión y sensibilidad, ideal para cuantificar etanol en concentraciones muy bajas. Sin embargo, requiere equipos costosos y una preparación compleja de las muestras . El método LC-UV utiliza una columna cromatográfica para separar los componentes líquidos de la muestra y los detecta por su absorción de luz ultravioleta. Es un método rápido y directo con buena precisión, aunque susceptible a interferencias de otras sustancias que absorben UV (Bedner *et al.*, 2021).

Los espectros de absorción de infrarrojo obtenidos a través de espectrómetría de infrarrojo por transformada de Fourier (FT-IR, Fourier transform infrared transmission) es una técnica que mide la absorción de luz infrarroja que atraviesa una muestra, proporcionando información detallada sobre la composición molecular. Al incorporar la reflectancia total atenuada (ATR, Attenuated total reflection) se obtienen de espectros de absorción sin necesidad de preparación compleja de la muestra. La técnica ATR-FTIR para la caracterización de mezclas líquidas, específicamente de soluciones de agua y etanol, demostró ser una técnica rápida, robusta y no destructiva con las muestras, aunque menos sensible para detectar concentraciones muy bajas de etanol comparado con otras técnicas como GC-FID y qNMR (Bedner *et al.*, 2021; MacDonald y Bureau, 2003).

Otro de los métodos evaluados para conocer la cantidad de alcohol presentes en los desinfectantes fue la espectroscopía de lente térmica de femtosegundos (FTLS, Femtosecond Thermal Lens Spectroscopy). Esta técnica se basa en la detección de cambios térmicos inducidos por pulsos láser ultrarrápidos. Cuando un pulso láser de femtosegundos interactúa con la muestra, genera una distribución de temperatura transitoria que afecta el índice de refracción del medio, creando una "lente térmica". La intensidad y las características de esta lente térmica se utilizan para analizar la composición de una mezcla simple de agua, etanol y alcohol isopropílico. Esta técnica presenta alta sensibilidad y precisión y es no destructiva; sin embargo, requiere de equipos costosos y de un personal altamente capacitado para operar dichos equipos (Chakraborty *et al.*, 2023).

La cuantificación de etanol y agua es importante en procesos de manufactura y control de calidad de muchos productos comerciales como la cerveza, el vino, el licor y el enjuague bucal. Otro ejemplo de cromatografía de gases en conjunto con otras técnicas para identificar etanol y agua es la cromatografía de gases acoplada a la espectrometría de masas (GC-MS, Gas Chromatography-Mass Spectrometry). El método GC-MS se ha empleado para identificar etanol y otros componentes químicos en la cerveza casera. Este método demostró ser simple, rápido y económico pero requiere de un solvente para realizar el análisis, lo cual implica alterar la muestra original (Tsenang *et al.*, 2023). Otra variación aplicada a productos comerciales es la cromatografía de gases capilar con líquidos iónicos (ILCG, Ionic Liquid Capillary Gas Chromatography) con detección de conductividad térmica (TCD, Thermal Conductivity Detection) y/o detección de ionización por descarga de barrera (BID, Barrier Discharge Ionization Detection). El método TCD mide los cambios en la conductividad térmica del gas portador al pasar por la columna, permitiendo la cuantificación de los componentes y el método BID ioniza los compuestos separados y mide la corriente resultante, ofreciendo alta sensibilidad para la detección de etanol y agua. La combinación de estos métodos con el ILCG permitió una rápida separación y detección del etanol y el agua en varias muestras de los productos comerciales mencionados. Sin embargo, estos análisis requieren de equipos costosos . (Weatherly *et al.*, 2014)

Otro método que utiliza espectroscopia fototérmica para caracterizar propiedades generales de las mezclas de alcohol-agua, es el análisis de patrones de difracción generados por la interacción de luz-muestra cuando se produce un gradiente de temperatura en la mezcla. La variación en el índice de refracción generado por el gradiente de temperatura sobre el líquido genera patrones de difracción bien definidos que se pueden analizar para extraer información sobre las propiedades del líquido. Para analizar los patrones, se utilizó un procesamiento de imágenes estándar, que consiste en la conversión de imágenes en color a escala de grises y una transformada de Fourier espacial. Este método es no invasivo y se puede aplicar a una amplia variedad de líquidos; sin embargo, puede requerir de un modelo matemático detallado que relacione los patrones de difracción generados con las propiedades de la muestra. (Domínguez-Juárez *et al.*, 2023b).

La literatura muestra que la identificación, o análisis cualitativo, de mezclas acuosas que contienen alcohol a menudo se logra mediante mediciones ópticas. Aparte de los métodos ya mencionados, otras técnicas para caracterizar las mezclas binarias de alcohol y agua incluyen la espectrometría de masas, la difracción de neutrones, la difracción de rayos X, y la emisión de rayos X (Wakisaka y Matsuura, 2006).

1.2. Descripción del problema

Los alcoholes son compuestos orgánicos que contienen uno o más grupos hidroxilos (-OH), particularmente el metanol, etanol y propanol son alcoholes que presentan una estructura molecular similar con el agua y tienen una baja absorción de luz (Brown *et al.*, 2014). Sin embargo, estos son miscibles en agua en cualquier proporción de mezcla.

La presencia y cuantificación de alcoholes en agua es un aspecto crucial en diversos procesos de manufactura y control de calidad de productos comerciales como la cerveza, el vino, el licor, gel antibacterial y el enjuague bucal. La precisión en la medición de la concentración de alcoholes como el etanol y el propanol es esencial para garantizar la calidad y la seguridad de estos productos.

Adicionalmente, con el advenimiento de la pandemia mundial declarada en 2019 debido al Coronavirus (COVID-19), la demanda de desinfectantes de manos a base de alcohol experimentó un aumento significativo. Estos desinfectantes deben contener al menos un 60 % de alcohol para ser efectivos contra el virus, lo que indica la necesidad de métodos precisos y fiables para medir la concentración de alcohol en soluciones acuosas.

Otro sector donde se requiere la medición precisa de materiales líquidos para controlar las fotorreacciones es en la protonación en fluidos estimulados por luz o la recolección de energía a través de captadores complejos de luz. En estos procesos se busca investigar las propiedades químicas y fotoquímicas de los productos, e incluso detectar la generación de combustibles verdes. En los procesos químicos relacionados con la producción de energías renovables en soluciones acuosas, los alcoholes juegan un papel fundamental.

Existen métodos para conocer cuantitativamente proporciones de alcoholes como metanol, etanol y propanol, sin embargo, debido a las características que presentan estos alcoholes existen algunos retos que mantienen vigente la investigación y desarrollo de estrategias para realizar esta cuantificación. Por ejemplo, los métodos para analizar alcoholes en fase gaseosa en la atmósfera y en fluidos biológicos a altas concentraciones no son aplicables al análisis de mezclas acuosas debido a la baja concentración de los alcoholes en el agua. Otra desventaja de estos mé-

todos son las reacciones de los reactivos con los alcoholes y el agua, las cuales son invasivas con la muestra y limitan su recuperación y posterior uso (Magolan, 2005).

Otros métodos para caracterizar las mezclas binarias de alcohol y agua incluyen espectrometría de masas, difracción de neutrones, difracción de rayos X y emisión de rayos X (Wakisaka y Matsuura, 2006). Estos procedimientos suelen necesitar equipos e instalaciones especializados y costosos, lo que puede restringir su disponibilidad y uso en determinados entornos o circunstancias.

1.3. Justificación

La interacción de luz coherente (con propiedades conocidas) con líquidos polares o no polares ha abierto la puerta para investigar sus propiedades térmicas lineales y no lineales, así como efectos estacionarios y transitorios en el tiempo. Debido a la absorción óptica, un líquido libera calor local que genera un cambio no homogéneo en el índice de refracción en las proximidades del haz óptico de excitación (Callen *et al.*, 1967). Esta alteración del índice de refracción actúa como una lente negativa (efecto de lente térmica) en el líquido. Las características de la lente térmica se pueden analizar para obtener una medición directa de la energía óptica absorbida por un líquido y sus propiedades ópticas y termofísicas (Proskurnin *et al.*, 2015, 2022). Por estas razones, el efecto de la lente térmica (también conocido como espectrometría de lente térmica o espectrometría fototérmica) tiene múltiples aplicaciones prácticas, que van desde la caracterización de materiales (Shimizu *et al.*, 2022), pasando por aplicaciones en biomedicina (Pleitez *et al.*, 2015; Hertzberg *et al.*, 2017) y química analítica (Kitamori, 2019), hasta microscopia (Adhikari *et al.*, 2020) y fluídica (Chen *et al.*, 2021).

Los efectos de las lentes térmicas se pueden analizar de diferentes maneras. Una alternativa no invasiva es analizar el haz en el campo lejano. Debido a las propiedades de coherencia del haz incidente se producen patrones de interferencia, que están asociadas con la auto-modulación de fase del perfil del campo eléctrico en un plano perpendicular al haz de excitación (Gordon *et al.*, 1965; Dabby *et al.*, 1970; Greenfield *et al.*, 2011; Dominguez-Juarez *et al.*, 2015). Para generar los patrones de interferencia, se modula el índice de refracción de la muestra mediante un gradiente de temperatura inducido por la fuente de luz láser, aprovechando los fenómenos de interacción entre la luz coherente y la mezcla bioquímica binaria. Estos patrones de interferencia pueden correlacionarse con las propiedades ópticas de la muestra, proporcionando información valiosa sobre sus características intrínsecas. No obstante, la tarea de analizar esta información presenta desafíos significativos cuando se utilizan métodos convencionales, como la inspección visual o las técnicas estándar de procesamiento de imágenes, debido a la complejidad de los patrones generados.

En la actualidad, los algoritmos de inteligencia artificial, especialmente las redes neuronales artificiales, han demostrado resultados sobresalientes en la resolución de problemas de reconocimiento de patrones. La alta capacidad de estas redes para abordar problemas complejos ha permitido identificar características clave que facilitan la asociación entre vectores de entrada y categorías objetivo (Abiodun *et al.*, 2019). Por esta razón, las redes neuronales artificiales se presentan como una herramienta ideal para correlacionar patrones de interferencia con mezclas bioquímicas de alcohol y agua.

En esta tesis, se diseñan e implementan redes neuronales artificiales con el fin de identificar mezclas bioquímicas binarias a diferentes concentraciones utilizando patrones de difracción generados mediante espectroscopía fototérmica. La red neuronal propuesta es una red de propagación hacia adelante de dos capas, con neuronas de activación sigmoide en la capa oculta y neuronas softmax en la capa de salida (Kim y Park, 2018). Aunque existen librerías especializadas en reconocimiento de patrones y clasificación de imágenes (Kim, 2017), estas suelen ser complejas de modificar y adaptar a necesidades específicas. Para superar este obstáculo, todas las redes neuronales desarrolladas en esta tesis fueron programadas utilizando algoritmos propios.

La baja dimensionalidad de la arquitectura de la red neuronal propuesta facilitó su implementación en un dispositivo móvil. Para ello, se diseñó una aplicación en lenguaje de programación Kotlin para dispositivos con sistema operativo Android. Esta aplicación permite identificar mezclas binarias a través de imágenes capturadas por la cámara del dispositivo o mediante la importación de imágenes previamente guardadas en la memoria del mismo. Es importante destacar que, en este trabajo, se evitó el uso de redes neuronales con un gran número de capas ocultas y neuronas, debido al alto consumo de recursos computacionales que estas arquitecturas requieren durante su entrenamiento (Abiodun *et al.*, 2019).

1.4. Hipótesis

Es posible desarrollar una red neuronal artificial de una sola capa oculta, e implementarla en una aplicación, que habilite la identificación de muestras bioquímicas binarias de agua-alcohol con un 90 % de eficiencia en el reconocimiento usando como entrada los patrones de difracción adquiridos por la cámara del dispositivo.

1.5. Objetivos

1.5.1. Objetivo general

Desarrollar una red neuronal de una sola capa oculta e implementarla en una aplicación que permita identificar mezclas bioquímicas binarias de agua-alcohol a partir de los patrones difracción generados en el campo lejano.

1.5.2. Objetivos específicos

1. Construir una base de datos que integre imágenes de los patrones de difracción en el régimen de campo lejano adquiridos por una cámara CCD para diferentes concentraciones de mezclas bioquímicas binarias de agua-alcohol considerando los alcoholes: etanol, metano, 1-propanol y 2-propanol.

2. Preprocesar el conjunto de observaciones que conforma la base de datos con el fin de reducir la dimensionalidad del espacio de características.

3. Desarrollar un modelo computacional en MATLAB para el entrenamiento de una red neuronal artificial de una sola capa oculta de neuronas sigmoide y neuronas softmax en la capa de salida, que implemente el método del descenso del gradiente para minimizar la entropía cruzada.

4. Entrenar y evaluar una red neuronal artificial usando las imágenes de la base de datos creada para identificar cinco tipos de mezclas bioquímicas: agua, metanol, etanol, 1-propanol y 2-propanol.

5. Construir una base de datos que integre imágenes de los patrones de difracción en el régimen de campo lejano adquiridos por una cámara de celular para diferentes concentraciones de mezclas bioquímicas binarias de agua-alcohol considerando los alcoholes: etanol, metano y 1-propanol.

6. Entrenar y evaluar una red neuronal artificial en MATLAB usando el conjunto de imágenes adquiridas con la cámara de celular para identificar cuatro tipos de mezclas bioquímicas: agua, metanol, etanol y 1-propanol.

7. Desarrollar un modelo computacional de una red neuronal artificial de una sola capa oculta de neuronas sigmoide y neuronas softmax en la capa de salida en lenguaje Kotlin para sistema operativo Android que implemente los pesos sinápticos optimizados de la red neuronal entrenada en el punto anterior.

1.5.3. Organización de la tesis

Esta tesis está organizada de la siguiente forma: En el Capítulo 2 se presenta la fundamentación teórica, donde se abordan los conceptos esenciales que sustentan el desarrollo de este trabajo. Esto incluye una revisión de las redes neuronales artificiales y el fenómeno de difracción. En el Capítulo 3 se describe la metodología empleada, detallando el diseño y la implementación de la red neuronal propuesta, así como los procedimientos experimentales utilizados para generar y capturar los patrones de difracción. Además, se explica el proceso de desarrollo de la aplicación móvil, desde la programación en Kotlin hasta la integración con la red neuronal. El Capítulo 4 presenta los resultados obtenidos, donde se analizan tanto la precisión de la red neuronal en la identificación de mezclas binarias como el desempeño de la aplicación móvil en distintas condiciones experimentales. Finalmente, en el Capítulo 5 se discuten las conclusiones del trabajo, destacando las contribuciones más relevantes de la investigación, las limitaciones encontradas y las posibles direcciones para futuros estudios.

Capítulo 2

Fundamentación teórica

2.1. Difracción

2.1.1. Interferencia

La luz se origina a partir de las transiciones de electrones que se producen dentro de átomos excitados debido a una temperatura elevada. Cada átomo excitado emite un tren de ondas duran te un periodo de tiempo. Cuando estos trenes de ondas se superponen en el espacio, se produce interferencia. Es decir, la interferencia óptica corresponde a la interacción, o suma algebraica en cada punto de la región de solapamiento, de dos o más ondas de luz (Hammond, 2015; Hecht, 2016).

Los efectos de la interferencia son más evidentes cuando las ondas que se superponen son ondas sinusoidales coherentes, es decir, que tienen la misma longitud de onda (monocromáticas), la misma frecuencia y una relación de fase constante. Si dos o más ondas llegan a un punto en fase, se produce una interferencia constructiva, donde la amplitud de la onda resultante es la suma de las amplitudes de las ondas originales. Cuando las ondas llegan desfasadas por media longitud de onda existe interferencia destructiva, es decir, la amplitud de la onda resultante es cero (Serway y Jewett, 2009).

El principio de Huygens establece que cada punto de un frente de onda funciona como una fuente emisora de ondas secundarias, las cuales se propagan en todas direcciones a la misma velocidad que la onda original. La interferencia y la difracción resultan de la superposición de estas ondas y están relacionadas con el principio de Huygens (Serway y Jewett, 2009).

2.1.2. Difracción

La difracción es una propiedad común de los fenómenos ondulatorios que ocurre cuando una porción de un frente de onda, ya sea de sonido, luz o ondas de materia, se encuentra con un

obstáculo. La difracción de la luz ocurre cuando una fracción de la luz emitida por una fuente se separa debido a la presencia de un cuerpo opaco y esto es lo que da origen a su nombre: división en fracciones. El patrón de difracción consiste en una serie de regiones claras y oscuras; correspondientes a las condiciones de interferencia constructiva y destructiva, respectivamente (Hammond, 2015).

La teoría de Rayleigh-Sommerfeld describe la luz como un fenómeno escalar, rechazando la naturaleza vectorial del campo electromagnético. La teoría del campo escalar es muy precisa para describir los fenómenos cuando se cumplen dos condiciones: 1) las aberturas son más grandes que la longitud de onda de la luz y 2) las distancias de observación son suficientemente grandes. Con estas condiciones, en un medio dieléctrico lineal, isentrópico, homogéneo y no dispersivo, todos los componentes del campo eléctrico y magnético se comportan de forma idéntica y su comportamiento se puede describir mediante una ecuación de onda escalar. En estas circunstancias, la polarización del campo electromagnético no es significativa, por lo que no es necesario utilizar el formalismo vectorial. La teoría escalar es adecuada para describir la difracción en estructuras que son mayores que la longitud de onda de la luz (Goodman, 2005).

Aplicando algunas aproximaciones a la teoría de difracción escalar se obtiene la aproximación de Fresnel y Fraunhofer. La difracción de Fresnel es la difracción de campo cercano y la difracción de Fraunhofer es difracción de campo lejano. La difracción de Fraunhofer ocurre cuando la fuente, el obstáculo y la pantalla están lo suficientemente distantes, permitiendo que todas las líneas desde la fuente al obstáculo y desde el obstáculo a la pantalla se consideren paralelas.(Goodman, 2005; Serway y Jewett, 2009).

2.2. Redes neuronales

El *machine learning*, o aprendizaje automático, es una subdisciplina de la inteligencia artificial que permite a las máquinas aprender, lo que implica que pueden generalizar conocimiento a partir de experiencias en lugar de ser programadas de manera explícita. El *machine learning* permite reconocer patrones que se encuentran enterrados en grandes conjuntos de datos y después de una etapa de entrenamiento, habilitan la idenficación de conjuntos de datos no antes vistos. Por su parte, el *deep learning*, o aprendizaje profundo, es una subcategoría del *machine learning* que permite que los sistemas no solo aprendan de la experiencia, sino que también habilita la extracción de características de forma automática (BBVA, 2019; LeCun *et al.*, 2015; Goodfellow *et al.*, 2016).

2.2.1. Definición

Las redes neuronales artificiales (RNA) son modelos computacionales reduccionistas inspirados en las neuronas biológicas. Estas redes constituyen una técnica de aprendizaje automático (machine learning) que interconecta nodos, denominados neuronas, para formar estructuras organizadas en capas que emulan la red de neuronas del cerebro humano. El funcionamiento de las RNA se basa en el entrenamiento con datos, lo que les permite aprender y mejorar su precisión con el tiempo. Una vez entrenados, estos algoritmos se convierten en poderosas herramientas capaces de resolver problemas complejos y realizar diversas tareas de manera eficiente. Las RNA son especialmente útiles en la clasificación de datos, donde pueden categorizar elementos en diferentes grupos basándose en características aprendidas durante el proceso de entrenamiento. Además, son efectivas en la segmentación de imágenes y datos, dividiendo la información en partes significativas para su análisis detallado. Estas redes también tienen la capacidad de generar nuevos datos y predecir resultados, utilizando patrones identificados previamente. (IBM, 2020).

Las aplicaciones de las redes neuronales artificiales son diversas y abarcan múltiples campos, tales como las finanzas, la medicina, la física, la ingeniería, la robótica y el transporte. En el ámbito financiero, se utilizan para predecir tendencias del mercado y gestionar riesgos (Fadlalla y Lin, 2001; McNelis, 2005). En medicina, ayudan en el diagnóstico de enfermedades y en la interpretación de imágenes médicas (Baxt, 1995; Penny y Frost, 1996; Mall *et al.*, 2023). En física e ingeniería, contribuyen a la simulación y optimización de sistemas complejos (Escobar-Ruiz *et al.*, 2024; Cuomo *et al.*, 2022; Cai *et al.*, 2021). En robótica, mejoran la toma de decisiones y la navegación autónoma (Chhikara *et al.*, 2020). En el transporte, optimizan rutas y gestionan el tráfico (Jiang y Luo, 2022; Tedjopurnomo *et al.*, 2020).

Una red neuronal se caracteriza por los siguientes elementos (Ponce, 2010):

- Un conjunto de unidades de procesamiento o neuronas.
- Un estado de activación para cada unidad.
- Conexiones entre las unidades definidas por un peso que determina el efecto de una señal de entrada en la unidad.
- Una regla de propagación que determina la entrada efectiva de una unidad a partir de las entradas externas.
- Una entrada externa o *bias* para cada unidad.
- Un método para reunir la información, conocido como regla del aprendizaje .

2.2.2. Arquitectura

Al igual que en el cerebro humano, la unidad básica de una red neuronal artificial es la neurona, también conocida como unidad de procesamiento. Las neuronas se organizan en capas: una capa de entrada, que contiene neuronas que representan los campos de entrada; una o varias capas ocultas; y una capa de salida, que incluye una o más neuronas que representan el campo o los campos de destino. Con el objetivo de imitar la sinapsis biológica entre neuronas, las neuronas artificiales se conectan mediante ponderaciones, denominadas pesos sinápticos. Generalmente, los datos de entrada se introducen en la primera capa, y los valores se transmiten desde cada neurona de una capa hacia cada neurona de la capa siguiente. Este tipo de conexión y flujo de información se denomina propagación hacia adelante. Al final del proceso, se envía un resultado hacia la capa de salida (IBM, 2021). Esta arquitectura se ilustra en la figura 2.1.

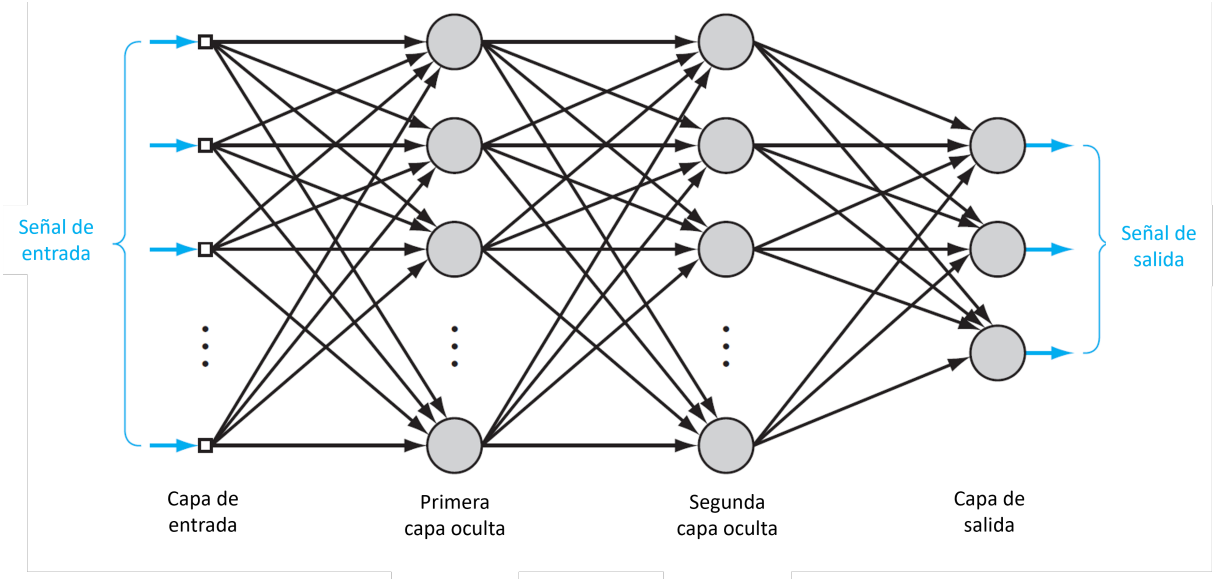


Figura 2.1: Esquema básico de una red neuronal multicapa (Haykin, 2009).

Sin embargo, la arquitectura de propagación hacia adelante no es la única existente en el ámbito de las redes neuronales artificiales. Existen otras arquitecturas que se han desarrollado para abordar diferentes tipos de problemas y mejorar la eficiencia y precisión del aprendizaje. Algunas de las arquitecturas más destacadas incluyen: redes neuronales recurrentes, redes neuronales siamesas, redes neuronales de corto alcance y redes neuronales adversarias (Nielsen, 2015; LeCun *et al.*, 2015).

2.2.3. Neurona

Cada neurona se puede definir como un modelo de regresión lineal, compuesto por datos de entrada–también conocidas como características–, pesos sinápticos, un sesgo (o umbral) y una salida. La salida de cada neurona está dada por la suma ponderada z (Ponce, 2010) siguiente:

$$z = \sum_{i=1}^{n} w_i x_i + b \tag{2.1}$$

donde n es el número de entradas o características, w_i es el peso sináptico (o importancia) asociado a i-ésima característica, x_i son las entradas y b es el *bias* o sesgo. Este último representa el error esperado del modelo debido a factores que desconocemos. Finalmente, a la entrada z, o regla de propagación se le aplica una función no lineal–en su forma más general un operador no lineal– a, llamada función de activación, tal que la salida y de la neurona está dada por (Ponce, 2010):

$$y = a(z) = a\left(\sum_{i=1}^{n} w_i x_i + b\right) \tag{2.2}$$

La ecuación 2.1 se puede reescribir de forma matricial como:

$$z = \mathbf{w}^T \mathbf{x} + b \tag{2.3}$$

El término matricial $\mathbf{w}^T\mathbf{x}$ es el producto interno de los vectores $\mathbf{w}$ y $\mathbf{x}$ donde (Haykin, 2009) :

$$\mathbf{w}^T = \begin{bmatrix} w_1 & w_2 & \dots & w_n \end{bmatrix} \qquad \mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ \dots \\ x_n \end{bmatrix}$$

En la figura 2.2 se presenta el esquema de una neurona.

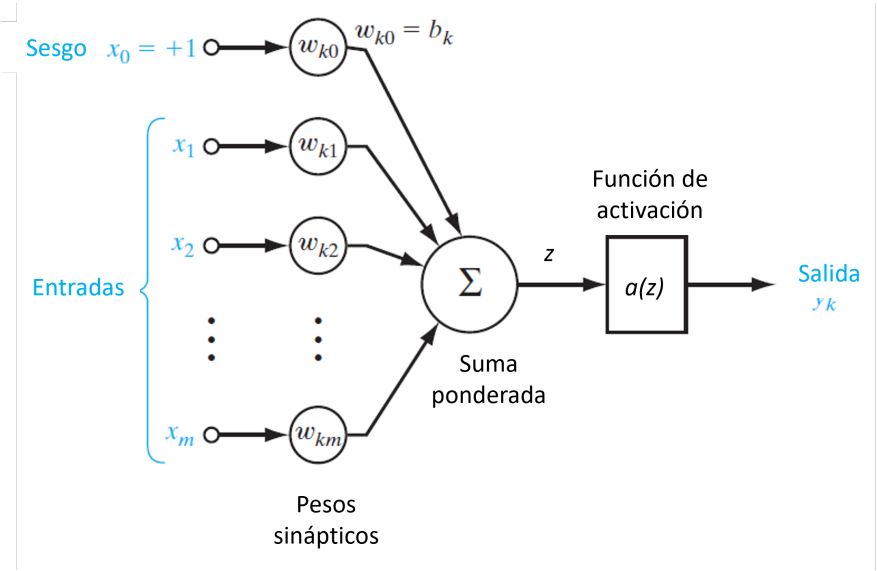


Figura 2.2: Diagrama de una neurona con multiples entradas (Hagan *et al.*, 2014).

2.2.4. Función de activación

La función de activación es una función $f(x)$ tal que $f : \mathbb{R} \to \mathbb{R}$, que se aplica a la regla de propagación z, y permite que la neurona pueda aprender patrones complejos. Esta función introduce no linealidad en el modelo, lo cual es indispensable para que la red neuronal pueda capturar y representar relaciones complejas en los datos. Sin la función de activación, la capacidad de la red se reduciría a aprender únicamente funciones lineales, limitando severamente su aplicabilidad y efectividad.

Existen diversos tipos de funciones de activación utilizadas en el aprendizaje autónomo (Calvo, 2022). La selección de la función de activación varía según el tipo de problema y la arquitectura de la red neuronal. Elegir la función adecuada puede mejorar notablemente el rendimiento y la capacidad de generalización del modelo. Además, la función de activación debe ser diferenciable para permitir la propagación hacia atrás del error durante el entrenamiento de la red. Este requisito es crucial para la implementación de algoritmos de optimización como lo es el descenso del gradiente, que ajustan los pesos sinápticos con el objetivo de minimizar la función de costo de la red neuronal.

Lineal

La salida de una función de activación lineal es igual a la entrada (figura 2.3):

$$a(z) = z \tag{2.4}$$

Este tipo de función se emplea para las redes neuronales ADALINE (Hagan *et al.*, 2014).

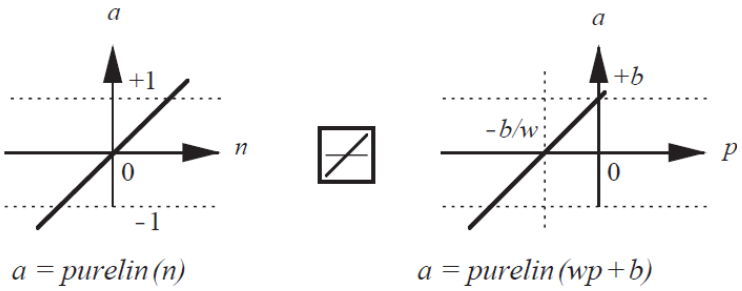


Figura 2.3: Función de activación lineal (Hagan *et al.*, 2014).

Hard Limit

Esta función se utiliza para clasificación binaria. La gráfica de la función se muestra en la figura 2.4. En las neuronas con esta función, cuando la suma de las entradas es mayor o igual que el umbral de la neurona, la activación es 1; si es menor, la activación es 0 (Hagan *et al.*,

2014):

$$a(z) = \begin{cases} 1 & z \geq 0 \\ 0 & z < 0 \end{cases} \tag{2.5}$$

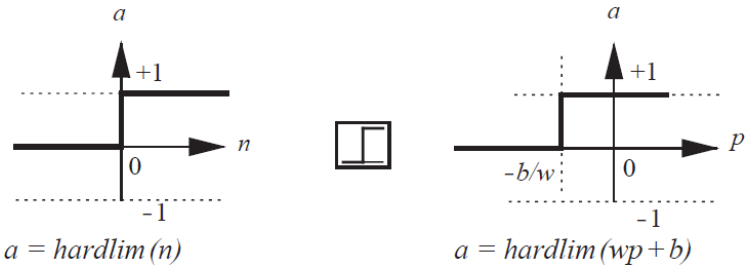


Figura 2.4: Función de activación escalón o hardlim (Hagan *et al.*, 2014)

Sigmoide

La función sigmoide es comúnmente utilizada en las capas ocultas de redes neuronales multicapas entrenadas con el algoritmo de *backpropagation*. También se le conoce como función logística y su rango de valores de salida va de cero a uno, lo que permite interpretar la salida como una probabilidad (figura 2.5). Está definida como (Hagan *et al.*, 2014):

$$a(z) = \frac{1}{1 + e^{-z}} \tag{2.6}$$

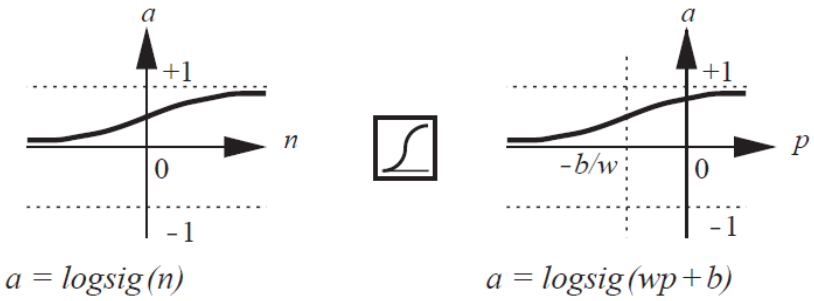


Figura 2.5: Función de activación sigmoide (Hagan *et al.*, 2014).

Softmax

La función *softmax* es la más utilizada en la capa de salida para problemas de clasificación multiclase. Esta función produce una distribución de probabilidad sobre las clases a predecir. Dichas clases son excluyentes, es decir, si pertenece a una clase, ya no se puede asignar a otra. Esta función normaliza los valores de salida, es decir, cada salida toma un valor entre 0 y 1 y la suma todas ellas dan 1. Esta función esta definida como (Hagan *et al.*, 2014):

$$a(Z)_i = \frac{e^{z_i}}{\sum_{j=1}^{k} e^{z_j}} \tag{2.7}$$

donde Z es el vector de entrada de la última capa; los subíndices i, j denotan la neurona de la capa; y k es el número total de clases o neuronas en la última capa (Hagan *et al.*, 2014).

2.2.5. Tipos de redes

Perceptrón

El Perceptron fue el primer modelo de RNA presentado a la comunidad científica por el psicólogo Frank Rosenblatt en 1958 (Rosenblatt, 1961). Este es el tipo de red neuronal más simple que existe debido a que esta compuesto por una única capa de entrada y un nodo de salida. Su estructura se puede definir mediante la ecuación 2.2. Los modelos que clasifica un perceptrón son problemas binarios, por lo que se emplea la función *hardlim* como función de activación (figura 2.6). Si, al comparar, la suma del producto de las entradas por los pesos sinápticos es mayor o igual al patrón establecido, la salida de la red es uno; de lo contrario, la salida es cero (Ponce, 2010).

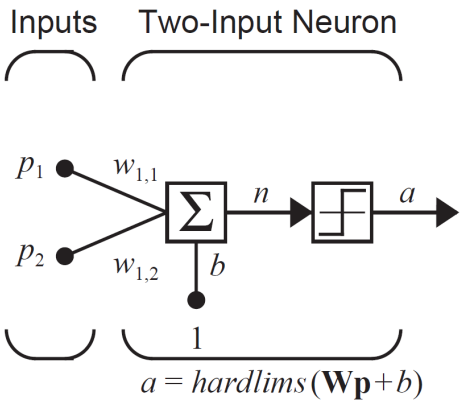


Figura 2.6: Diagrama de la red neuronal perceptrón(Hagan *et al.*, 2014).

La aplicación de este tipo de red es limitada debido a su incapacidad de ejecutar ciertas funciones elementales como la resolución de problemas no lineales, por ejemplo la compuerta lógica XOR, o la clasificación de más de dos categorías. Para resolver dichas tareas es necesario hacer uso de más neuronas. A pesar de sus limitaciones el perceptrón es importante por razones históricas y también es una herramienta útil para entender conceptos clave de las redes neuronales (Hagan *et al.*, 2014).

ADALINE

De forma casi simultánea al desarrollo del Perceptron, Bernard Widrow y su estudiante Marcian Hoff presentaron su red neuronal ADALINE (*Adaptive Linear Neuron*), la cual tiene una topología similar a la del perceptrón (figura 2.6). A diferencia del perceptón, la red ADALINE emplea

una función de activación lineal y puede tener más de una neurona en su capa de procesamiento; sin embargo, ambos tipos de redes presentan la limitación de únicamente resolver problemas lineales. A pesar de esto, la red ADALINE fue un gran adelanto en el desarrollo de las redes neuronales debido a que utiliza el algoritmo del gradiente descendente para la deducción de la regla de entrenamiento (Hagan *et al.*, 2014; Bravo y Sotelo, 2009).

Perceptrón multicapa

Un perceptrón consta solo de una capa de entrada y una de salida. En cambio, en una red multicapa o perceptrón multicapa, hay varias capas intermedias, conocidas como capas ocultas, entre la capa de entrada y la de salida. Estas capas ocultas son fundamentales para que la red aprenda representaciones complejas y modele relaciones no lineales en los datos. La inclusión de múltiples capas ocultas permite a la red crear jerarquías de características, donde las capas inferiores aprenden características simples y las superiores combinan esas características para generar representaciones más abstractas y complejas. Una red con una sola capa oculta se denomina red neuronal *shallow* o vanilla, mientras que aquellas con más de dos capas ocultas se conocen como redes neuronales profundas (Kim, 2017; Aggarwal, 2018).

Para cada capa, el vector de salida de las neuronas está dada por la siguiente ecuación:

$$\mathbf{Z} = \mathbf{W}\mathbf{X} + \mathbf{b} \tag{2.8}$$

donde $\mathbf{X}$ es el vector de entrada, $\mathbf{b}$ es el vector que contiene los *bias* de cada neurona; ambos de dimensiones $n \times 1$, donde n es el número de entradas. $\mathbf{W}$ es la matriz que contiene los pesos asociados a cada neurona de dimensiones $m \times n$, donde m es el número de nodos de la capa actual (Kim, 2017). La arquitectura específica de las redes neuronales multicapa se denomina redes *feedforward* debido a que las capas sucesivas se alimentan unas a otras en la dirección hacia adelante (*fordward*), de la entrada a la salida. En este tipo de redes todos los nodos de una capa están conectados a los de la capa siguiente como se observa en la figura 2.1 (Aggarwal, 2018).

2.2.6. Función de costo

Para el entrenamiento de la red neuronal, buscamos un algoritmo que permita encontrar los pesos y sesgos de manera que la red se aproxime a la clase objetivo $y(x)$ para todas las entradas x. Para cuantificar la proximidad a este objetivo, definimos la función de costo (Nielsen, 2015). Una función de costo es una función de variable real que mide el error entre el valor real y el valor predicho por la red neuronal, con el objetivo de optimizar los parámetros de la red (Santillana Quesada, 2022). Este error se cuantifica de diversas formas, dependiendo del tipo de problema. En problemas de regresión, una función de costo común es el error cuadrático medio (MSE, por sus siglas en inglés) (Hodson, 2022), mientras que en problemas de clasificación se utilizan funciones como la entropía cruzada (Mao *et al.*, 2023).

El proceso de optimización de los parámetros de la red neuronal se lleva a cabo mediante algoritmos de optimización, siendo el más común el descenso del gradiente. Este algoritmo ajusta los pesos y sesgos de la red en la dirección que minimiza la función de costo. El ajuste se realiza iterativamente, calculando el gradiente de la función de costo con respecto a los pesos y sesgos, y actualizándolos en cada paso para reducir el error.

Error cuadrático medio

Para un patrón de entrada x con una salida estimada a, y una salida deseada $y(x)$, buscamos minimizar la distancia $|y(x) - a|$. Para ello minimizamos el error cuadrático medio, definido por (Nielsen, 2015):

$$C(w, b) = \frac{1}{n} \sum_{x} (y(x) - a)^2 \tag{2.9}$$

donde w representa los pesos en la red, b son los sesgos o *biases*, n es el número total de entradas de entrenamiento, a es el vector de las salida la red neuronal y x son las entradas. Esta función también es llamada función de costo cuadrática y disminuye conforme la salida a se aproxima al objetivo $y(x)$ (Nielsen, 2015).

Entropía cruzada

La entropía de una variable aleatoria X es el nivel de incertidumbre inherente al posible resultado de la variable. Para una variable discreta con distribución de probabilidad $p(x)$, la entropía está definida como (Koech, 2020):

$$H(x) = -\sum_{x} p(x) \log p(x) \tag{2.10}$$

El signo negativo de esta ecuación se debe a que $p(x)$, al ser una distribución de probabilidad, debe estar dentro del intervalo $(0, 1)$. En la figura 2.7, notamos que $\log(p(x)) < 0 \,\forall\, p(x) \in (0, 1)$, por lo que cumple con este requisito. De la ecuación 2.10 notamos que cuanto mayor sea el valor de la entropía, $H(x)$, mayor es la incertidumbre de la distribución de probabilidad y cuanto menor sea el valor, menor será la incertidumbre. Esta característica de la entropía permite calcular la probabilidad en función de lo alejado que esté el valor calculado del valor esperado (Koech, 2020).

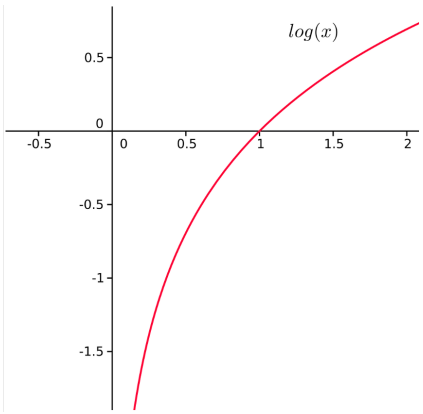


Figura 2.7: Gráfica de la función $\log(x)$ (Koech, 2020)

Sea y un vector *one-hot*, es decir una representación binaria para codificar categorías discretas –por ejemplo a la clase que pertenece una imagen– tal que $y_1, ...y_k \in \{0, 1\}$. La función de costo de entropía cruzada para problemas de clasificación se define como (Aggarwal, 2018):

$$C = -\frac{1}{n}\sum_{i}^{k} y_i \ln \hat{y}_i^L \tag{2.11}$$

donde k es el número de clases, n es el número de instancias u observaciones y L es la última capa de la red neuronal. Al ser una función de costo, $\hat{y}_1, ..., \hat{y}_k$ son las probabilidades obtenidas de las k clases, es decir, las $a_1, ...a_k$ salidas de la función de activación de la última capa. En general, la función de costo de entropía cruzada es más fácil de optimizar que el error cuadrático medio Aggarwal (2018).

Para problemas de clasificación multiclase la función de costo Entropía cruzada está definida como (Nielsen, 2015):

$$C = -\frac{1}{n}\sum_{i}^{k} \left[y_i \ln a(Z^L) + (1 - y_i) \ln\big(1 - \ln a(Z^L)\big)\right] \tag{2.12}$$

2.2.7. Entrenamiento

En el ámbito del *Machine Learning*, existen tres tipos de aprendizaje: supervisado, no supervisado y reforzado. En el aprendizaje supervisado, cada conjunto de datos de entrenamiento debe estar etiquetado, lo que significa que cada entrada está vinculada a una salida o categoría específica. Este método permite que el modelo aprenda a relacionar entradas con salidas correctas utilizando ejemplos etiquetados. El proceso de aprendizaje en redes neuronales con un enfoque supervisado implica múltiples revisiones y ajustes de los pesos W para disminuir la discrepancia entre la salida correcta (establecida por los datos de entrenamiento) y la salida producida por

la red (Kim, 2017). Esta discrepancia se mide a través de la función de costo, y el objetivo del entrenamiento es minimizar dicha función.

En contraste, el aprendizaje no supervisado no utiliza etiquetas para los datos de entrenamiento. En su lugar, el modelo intenta encontrar patrones o estructuras inherentes en los datos. Ejemplos de algoritmos no supervisados incluyen el análisis de componentes principales (PCA) y los algoritmos de clustering como el k-means (Barlow, 1989).

Finalmente, el aprendizaje por refuerzo es una forma de aprendizaje en la que un agente aprende a tomar decisiones al interactuar con un entorno. El agente recibe recompensas o castigos en función de sus acciones y adapta su estrategia para maximizar la recompensa total a lo largo del tiempo. Este tipo de aprendizaje es común en aplicaciones como el control robótico y los juegos (Sutton y Barto, 2018).

Regla delta

La regla delta es la regla de aprendizaje representativa de la red neuronal de una sola capa. Aunque no es capaz de entrenar redes multicapa, es muy útil para estudiar conceptos importantes de la regla de aprendizaje de la red neuronal. Esta regla, también conocida como la regla de aprendizaje del perceptrón, se basa en el principio de ajustar los pesos de las conexiones neuronales para minimizar el error entre la salida deseada y la salida real de la red. La regla delta también introduce conceptos esenciales como el gradiente descendente, que es un método de optimización utilizado para minimizar funciones de costo en una variedad de algoritmos de aprendizaje automático (Russell y Norvig, 2016).

Descenso del gradiente

Dado que el objetivo es que la salida de la red se acerque lo más posible a la salida deseada, el aprendizaje de la red se plantea como un problema de minimización de la función de costo. El método de optimización (minimización) más habitualmente empleado es el denominado descenso por el gradiente. Si calculamos la variación de la función de costo C, el sentido de la máxima variación (máximo gradiente) apuntará hacia una colina en la hipersuperficie de C. Modificamos los pesos W en sentido contrario al indicado por el gradiente de la función de costo, de esta forma se desciende por la superficie del error hasta encontrar un mínimo. Para encontrar la configuración de pesos óptima mediante este algoritmo, para cada iteración o época (j) se utiliza la siguiente fórmula ((Martín del Brío y Sans Molina, 2006):

$$\Delta W = -\alpha \nabla C \tag{2.13}$$

donde α es la taza de aprendizaje o tamaño del paso tomado en cada iteración. Lo anterior se puede reescribir como:

$$W(j+1) - W(j) = -\alpha \frac{\partial C}{\partial W(j)} \tag{2.14}$$

Backpropagation

La retropropagación del error (*backpropagation*) es un tipo de entrenamiento supervisado que se utiliza en redes multicapas. Fue introducido en los años 70s pero su uso cobró relevancia hasta 1986 en el artículo de David Rumelhart, Geoffrey Hinton y Ronald Williams (Rumelhart *et al.*, 1986). Este artículo describe varias redes neuronales donde el algoritmo *backpropagation* hace posible un aprendizaje más rápido que los métodos empleados anteriormente (Nielsen, 2015).

Este algoritmo es empleado para calcular las derivadas parciales de la función de costo con respecto a cada uno de los parámetros de la red, es decir, el gradiente. Para aplicar este algoritmo en una iteración o época se realizan los siguientes cálculos (Nielsen, 2015):

1. Se establece la función de activación a^l para la capa l-ésima de la red neuronal.

2. Se realiza el feedfordward utilizando la ecuación 2.2, es decir, para cada $l = 2, 3, ..., L$ capas se obtiene la salida dada por la función de activación $a^l(z)$.

3. Se obtiene el error en la última capa dado por la ecuación:

$$\delta^l = \frac{\partial C}{\partial a^L} \cdot \frac{\partial a^L}{\partial z^L} \tag{2.15}$$

4. Se retropropaga el error a cada $l = L-1, L-2, ..., 2$ capa mediante la ecuación:

$$\delta^l = W^L \cdot \delta^L \cdot \frac{\partial a^{l+1}}{\partial z^{l+1}} \tag{2.16}$$

5. Se utiliza el gradiente de la función de la función de costo dados por:

$$\frac{\partial C}{\partial w^L} = a^{l-1}\delta^l \tag{2.17}$$

6. Se emplea el descenso del gradiente para obtener los nuevos pesos W; esto es, se introduce la ecuación 2.17

2.3. Software utilizado

2.3.1. MATLAB

MATLAB, abreviatura de "*matrix laboratory*"(laboratorio de matrices), es un sistema de cómputo numérico con un lenguaje de programación (lenguaje M) basado en matrices con gráficas integradas que facilitan la visualización de los datos y la obtención de información a partir de ellos. Permite leer formatos de archivos comunes, como hojas de cálculo, texto, imágenes, audio y vídeo, y formatos de datos científicos. Estos datos pueden necesitar técnicas de preprocesamiento

para asegurar un análisis preciso, eficiente y significativo. MATLAB cuenta con diversas funciones para realizar dichas tareas (Mathworks, sf).

MATLAB ofrece una herramienta para el diseño e implementación de redes neuronales conocida como Deep Learning Toolbox. Las redes de reconocimiento de patrones en esta toolbox son redes de tipo feedforward que se pueden entrenar para clasificar entradas en clases específicas. Los datos objetivo de estas redes deben consistir en vectores one-hot (Mathworks, sf).

2.3.2. Kotlin

Kotlin es un lenguaje de programación de código abierto y tipado estáticamente que admite tanto la programación funcional como la orientada a objetos. Su sintaxis y conceptos son similares a los de otros lenguajes como C#, Java y Scala, entre otros. Kotlin es interoperable con Java, lo que permite invocar código Java desde Kotlin y viceversa. Google respalda oficialmente a Kotlin para el desarrollo en Android, lo que significa que la documentación y las herramientas de Android están diseñadas para ser compatibles con Kotlin (Developers., 2023).

Android Studio es el entorno de desarrollo integrado (IDE) oficial utilizado para crear aplicaciones en Android. Proporciona herramientas para depurar y optimizar el rendimiento del código, así como una variedad de marcos de trabajo y herramientas de prueba. Además, Android Studio es compatible con múltiples APKs, lo que permite generar de manera eficiente varios APKs adaptados a diferentes densidades de pantalla. Esto facilita la creación de aplicaciones para diversas versiones de Android y configuraciones de dispositivos (Developers., 2023).

Capítulo 3

Metodología

3.1. Construcción de la base de datos utilizando una cámara CCD

3.1.1. Arreglo experimental

A continuación se describe el arreglo experimental utilizado para generar las imágenes empleadas en el entrenamiento de la red neuronal montado en el Laboratorio de Óptica y Fotónica del Centro de Física Aplicada y Tecnología Avanzada. La técnica aquí descrita permitió observar patrones de difracción bien definidos a partir de agua, alcoholes y mezclas binarias, que proporcionaron una huella dactilar del efecto fisicoquímico en dichos líquidos.

Primero, se generó luz visible súpercontinua (SCV) a partir de un cristal de zafiro bombeado con un láser de femtosegundo de 1040 nm con aproximadamente 0,427 mW de potencia media. Dicho láser se utilizó como luz de excitación en los líquidos. La región visible del súpercontinuo se utilizó como haz para sondear. La disposición óptica incluyó cuatro lentes (L1, L2, L3, L4), un divisor de haz 50/50 (BS), dos espejos (M1, M2) y un divisor de haz de polarización lineal variable (PBS). También se utilizaron una placa motorizada de media onda controlada por ordenador montada en una platina giratoria (M-WP), un tapón de haz (M-BS) y dos cámaras (CMOS, EOS-M50). Se utilizó un interferómetro de Michelson (haz verde) para controlar los cambios mínimos de temperatura en la muestra. En dicho interferómetro, se utilizó un LED de 520 nm como fuente de luz y las franjas de interferencia se monitorizarán con una cámara CMOS.

Las muestras se colocaron en una célula de cuarzo (QC) de 1 cm de longitud. La posición de la célula se ajustó y se colocó perpendicular a los rayos láser de tal forma que la luz enfocada se encontrará aproximadamente en el centro de la célula. Los patrones de difracción de campo lejano estables se proyectaron en una pantalla blanca y se fotografiaron con una cámara Cannon EOS-M50 con ISO 400 (Gordon *et al.*, 1965; Dabby *et al.*, 1970; Greenfield *et al.*, 2011; Dominguez-Juarez *et al.*, 2015). El arreglo experimental descrito se muestra en la figura 3.1.

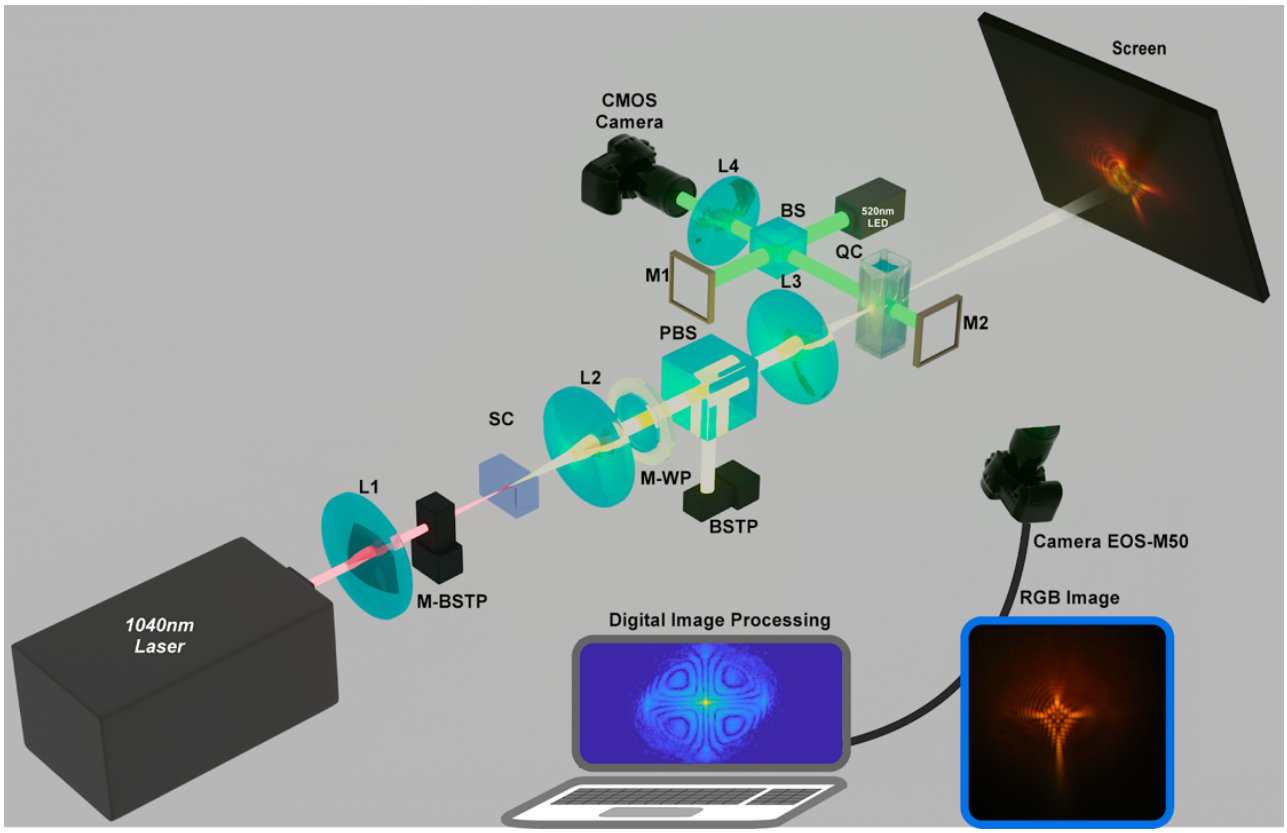


Figura 3.1: Esquema del montaje experimental utilizado para fotografiar los patrones de difracción del agua y las mezclas binarias (elaborada por Dr. Jorge Luis Domínguez Juárez).

Se adquirieron fotografías de 4 mezclas binarias de alcohol-agua con fracciones molares que se muestran en la tabla 3.1 con la correspondiente fracción molar de agua $X-_{H2O}$ y alcohol $X-_{OH}$ en las mezclas preparadas. Se obtuvieron 480 fotografías con propanol-1, 480 fotografías con propanol-2, 488 fotografías con etanol y 480 fotografías con metanol. Así mismo, se adquirieron 235 fotografías de agua sin algún alcohol presente. Las fotografías se agruparon en 5 carpetas de acuerdo a su composición química y se nombraron de acuerdo al alcohol y la concentración de éste en la muestra

Tabla 3.1: Fracción molar de agua $X-_{H2O}$ y alcohol $X-_{OH}$ en las mezclas binarias orgánicas polares puras

$X-_{H2O}$	$X-_{OH}$	$X-_{H2O}$	$X-_{OH}$
Agua	Alcohol	Agua	Alcohol
0	1	0.65	0.35
0.05	0.95	0.7	0.3
0.1	0.9	0.75	0.25
0.15	0.85	0.8	0.2
0.2	0.8	0.825	0.175
0.25	0.75	0.85	0.15
0.3	0.7	0.875	0.125
0.35	0.65	0.9	0.1
0.4	0.6	0.925	0.075
0.45	0.55	0.95	0.05
0.5	0.5	0.975	0.025
0.55	0.45	0.99	0.01
0.6	0.4	1	0

3.1.2. Pre-procesamiento de las imágenes

Las imágenes RGB capturan de manera efectiva los objetos y sus colores reales; sin embargo, cuando se almacenan digitalmente, contienen una gran cantidad de información. Las fotografías adquiridas con la cámara CCD, con una resolución de 6000 x 4000 píxeles y en formato RGB, ocupaban aproximadamente 1.7 MB cada una. Como resultado, el conjunto de imágenes generado representaba un volumen significativo de datos a analizar. Para reducir la demanda de recursos computacionales necesaria para este análisis, fue imprescindible desarrollar un algoritmo de preprocesamiento para optimizar el conjunto de observaciones.

El primer paso en el preprocesamiento consistió en la lectura y etiquetado de cada una de las imágenes en MATLAB, clasificándolas según la mezcla bioquímica a la que pertenecían: agua, metanol, etanol, 1-propanol y 2-propanol. Dado que el patrón de difracción solo aparece en una porción específica de la fotografía (figura 3.2), el siguiente paso fue reducir el número de píxeles de cada imagen para eliminar la mayor cantidad de información no relevante, es decir, áreas que no contenían el patrón de difracción. El tamaño de recorte se seleccionó considerando la ubicación y cantidad de píxeles que contenían el patrón de difracción en las muestras con mayor concentración de alcohol. Además, se realizó una conversión de las imágenes del formato RGB a escala de grises, lo que permitió reducir aún más la cantidad de información a analizar. Como resultado, se obtuvieron imágenes de 1001 x 1001 píxeles en escala de grises, tal como se muestra en la figura 3.3.

Figura 3.2: Fotografía de la mezcla alcohol-agua con propanol-1 en una concentración de 19 %

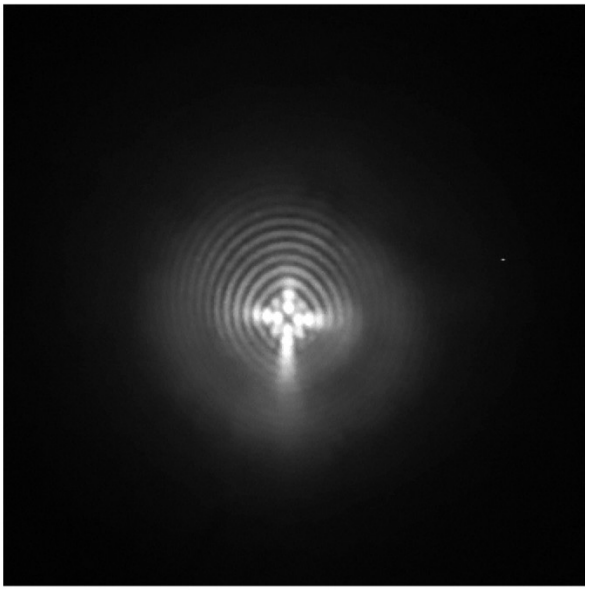


Figura 3.3: Fotografía recortada y en escala de grises de la mezcla alcohol-agua con propanol-1 en una concentración de 19 %

El siguiente paso fue pasar la información de las imágenes a una matriz de 1001 x 1001 en formato *double* como se muestra en la figura 3.4. Es importante mencionar que la figura mostrada implementa colores falsos para hacer referencia a las intensidades por píxel.

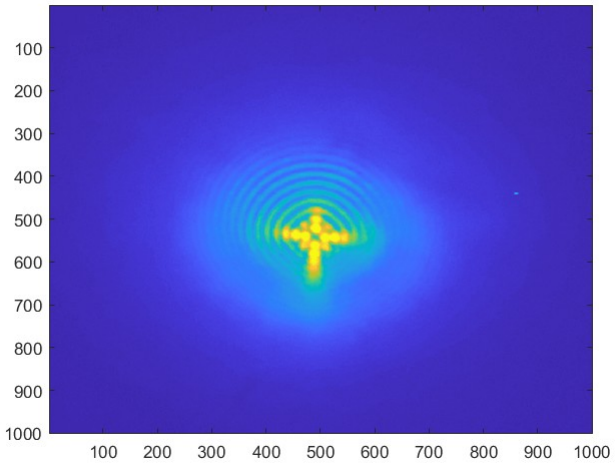


Figura 3.4: Representación gráfica de la matriz de la figura 3.3

En el procesamiento digital de imágenes, existe una técnica denominada *downsampling*, que consiste en reducir el número de píxeles de una imagen en función de la frecuencia de muestreo. En esta técnica, el valor de un píxel en la imagen reducida se calcula como el valor medio de todos los píxeles dentro de un bloque específico. Al aplicar *downsampling*, se disminuye tanto la resolución como el tamaño de la imagen, lo que a su vez reduce la cantidad de información almacenada (Intel, sf). Dada la utilidad de esta técnica, se decidió implementarla como el último paso del preprocesamiento de las imágenes (figura 3.5). El código utilizado para aplicar *downsampling* se incluye en el anexo 8.4.

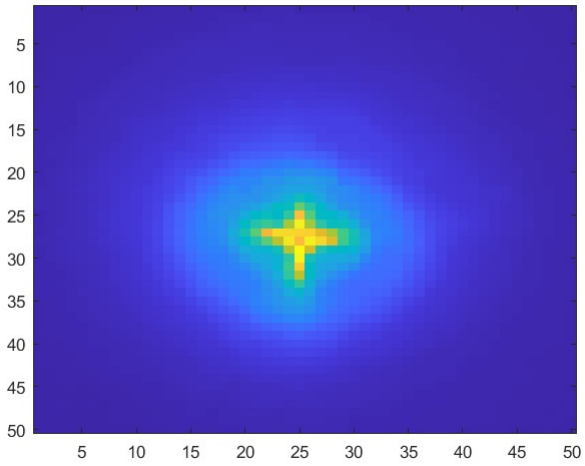


Figura 3.5: Downsampling aplicado a la figura 3.4

.

El preprocesamiento aplicado permitió reducir las fotografías originales de 6000 x 4000 píxeles a matrices de 50 x 50 píxeles, disminuyendo significativamente la cantidad de información

utilizada para el entrenamiento de la red neuronal. El tamaño del downsampling fue seleccionado durante la etapa de entrenamiento. Para ingresar las imágenes en la red neuronal, estas se almacenaron como vectores columna, los cuales se combinaron para formar una matriz que contiene la información de todas las fotografías. Además, se generó una matriz objetivo, donde cada columna correspondiente a una imagen, para indicar la clase a la que pertenece. Cada columna de esta matriz incluye un valor de uno en la fila correspondiente a la clase de la imagen y ceros en los demás elementos. El código empleado para el preprocesamiento de las imágenes y la generación de las matrices utilizadas en el entrenamiento de la red neuronal se encuentra en el anexo 8.5.

3.2. Desarrollo de la red neuronal

Debido a que Matlab permite la codificación de algoritmos a través de un lenguaje de alto nivel e implementa una amplia gama de herramientas, se eligió como software para el desarrollo de la red neuronal propuesta. A continuación se describe el proceso de diseño y desarrollo de la red neuronal propuesta para la identificación de las mezclas bioquímicas.

3.2.1. MATLAB *patternnet*

En los problemas de reconocimiento de patrones, se necesita una red neuronal para clasificar las entradas en un grupo de categorías. Una red de dos niveles con una función de activación sigmoide en la capa oculta y una función de activación softmax para la capa de salida es una arquitectura útil para la resolución de ese tipo de problemas. Dentro del toolbox de MATLAB se encuentra una red neuronal con la arquitectura mencionada a la cual podemos acceder con el comando *patternnet* (Kim y Park, 2018).

La sintaxis del comando es la siguiente:

net = patternnet(hiddenSizes,trainFcn,performFcn)

donde

- **hiddenSizes** es el número de neuronas en la capa oculta (predeterminada = 10)
- **trainFcn** es la función de entrenamiento (predeterminada = 'trainscg')
- **performFcn** el la función de costo (predeterminada = 'crossentropy').
- **net** es la red de reconocimiento de patrones generada.

Para la red neuronal que habilita la identificación de mezclas binarias, se seleccionó la función de entrenamiento 'traingd', la cual utiliza el algoritmo de backpropagation en MATLAB para

Tabla 3.2: Taza de aprendizaje contra entropía cruzada para la red *patternet* con 5 neuronas en la capa oculta y 5000 épocas.

Taza de aprendizaje	0.01	0.005	0.001	0.0005	0.0001
Entropía cruzada	0.0858	0.126	0.179	0.249	0.275

Tabla 3.3: Número de neuronas en la capa oculta contra entropía cruzada para la red *patternet* con un tamaño de paso de 0.005 y 5000 épocas.

No. de neuronas en la capa oculta	5	10	15	20	25
Entropía cruzada	0.177	0.155	0.115	0.179	0.101

actualizar la matriz de pesos W y el sesgo b mediante el método de gradiente descendente (ecuación 2.14) (Mathworks, sf). La función de costo empleada fue la función predeterminada, es decir, la entropía cruzada (ecuación 2.11).

Para el entrenamiento de la red, los datos se dividieron aleatoriamente en tres conjuntos: 70 % se destinó al entrenamiento, 15 % a la validación, y 15 % a las pruebas. El propósito de esta división fue identificar los hiperparámetros que minimizaran la entropía cruzada, lo cual reflejaría un mejor rendimiento en el reconocimiento de imágenes. Se llevaron a cabo pruebas variando la tasa de aprendizaje con los valores [0.01, 0.005, 0.001, 0.0005, 0.0001], y se observó que las tasas de 0.005 y 0.01, con 5000 épocas, producían el valor más bajo en la función de costo. Los resultados de una iteración representativa en la búsqueda de la tasa de aprendizaje óptima se muestran en la figura 3.2.

Además, dado que en la construcción de la base de imágenes se establecieron cinco categorías, se decidió que el número de neuronas en la capa oculta sería un múltiplo de cinco. Se realizaron pruebas con diferentes múltiplos, y se seleccionó el número de neuronas que proporcionó el menor valor en la función de costo. Los resultados se presentan en la tabla 3.3. A partir de este estudio, se eligió una tasa de aprendizaje de 0.005 y 10 neuronas en la capa oculta

La red *patternnet* permite ver la matriz de confusión del entrenamiento. Esta matriz nos indica el porcentaje de fotografías que clasificó correctamente y el porcentaje que clasificó incorrectamente para cada clase. Para las pruebas realizadas se revisó esta matriz para observar el desempeño de la red para la clasificación de las imágenes. Los resultados obtenidos se encuentran en la sección 4.1. El código utilizado para el entrenamiento así como la red con los parámetros elegidos se pueden consultar en el anexo 8.3

3.2.2. Algoritmo Backpropagation

Para el entrenamiento de la red neuronal multicapa fue necesario realizar el desarrollo matemático del algoritmo Backpropagation para posteriormente programarlo tanto en MATLAB como en Kotlin. A continuación se describe dicho desarrollo matemático.

Para el entrenamiento es necesario saber cómo se ve afectado el costo C ante un cambio del parámetro W. Este cambio en los pesos se puede escribir matemáticamente como una derivada parcial $\frac{\partial C}{\partial W}$. Sin embargo, como la función de costo también depende del sesgo b, es necesario calcular también la derivada parcial $\frac{\partial C}{\partial b}$.

Desarrollo para la última capa

El vector de salida de las neuronas de cada capa es el resultado de la suma ponderada dada por la ecuación 2.8. Para la última capa de la red, denotada por el superíndice L, la ecuación es:

$$Z^L = W^L X + b^L \tag{3.1}$$

es decir, es el producto de la matriz de pesos W por la matriz de entradas X más el sesgo b. A esta suma ponderada Z se le aplica una función de activación a, tal que $a(Z^L)$. Posteriormente se utiliza la función de costo C, que se puede escribir como:

$$C(a(Z^L), d) \tag{3.2}$$

donde d es el objetivo. Debido a que ésta es una composición de funciones, es necesario utilizar la regla de la cadena para obtener las derivadas

$$\frac{\partial C}{\partial W^L} = \frac{\partial C}{\partial a^L} \cdot \frac{\partial a^L}{\partial Z^L} \cdot \frac{\partial Z^L}{\partial W^L} \tag{3.3}$$

$$\frac{\partial C}{\partial b^L} = \frac{\partial C}{\partial a^L} \cdot \frac{\partial a^L}{\partial Z^L} \cdot \frac{\partial Z^L}{\partial b^L} \tag{3.4}$$

De estas derivadas se pueden hacer las siguientes observaciones:

- $\frac{\partial C}{\partial a^L}$ indica cómo varía el costo de la red cuando variamos la activación (salida) de las neuronas en la última capa.
- $\frac{\partial a^L}{\partial Z^L}$ es la deriva de la función de activación a con respecto a la suma ponderada Z. Esta derivada indica cómo varía la salida de la neurona cuando variamos la suma ponderada de la neurona.

- $\frac{\partial Z^L}{\partial b^L}$ indica cómo varía la suma ponderada Z con respecto a una variación en el sesgo. De la ecuación 3.1 se observa que
$$\frac{\partial Z^L}{\partial b^L} = 1 \tag{3.5}$$
- $\frac{\partial Z^L}{\partial W^L}$ indica cómo varía la suma ponderada Z con respecto a una variación en los pesos W.

En esta última expresión, la derivada de Z con respecto a los pesos W, es la salida a^{L-1} de la capa anterior, es decir:
$$\frac{\partial Z^L}{\partial W^L} = a_i^{L-1} \tag{3.6}$$
donde el índice i representa a la neurona i-ésima en la capa L.

Analizando la ecuación 3.3 se puede observar que
$$\frac{\partial C}{\partial a^L} \cdot \frac{\partial a^L}{\partial Z^L} = \frac{\partial C}{\partial Z^L} \tag{3.7}$$
La derivada de la ecuación 3.7 representa cómo varía la función de costo cuando cambia la suma ponderada. Esto nos indica en qué grado se modifica el error cuando se produce un cambio en la suma de una neurona, es decir, nos indica la responsabilidad de dicha neurona en el resultado final. Con esto definimos el error imputado a la neurona como:
$$\delta^L = \frac{\partial C}{\partial Z^L} = \frac{\partial C}{\partial a^L} \cdot \frac{\partial a^L}{\partial Z^L} \tag{3.8}$$
Sustituyendo las ecuaciones 3.5 y 3.8 en la ecuación 3.4 se obtiene la derivada del costo respecto al sesgo:
$$\frac{\partial C}{\partial b^L} = \delta^L \tag{3.9}$$
Así mismo, sustituyendo las ecuaciones 3.6 y 3.8 en la ecuación 3.3 se obtiene la derivada del costo respecto al peso, es decir, el error de las neuronas multiplicado por la activación de la capa previa:
$$\frac{\partial C}{\partial W^L} = \delta^L a_i^{L-1} \tag{3.10}$$

Cálculo para las capas ocultas

Para las capas ocultas $L-1, L-2, ..., 1$ la función de costo está dada por la expresión:
$$C(a^L(Z^L)) = C\{a^L\left[W^L a^{L-1}(Z^{L-1}) + b^L\right]\}$$
$$C\{a^L\left[W^L a^{L-1}(W^{L-1}a^{L-2} + b^{L-1}) + b^L\right]\} \tag{3.11}$$
Se aplica la regla de la cadena:
$$\frac{\partial C}{\partial W^{L-1}} = \frac{\partial C}{\partial a^L} \cdot \frac{\partial a^L}{\partial Z^L} \cdot \frac{\partial Z^L}{\partial a^{L-1}} \cdot \frac{\partial a^{L-1}}{\partial Z^{L-1}} \cdot \frac{\partial Z^{L-1}}{\partial W^{L-1}} \tag{3.12}$$

$$\frac{\partial C}{\partial b^{L-1}} = \frac{\partial C}{\partial a^L} \cdot \frac{\partial a^L}{\partial Z^L} \cdot \frac{\partial Z^L}{\partial a^{L-1}} \cdot \frac{\partial a^{L-1}}{\partial Z^{L-1}} \cdot \frac{\partial Z^{L-1}}{\partial b^{L-1}} \tag{3.13}$$

De estas derivadas se pueden hacer las siguientes observaciones:

- De la ecuación 3.8 se observa que $\frac{\partial C}{\partial a^L} \cdot \frac{\partial a^L}{\partial Z^L} = \delta^L$.
- De la ecuación 3.6
$$\frac{\partial Z^{L-1}}{\partial W^{L-1}} = a^{L-2} \tag{3.14}$$
- De la ecuación 3.5
$$\frac{\partial Z^{L-1}}{\partial b^{L-1}} = 1 \tag{3.15}$$
- $\frac{\partial a^{L-1}}{\partial z^{L-1}}$ es la derivada de la función de activación respecto a la salida ponderada.
- $\frac{\partial z^L}{\partial a^{L-1}}$ indica cómo varía la suma ponderada de la capa, cuando se varía el output de una neurona en la capa previa. Esta derivada es la matriz de pesos W^L, es decir
$$\frac{\partial z^L}{\partial a^{L-1}} = W^L \tag{3.16}$$

Sustituyendo las ecuaciones 3.8, 3.14, 3.15 y 3.16 en la ecuación 3.12:

$$\frac{\partial C}{\partial W^{L-1}} = W^L \cdot \delta^L \cdot a^{L-2} \cdot \frac{\partial Z^{L-1}}{\partial W^{L-1}} \tag{3.17}$$

Se define el error en las neuronas de la capa $L-1$ como

$$\delta^{L-1} = W^L \cdot \delta^L \cdot a^{L-2} \tag{3.18}$$

Con esto la generalización para las capas $l = L-1, L-2, ...2$ de las ecuaciones 3.12 y 3.13 son:

$$\frac{\partial C}{\partial W^l} = \delta^l \cdot a^{l-1} \tag{3.19}$$

$$\frac{\partial C}{\partial b^l} = \delta^l \tag{3.20}$$

Derivada de la función de costo

Cuando se define la función de costo a utilizar en la red neuronal se puede obtener la expresión matemática para el error en la última capa dada por la ecuación 3.8. Sea la función de costo el error cuadrático medio dado por la ecuación 2.9, entonces esta derivada está dada por :

$$\frac{\partial C(a^L)}{\partial a^L} = \frac{1}{2}\frac{\partial}{\partial a_i^L}\sum_{i=0}^{N}(y_i - a_i^L)^2 = (a_j^L - y_j) \tag{3.21}$$

3.2.3. Método de descenso de gradiente con función de costo Entropía Cruzada y neuronas softmax

En esta subsección realizamos la derivación del método del descenso del gradiente cuando se tienen neuronas con funciones de tipo softmax en la capa de salida, neuronas sigmoides en la capa oculta y entropia cruzada como función de costo. Esta arquitectura será la que usemos para realizar la clasificación de mezclas binarias de agua-alcohol. Primero consideremos que a la suma ponderada Z le aplicamos la función de activación Softmax dada por la ecuación 2.7, de la cual obtenemos las siguientes probabilidades,

$$a(Z) = \frac{e^{Z_k}}{\sum_{j=1}^{n} e^{Z_j}} \tag{2.7}$$

La función para la entropía cruzada esta definida por la ecuación 2.11:

$$C = -\sum_{j=1}^{n} y_j \ln(a(Z)) \tag{2.11}$$

donde y_j es el vector que indica a qué clase pertenece la imagen. Como este es un one-hot vector, es decir, es un vector que tiene ceros en todos sus elementos a excepción de la fila que corresponde a la categoría de la imagen, donde tendrá un uno, la función se puede reescribir de la forma:

$$C = -\ln(a(Z)) = -\ln\left(\frac{e^{Z_k}}{\sum_{j=1}^{n} e^{Z_j}}\right) \tag{3.22}$$

Para obtener el error en la última capa dada por la ecuación 3.8, primero reescribimos la función de costo utilizando propiedades de los logaritmos:

$$C = -\ln\left(\frac{e^{Z_k}}{\sum_{j=1}^{n} e^{Z_j}}\right) = -\left[\ln e^{Z_j} - \ln\sum_{j=1}^{n} e^{Z_j}\right] = \ln\sum_{j=1}^{n} e^{Z_j} - Z_k \tag{3.23}$$

El error esta dado por el cambio la función de costo C para cada elemento Z. Para esto derivamos la ecuación 3.23:

$$\frac{\partial C}{\partial Z_i} = \frac{\partial[\ln\sum_{j=1}^{n} e^{Z_j} - Z_k]}{\partial Z_i} = \frac{\partial \ln\sum_{j=1}^{n} e^{Z_j}}{\partial Z_i} - \frac{\partial Z_k}{\partial Z_i} \tag{3.24}$$

Analizando la derivada podemos observar lo siguiente:

- cuando $k \neq i$, la derivada se hace 0
- cuando $k = i$, la derivada es 1

Matemáticamente esto se representa con una delta de Kronecker:

$$\frac{\partial C}{\partial Z_i} = \frac{\partial \ln\sum_{j=1}^{n} e^{Z_j}}{\partial Z_i} - \delta_{ik} \tag{3.25}$$

Hacemos un cambio de variable

$$\varepsilon = \sum_{j=1}^{n} e^{Z_j}$$

$$\frac{\partial C}{\partial Z_i} = \frac{\partial \ln \varepsilon}{\partial Z_i} - \delta_{ik} = \frac{1}{\varepsilon}\frac{\partial \varepsilon}{\partial Z_i} - \delta_{ik} = \frac{1}{\sum_{j=1}^{n} e^{Z_j}}\frac{\partial \sum_{j=1}^{n} e^{Z_j}}{\partial Z_i} - \delta_{ik} \tag{3.26}$$

Como la derivada de e^x es e^x, notamos que al derivar el elemento $\sum_{j=1}^{n} e^{Z_j}$ obtendremos una suma de múltiples e^{Z_j}. Sin embargo, Z_j sólo dependerá de Z_i cuando $i = j$,esto es:

$$\frac{\partial e^{Z_{j=i}}}{\partial Z_i} = e^{Z_j}, \qquad \frac{\partial e^{Z_{j\neq i}}}{\partial Z_i} = 0$$

$$\frac{\partial C}{\partial Z_i} = \frac{1}{\sum_{j=1}^{n} e^{Z_j}}\frac{\partial e^{Z_i}}{\partial Z_i} - \delta_{ik} = \frac{1}{\sum_{j=1}^{n} e^{Z_j}} \cdot e^{Z_j} - \delta_{ik} = \frac{e^{Z_i}}{\sum_{j=1}^{n} e^{Z_j}} - \delta_{ik} \tag{3.27}$$

Analizando observamos que el primer elemento de la ecuación es la función softmax, es decir, la predicción obtenida para nuestro modelo para la clase i y la delta de Kronecker nos dará únicamente un 1 cuando $i = k$, lo que podemos traducir como el *one-hot* vector y_i.

Por lo tanto, el error en la última capa empleando la función de costo entropía cruzada y la función de activación softmax es:

$$\delta^L = \frac{\partial C}{\partial Z} = a(Z^L) - y \tag{3.28}$$

3.2.4. Error en una capa con función Sigmoide

Para calcular el error en la última capa L aplicamos la ecuación 3.8. Esta expresión depende de la derivada de la función de activación respecto a la suma ponderada. Cuando nuestra función de activación es la función sigmoide dada por la ecuación 2.6, entonces la capa oculta $l = L - 1$ tenemos lo siguiente:

$$a^L(Z^L) = \frac{1}{1 + e^{-Z^{L-1}}} \tag{3.29}$$

$$\delta^L = \frac{\partial a^L}{\partial Z^L} = a^L(Z^L) \cdot (1 - a^L(Z^L)) \tag{3.30}$$

Para calcular el error en las capas ocultas usamos la ecuación 3.18. Sustituyendo δ^L, obtenemos el error en la capa oculta dado por:

$$\delta^{L-1} = W^L \cdot \delta^L \cdot a^{L-1}(Z^{L-1}) \cdot (1 - a^{L-1}(Z^{L-1}))$$

$$\delta^{L-1} = W^L \cdot \big[a(Z^L) - y\big] \cdot \big[a^{L-1}(Z^{L-1}) \cdot (1 - a^{L-1}(Z^{L-1}))\big] \tag{3.31}$$

3.3. Desarrollo de la red neuronal para el reconocimiento de mezclas bioquímicas binarias

Se implementó una red neuronal utilizando la arquitectura de la red *patternet* en MATLAB, la cual consiste en una red neuronal artificial con una única capa oculta de neuronas sigmoides y neuronas softmax en la capa de salida, aplicando el método de descenso del gradiente para minimizar la entropía cruzada.

Para entrenar la red neuronal, se programaron las ecuaciones derivadas del algoritmo de retropropagación (Backpropagation). Se empleó la ecuación 3.28 para calcular el error en la capa de salida y la ecuación 3.31 para determinar el error en la capa oculta. La función de costo utilizada fue la entropía cruzada, adecuada para problemas de clasificación multiclase (ecuación 2.12). Para determinar el número óptimo de neuronas en la capa oculta, el tamaño del paso, y el número de épocas, se realizaron múltiples pruebas ajustando estos parámetros con el objetivo de encontrar el conjunto que proporcionara un mayor porcentaje de reconocimiento.

En estas iteraciones, el conjunto de datos se dividió aleatoriamente en un 70 % para entrenamiento y un 30 % para pruebas. La cantidad de imágenes de cada clase se detalla en la tabla 3.4.

Tabla 3.4: Cantidad de matrices pertenecientes a cada clase en cada conjunto de datos adquiridos con la cámara CCD

Clase	Agua con 1-propanol	Agua con 2-propanol	Agua	Agua con etanol	Agua con metanol
Entrenamiento	338	336	155	351	334
Prueba	142	144	80	137	146
Total	480	480	235	488	480

El primer hiperparámetro a encontrar fue el número de neuronas en la capa oculta. Para esto se entrenaron diferentes redes neuronales cambiando el numero de neuronas en la capa oculta en múltiplos de 5 desde el 10 hasta el 50. El numero de épocas se fijó en 2000 y la taza de aprendizaje en 0.0005. El porcentaje de reconocimiento y la entropía cruzada obtenida para cada número de neuronas en la capa oculta se muestra en la tabla 3.5. Se encontró que 35 y 45 fueron los números con un mayor porcentaje de reconocimiento para los datos de pruebas y una menor entropía cruzada.

Tabla 3.5: Porcentaje de reconocimiento y entropía cruzada para cada número de neuronas en la capa oculta

No. de neuronas	10	15	20	25	30	35	40	45	50
Porcentaje de reconocimiento	21.42	21.11	24.04	22.19	26.35	38.37	32.67	46.69	22.19
Entropía cruzada	0.49	0.49	0.48	0.49	0.47	0.42	0.42	0.35	0.49

El segundo hiperparámetro a encontrar fue la taza de aprendizaje. Para esto se utilizaron los dos números de neuronas en la capa oculta con mejores resultados (35 y 45) y cuatro diferentes tazas de aprendizaje (0.0005, 0.0001, 0.00005 y 0.00001). Se combinaron estos números y se utilizaron 2000 épocas para cada caso. El porcentaje de reconocimiento y la entropía cruzada obtenida para cada taza de aprendizaje con su respectivo número de neuronas en la capa oculta se muestra en la tabla 3.6. Cabe mencionar que todos experimentos se repitieron diez veces para garantizar la robustez de los resultados.

Tabla 3.6: Porcentaje de reconocimiento y entropía cruzada para cada taza de aprendizaje y número de neuronas en la capa oculta

Taza de aprendizaje	5×10^{-4}		1×10^{-4}		5×10^{-5}		1×10^{-5}	
No. de neuronas	35	45	35	45	35	45	35	45
Porcentaje de reconocimiento	70.570	55.932	88.290	65.947	95.993	70.724	79.199	6.555
Entropía cruzada	0.239	0.411	0.147	0.298	0.039	0.295	0.168	0.332

Se encontró que un número de neuronas en la capa oculta igual a 35 y una taza de aprendizaje de 0.00005, proporciona el porcentaje de reconocimiento más alto y la menor entropía cruzada para los datos de pruebas, es decir, las imagenes de los patrones de difracción adquiridos por la cámara CCD. Por este motivo, estos fueron los hiperparámetros elegidos para el entrenamiento de la red neuronal para le reconocimiento de mezclas bioquímicas binarias.

Otras de los factores modificados fue el tamaño para el downsampling. Inicialmente se eligió un tamaño de 20, se probaron tamaños de 15 y 25. Un tamaño de 25 disminuía las dimensiones de la matriz de 50 x 50 píxeles a 40 x 40 píxeles, sin embargo, el porcentaje de reconocimiento se reducía ($\approx 2\,\%$. Para un tamaño de ventana en el downsampling de 15, disminuía el tiempo de pre-procesamiento pero también disminuía el porcentaje de reconocimiento. Este tiempo de pre-procesamiento depende del hardware utilizado para dicha tarea. Se concluyó que modificar el tamaño de downsampling no impactaba significativamente el porcentaje de reconocimiento, un mayor tamaño aumentaba el tiempo de pre-procesamiento y disminuir el tamaño reducía la

cantidad de información para entrenar la red neuronal . Por estos motivos, se dejó el tamaño de downsampling en 20.

3.3.1. Entrenamiento

Para el entrenamiento de la red neuronal se dividió el conjunto de datos en un 70 % para el entrenamiento y un 30 % para las pruebas como se muestra en la tabla 3.4. Con los parámetros elegidos (35 neuronas en la capa oculta, taza de aprendizaje de 0.00005, 5000 épocas y tamaño de down sampling de 20) se obtuvo un porcentaje de reconocimiento de $\approx$ 96.5 %. También se obtuvo el porcentaje de reconocimiento de cada clase, el cuál fue superior al 96 % en cada una. El código generado para el entrenamiento de esta red neuronal se encuentra en el anexo 8.7.

Una vez entrenada la red neuronal, nuevamente se dividió el conjunto de datos de forma aleatoria y se realizó un nuevo entrenamiento, independiente al anterior. Este proceso se repitió en diez ocasiones para comprobar que el porcentaje de reconocimiento en las etapas de prueba y entrenamiento, se mantuviera superior al 90 % planteado en los objetivos. En todas las iteraciones se mantuvo la relación de imágenes por clase en las pruebas respecto a las imágenes por clase totales mostradas en la tabla 3.4. Los resultados obtenidos en la etapa entrenamiento y de prueba se encuentran en la sección 4.2.

3.4. Desarrollo de una aplicación móvil para la identificación de mezclas bioquímicas binarias de alcohol-agua

Debido a la baja complejidad de la red neuronal desarrollada para la identificación de mezclas binarias, en esta sección se describe la implementación de la red neuronal en una aplicación móvil para dispositivos con sistema operativo Android. El objetivo de esta aplicación móvil es aprovechar los recursos del dispositivo móvil, específicamente la cámara, para adquirir patrones de difracción producidos por la espectroscopia fototérmica y realizar la tarea de identificación sin depender de una cámara CCD alineada en el montaje experimental.

La red neuronal fue codificada en el lenguaje de programación Kotlin, aprovechando su compatibilidad con Android para asegurar una integración eficiente y una ejecución optimizada en dispositivos móviles. Se desarrolló una interfaz de usuario intuitiva que permite la captura de imágenes a través de la cámara del dispositivo, las cuales son procesadas en tiempo real por la red neuronal para identificar las mezclas binarias presentes.

3.4.1. Diseño del layout para la aplicación

Para el desarrollo de la aplicación se eligió el entorno de programación Android Studio y se programaron cada una de las tareas necesarias para el funcionamiento de esta. Con el fin de profundizar en el entendimiento y funcionamiento de la programación orientada a objetos se

programaron diversas tareas en el dispositivo movil con sistema operativo Android.

La primera actividad fue el desarrollo de una aplicación que modificara un texto al dar click a un botón. Esta actividad se realizó con el objetivo de familiarizarse con el lenguaje Kotlin y desarrollar acciones utilizando botones. Posteriormente, a esta misma aplicación, se le agregó la función de acceso a la cámara del dispositivo para adquirir una imagen. Así mismo, se agregó la función de guardar las imágenes tomadas en una carpeta en el almacenamiento interno del teléfono.

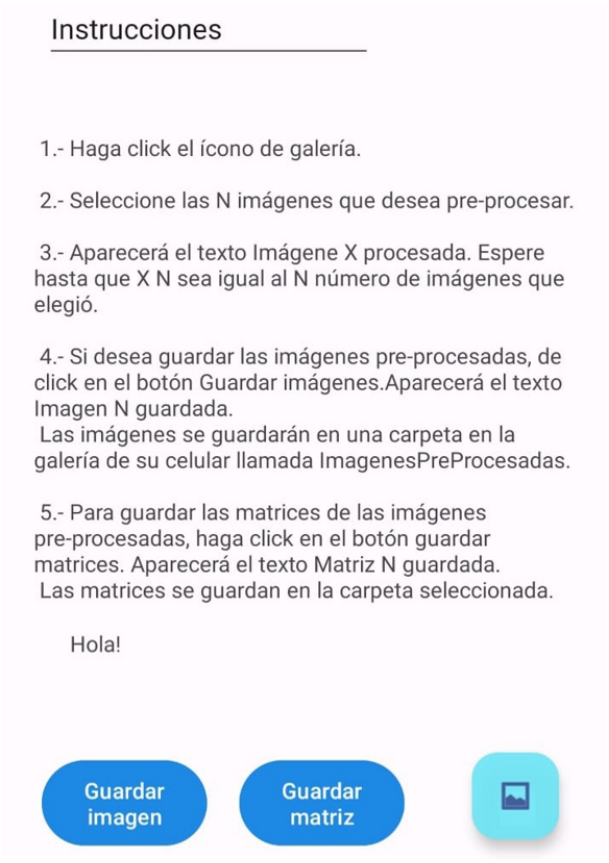


Figura 3.6: Interfaz de la aplicación generada para el pre-procesamiento de las imágenes.

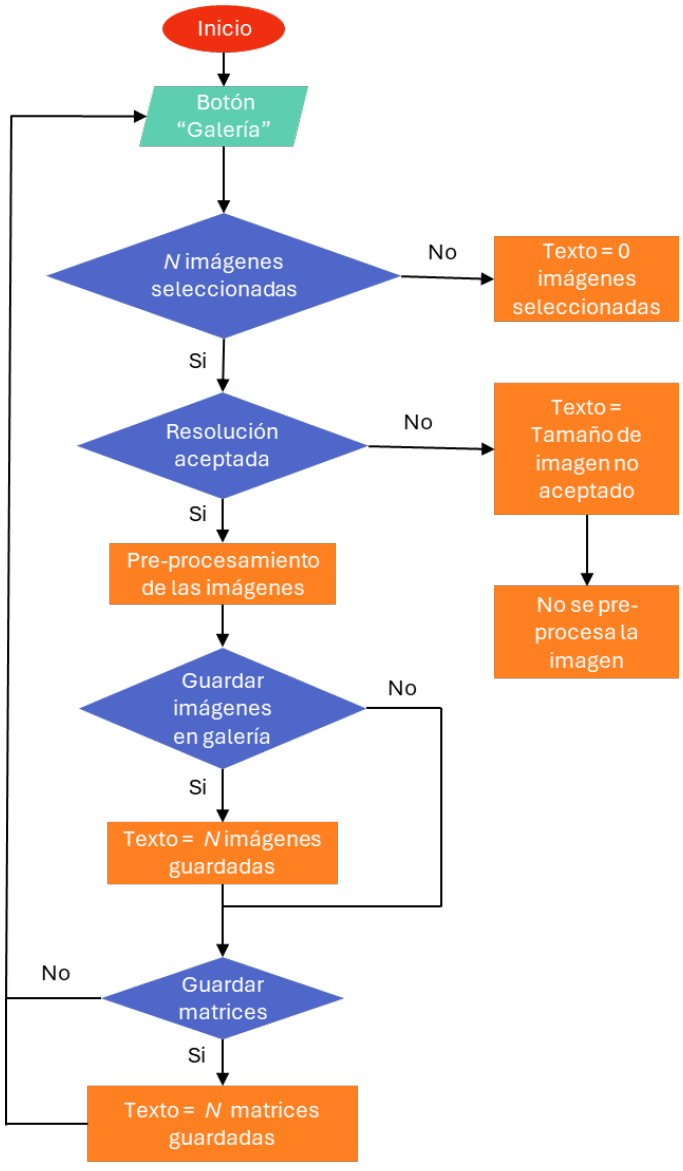


Figura 3.7: Diagrama de flujo de la app creada para el pre-procesamiento de las imágenes

Debido a que la implementación de esta red neuronal está planteada para dispositivos An-

droid, estos dispositivos procesaron las imágenes de los patrones de difracción de la mezcla bioquímica a reconocer. Por este motivo se desarrolló un código en Kotlin para realizar el mismo pre-procesamiento realizado en MATLAB para el conjunto de imágenes tomadas por la cámara CCD.

La aplicación desarrollada cuenta con 3 botones. El botón de la derecha con el icono de galería, permite acceder a la galería del dispositivo y seleccionar una o más imágenes al mismo tiempo. Una vez seleccionadas las imágenes a procesar, aparece un texto que indica qué número de imagen se ha procesado. El pre-procesamiento de las imágenes en el dispositivo Android requiere de una mayor cantidad de tiempo que el pre-procesamiento en un computadora. El código generado para dicho pre-procesamiento se encuentra en el anexo 8.8.

El botón "Guardar imagen"permite guardar las imágenes pre-procesadas en la galería del dispositivo. El botón "Guardar matriz"permite guardar en un archivo de texto las matrices generadas con el pre-procesamiento. La aplicación permite elegir dónde se guardarán dichas matrices. La figura 3.6 muestra la interfaz de la aplicación, la cual cuenta con los tres botones mencionados y las instrucciones para realizar el pre-procesamiento. La figura 3.7 muestra el diagrama de flujo del funcionamiento de la aplicación.

Se realizaron pruebas para comprobar el correcto funcionamiento de la aplicación con una imagen perteneciente a la clase metanol con una concentración de 42.7 %. Se realizó el preprocesameinto de la imagen y se comparó la matriz preprocesada obtenida por la aplicación de Android con la matriz preprocesada obtenida en MATLAB. A partir de esta comparación se encontró que los valores de ambas matrices eran diferentes debido a la forma en la que cada lenguaje procesa las imágenes. Por este motivo se decidió realizar el pre-procesameinto de las imágenes a utilizar para el entrenamiento de la red neuronal únicamente con la aplicación desarrollada en Kotlin.

3.4.2. Construcción de la base de datos mediante una cámara de celular

Las imágenes adquiridas por la cámara del dispositivo móvil se obtuvieron del mismo arreglo experimental utilizado en la construcción de la base de datos de la cámara CCD (como se describe en la sección 3.1). En este caso, la cámara Canon EOS-M50 fue sustituida por la cámara del dispostivo móvil. La figura 3.8 muestra una parte del arreglo experimental y el procedimiento seguido para la generación de la base de datos.

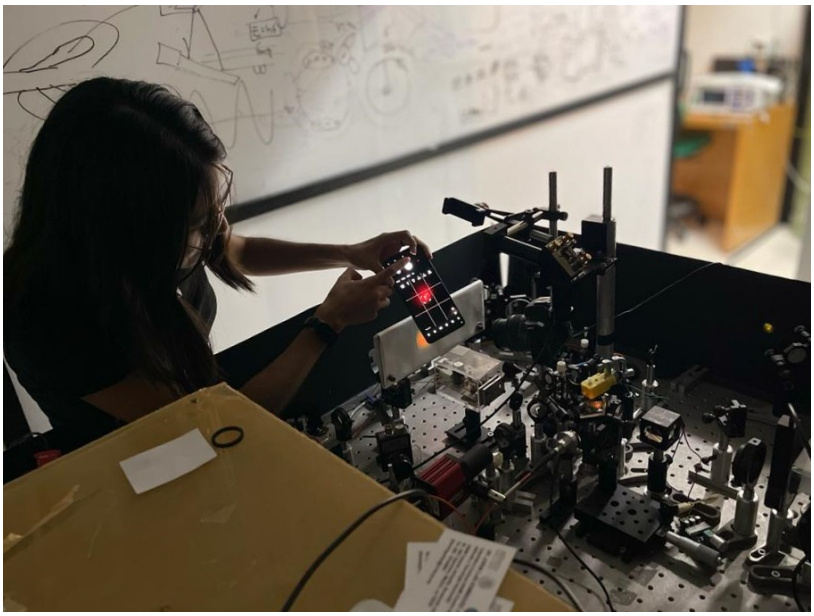


Figura 3.8: Fotografía tomada durante la toma de videos para la contrucción de la base de datos

Se tomaron vídeos de 30fps cambiando el ángulo de la cámara respecto al patrón de difracción para una misma muestra, esto con el objetivo de considerar las posibles variaciones que puedan existir cuando el usuario realice la adquisición de una imagen del patrón de difracción generado por la muestra. Posteriormente se extrajeron las fotogramas de dichos vídeos y se guardaron en diferentes carpetas de acuerdo con el tipo de sustancia bioquímica a la que pertenece la muestra.

Para este conjunto de datos se obtuvieron imagenes de las siguientes sustancias: agua, etanol, metanol y 1-propanol. Se obtuvieron un total de 702 imágenes pertenecientes a la clase agua, 712 imágenes pertenecientes a la clase etanol, 701 imágenes pertenecientes a la clase metanol y 769 imágenes de a la clase 1-propanol.

3.4.3. Pre-procesamiento de las imágenes

La imagenes se pre-procesaron con la aplicación generada para Android (sección 3.4.1). Las fotografías, al ser de 1280 x 720 píxeles, se recortaron para obtener imágenes cuadradas de 500 píxeles de lado y se cambiaron a escala de grises. Se eligió el tamaño de downsampling necesario para poder obtener imágenes de 50 x 50 píxeles, esto con el objetivo de mantener la arquitectura de la red neuronal generada en MATLAB para las imágenes de la cámara CCD (sección 3.3). En la figura 3.9 se muestra una de las fotografías obtenidas de metanol utilizando el celular, en la figura 3.10 se muestra esta misma fotografía después del pre-procesmaiento aplicado. La información de estas imágenes pre-procesadas se guardo en matrices de 50 x 50 en archivos de texto, un archivo por matriz.

Para poder ingresar la información de las imágenes a la red neuronal, se cargaron en MATLAB las matrices generadas mediante la aplicación en Kotlin. En MATLAB se almacenaron en forma de vector columna y los vectores resultantes se unieron para formar una matriz de imágenes. Para indicar a la red a qué clase pertenece cada una de las imágenes, se creó una matriz objetivo con codificación one-hot encoding, es decir, una columna para cada imagen, con un uno en la fila que corresponde a la clase de la imagen y ceros en los demás elementos.

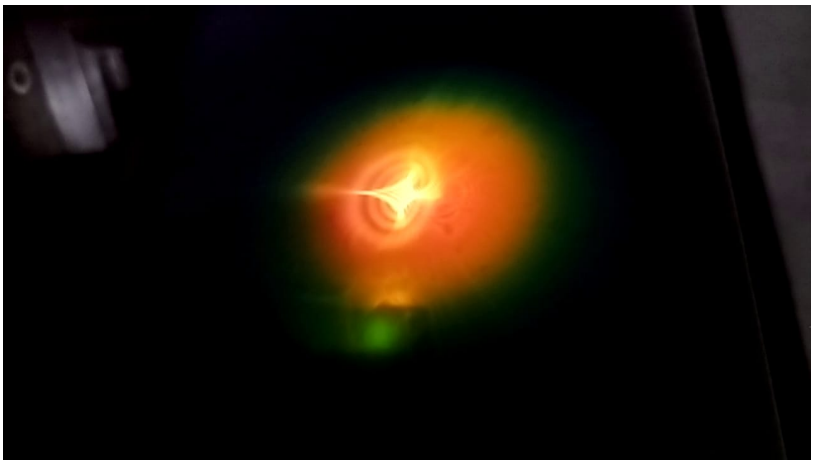


Figura 3.9: Fotografía tomada con un celular del patrón de difracción del metanol

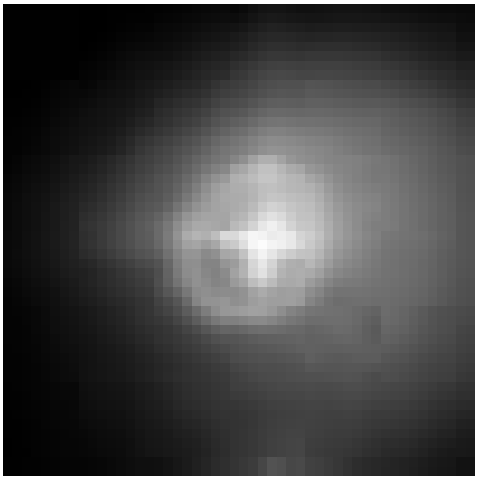


Figura 3.10: Pre-procesamiento de la fotografía tomada con un celular del patrón de difracción del metanol

3.4.4. Entrenamiento de la red neuronal

Para entrenar la red neuronal generada en la sección 3.3, ahora utilizando la base de datos generada mediante la cámara de un celular, se dividió de forma aleatoria el conjunto de datos en un 70 % para entrenamiento, 15 % para pruebas y 15 % para validación. Sin embargo, el numero

de épocas se incrementó hasta 10000 porque la función de costo desciende más lento en comparación con la red neuronal entrenada con imágenes adquiridas con la cámara CCD. La cantidad de fotografías de cada clase perteneciente a cada conjunto se muestran en la tabla 3.7. El código generado para el entrenamiento de esta red neuronal se encuentra en el anexo 8.7

Tabla 3.7: Cantidad de matrices pertenecientes a cada clase en cada conjunto de datos generados con la cámara de celular Android

Clase	1-Propanol	Agua	Etanol	Metanol
Entrenamiento	615	585	548	559
Prueba	154	117	164	142
Total	769	702	712	701

Una vez entrenada la red neuronal, nuevamente se dividió el conjunto de datos de forma aleatoria y se realizó un nuevo entrenamiento, independiente al anterior. Este proceso se repitió en diez ocasiones para comprobar que el porcentaje de reconocimiento en las etapas de prueba y entrenamiento, se mantuviera superior al 90 % obtenido en la primera iteración. En todas las iteraciones se mantuvo la relación de imágenes por clase en las pruebas respecto a las imágenes por clase totales mostradas en la tabla 3.7. Los resultados obtenidos en la etapa entrenamiento y de prueba se encuentran en la sección 4.3.

3.4.5. Implementación de la red neuronal en la aplicación

El desarrollo de la aplicación para la identificación de mezclas bioquímicas binarias de alcohol-agua integra la red neuronal desarrollada en MATLAB en la sección 3.3. Debido a que Kotlin no cuenta con un paquete para realizar las operaciones matemáticas necesarias para el funcionamiento de la red neuronal, éstas se programaron de forma nativa. Primero se programaron las operaciones de multiplicación de matrices y las funciones *sigmoide* y *softmax*, asi como las funciones para mostrar en la terminal los resultados de estas operaciones. El código generado con la red neuronal en lenguaje Kotlin se encuentra en la sección 8.9.

El código generado en la sección 3.3 fue incluido dentro de la aplicación para el preprocesamiento de imágenes. Adicionalmente, se incluyó la función para seleccionar imágenes de la galería, pero se limitó para permitir la selección de únicamente una imagen. Se colocó un recuadro para mostrar en pantalla la imagen elegida, debajo del recuadro se colocó un texto para indicar la clase a la que pertenece la imagen.

Para comprobar el correcto funcionamiento de las funciones matemáticas programadas, se eligió una de las imágenes de la base de datos generada mediante una cámara de dispositivo. La imágen se pre-procesó y se ingresó a la red neuronal programada en MATLAB. Se fueron guardando las matrices resultantes de cada paso en archivos de texto. La misma imágen fue selec-

cionada dentro de la aplicación desarrolla en Kotlin, se pre-procesó y se fue mostrando la matriz resultante de cada paso en la terminal. Las matrices obtenidas en MATLAB se compararon con las matrices obtenidas en Kotlin para comprobar que los valores de cada una coincidieran, indicando que las funciones matemáticas fueron programadas correctamente.

Dentro de la aplicación también se programó una función para tomar un fotografía desde la aplicación. Esta función muestra un recuadro con la vista previa de la cámara. La imagen capturada es guardad en la galería del dispositivo. Se realiza el mismo proceso que al seleccionar la imagen de la galería: la imagen se pre-procesa, se muestra en un recuadro en la pantalla de la aplicación y el texto indica a qué clase pertenece. La figura 3.11 muestra el diagrama de flujo de este proceso.

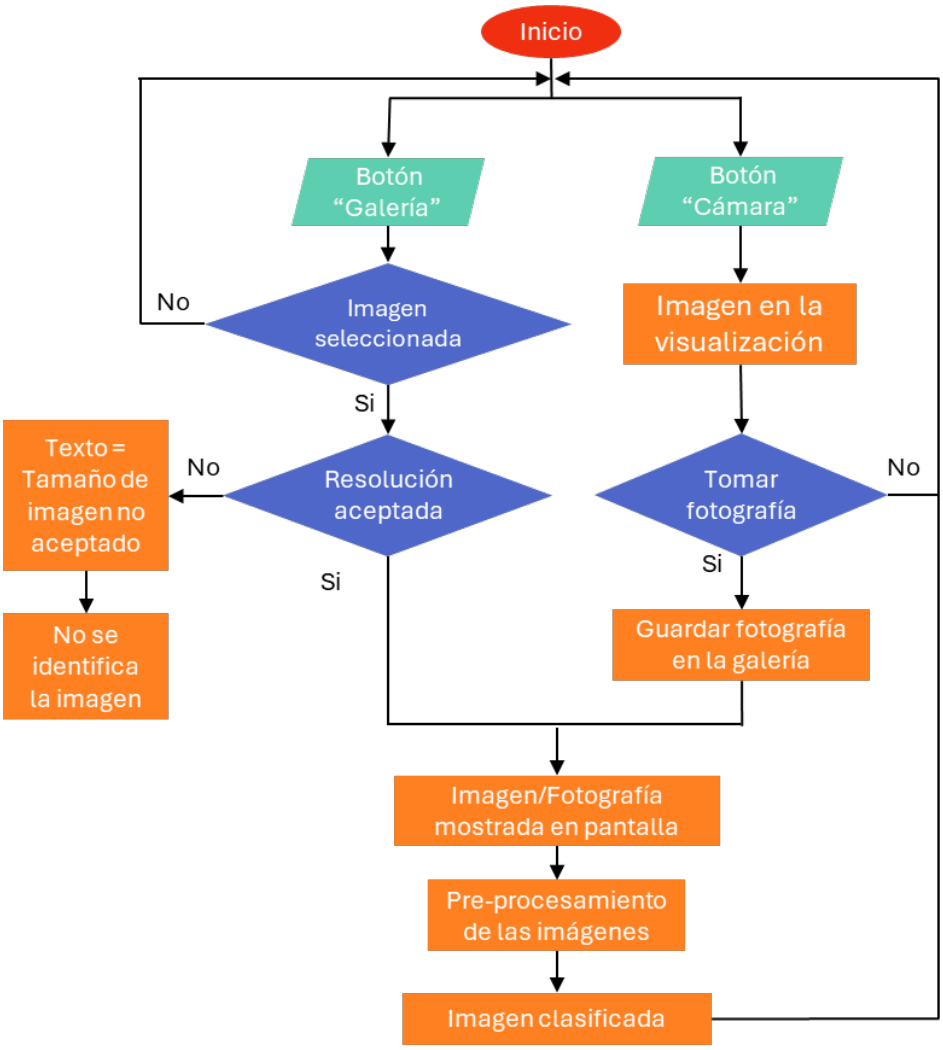


Figura 3.11: Diagrama de flujo de la app creada para la identificación de mezclas bioquímicas

La verificación del correcto funcionamiento de la aplicación se realizó utilizando el arreglo experimental descrito en la sección 3.1. Para ello se tomaron imágenes de los patrones de difrac-

ción generados por las sustancias agua, metanol y 1-propanol. La aplicación permitió la correcta identificación de dichas sustancias. Los porcentajes de reconocimiento se muestran en la sección 4.3.

Adicional a la pantalla principal, desde donde se utiliza la red neuronal artificial para le reconocimiento de sustancias bioquímicas, se agregaron dos pantallas. Una de estas pantallas contiene las instrucciones de cómo utilizar la aplicación. La otra pantalla contiene el *acerca de*, donde se incluye el resumen de este trabajo y los logos de las instituciones involucradas. En la sección 4.4 se muestran imágenes de la interfaz de la aplicación desarrollada así como una explicación más amplia de su contenido.

Capítulo 4

Resultados

4.1. MATLAB *patternnet*

En esta sección se describen los resultados obtenidos con la red neuronal entrenada con imágenes adquiridas con la cámara CCD. Para esto se entrenó una red neuronal *patternnet* de MATLAB con 25 neuronas en la capa oculta, una taza de aprendizaje de 0.005 y 5000 épocas de entrenamiento. Se pre-procesó el conjunto de imágenes obtenido con la cámara CCD utilizando el código generado en MATLAB para dicha tarea y se dividió dicho conjunto de forma aleatoria utilizando un 70 % para el entrenamiento, un 15 % para la validación y un 15 % para las pruebas.

Con los hiperparámetros mencionados y los conjuntos de imágenes generados se obtuvo un 91.7 % de reconocimiento en la etapa de entrenamiento, un 91.4 % en la etapa de validación y un 91.0 % en la etapa de prueba. El porcentaje de reconocmiento utiizando todas la imágenes fue de 91.6 % con una entropía cruzada de 0.076. Esta red neuronal se entrenó en repetidas ocasiones (diez veces) conservando los mismos hiperparámetros para comprobar que en todas la ocasiones los porcentajes de reconocimiento eran similares (entre 90-95 %).

En la imagen 4.1 se puede observar las matrices de confusión que genera la red *patternnet* para cada etapa: entrenamiento, validación, pruebas y general. Esta matriz es una herramienta de evaluación utilizada en problemas de clasificación que contiene las predicciones del modelo frente a las verdaderas categorías. Cada elemento de la matriz de confusión brinda información útil para evaluar el rendimiento de :

- Las filas representan la clase predicha (clase de salida) y las columnas representan la clase real (clase objetivo).
- Las celdas verdes en diagonal corresponden a las observaciones que están clasificadas correctamente.
- Las celdas rojas que no están en diagonal corresponden a las observaciones que están clasificadas incorrectamente:

  - Las celdas en la misma fila contienen los falsos positivos para cada clase, es decir, los casos en los que la red neuronal predijo incorrectamente una instancia de una clase como perteneciente a otra clase.
  - Las celdas en la misma columna contienen los falsos negativos, es decir, los casos en los que la red neuronal falló al predecir una instancia de una clase, clasificándola incorrectamente como otra clase.

- La última columna a la derecha muestra el porcentaje de las imágenes clasificadas correctamente entre las imágenes que la red predijo para cada clase. Los porcentajes en verde representan la Sensibilidad, es decir, la proporción de verdaderos positivos entre los casos positivos reales.

- La fila en la parte inferior muestra el porcentaje de las imágenes clasificadas correctamente entre las imágenes pertenecientes a cada clase. Los porcentajes en verde son el porcentaje de reconocimiento para cada clase.

- Tanto el número de observaciones como el porcentaje del número total de observaciones se muestran en cada celda.

- La celda en la esquina inferior derecha muestra la precisión general.

La clase de la red neuronal a la que pertenece cada mezcla binaria de agua-alcohol es la siguiente:

- Clase 1: agua con 1-propanol

- Clase 2: agua con 2-propanol

- Clase 3: agua

- Clase 4: agua con etanol

- Clase 5: agua con metanol

En la imagen 4.1 se puede observar que la clase 5 (agua con metanol) presentó la menor sensibilidad en todas las etapas. Esto nos indica que la mayoría de las veces que la red neuronal clasificó incorrectamente una mezcla, la identificó como clase 5. Esto ocurrió con el 5.8 % de las imágenes. Sin embargo, se puede observar que todas las clases presentaron un porcentaje de reconocimiento superior al 85 %.

Figura 4.1: Matriz de confusión generada por la red *patternet* con 25 neuronas en la capa oculta, una taza de aprendizaje de 0.005 y 5000 épocas utilizando el conjunto de datos generado con la cámara CCD.

4.2. Red Neuronal entrenada con el conjunto de datos generados por la cámara CCD

En esta sección se describen los resultados de una red neuronal artificial programada en Matlab bajo codificación propia, es decir, no se usaron librerias precargadas en Matlab para diseñar, entrenar y probar la red neuronal propuesta. La red neuronal desarrollada tiene una sola capa oculta de neuronas sigmoide y neuronas softmax en la capa de salida, e implementa el método del descenso del gradiente para minimizar la entropía cruzada generada. Se utilizaron 35

neuronas en la capa oculta, una taza de aprendizaje de 0.00005 y 5000 épocas de entrenamiento. Se preprocesó el conjunto de imágenes adquirido con la cámara CCD y se dividió el conjunto de forma aleatoria utilizando 70 % para el entrenamiento y un 30 % para las pruebas.

Con la arquitectura mencionada, se obtuvo un porcentaje general de reconocimiento del 95.5 % con un valor de entropía cruzada de 0.268. Este porcentaje de reconocimiento es ligeramente mayor al porcentaje de la red *patternnet* (91.4 %). Para la etapa de entrenamiento se obtuvo un 95.8 % de reconocimiento y para la etapa de prueba se obtuvo un 94.8 %. La red neuronal se entrenó en repetidas ocasiones (diez veces) con los mismos hiperparámetros separando el conjunto de datos de forma aleatoria en cada ocasión para comprobar que el porcentaje de reconocimiento no presentara grandes variaciones. En cada uno de esos entrenamiento y pruebas el porcentaje de reconcomiendo fue entre un 93-98 %.

La matriz de confusión fue generada de manera similar a la generada por MATLAB para observar el porcentaje de reconocimiento de cada clase. Las matrices para los datos de entrenamiento y de pruebas se observan en la figura 4.2. La clase de la red neuronal a la que pertenece cada mezcla binaria de agua-alcohol es la misma que en la red *patternnet*, es decir: clase 1: agua con 1-propanol, clase 2: agua con 2-propanol, clase 3: agua, clase 4: agua con etanol, clase 5: agua con metanol.

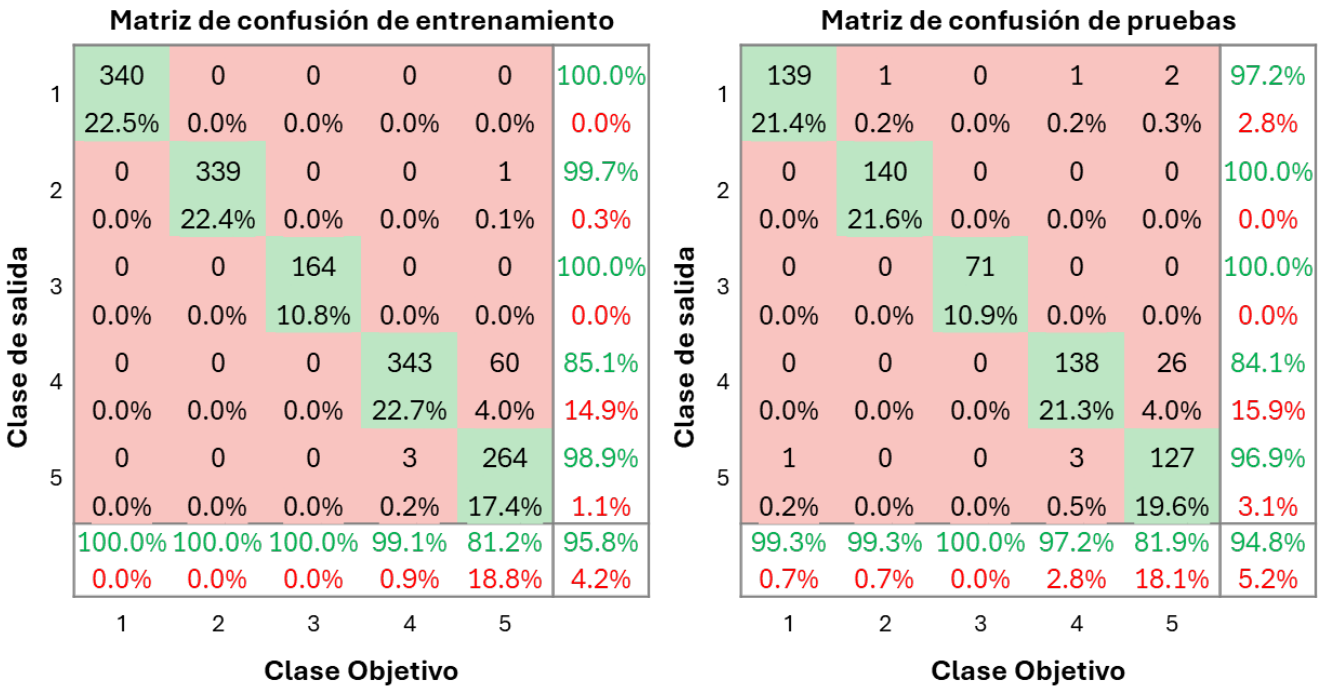


Figura 4.2: Matriz de confusión de la red neuronal programada en MATLAB con una taza de aprendizaje de 0.0005 y 5000 épocas utilizando el conjunto de datos generado con la cámara CCD.

En la figura 4.2 se puede observar que de todas las imágenes que la red neuronal identificó como clase 4 (agua con etanol), un 15-16 % de las imágenes pertenecían a la clase 5 (agua con metanol). Esto representa un 4.0 % del total de las imágenes. Sin embargo, todas las clases presentaron un porcentaje de reconocimiento superior al 81 %, ligeramente inferior al de la red *patternnet* (85 %).

4.3. Red Neuronal entrenada con el conjunto de datos generados con una cámara Android

Como se ha mencionado en la metodología, la arquitectura de la red neuronal es de baja dimensionalidad, motivando su implementación en una aplicación móvil. Para esto, se utilizó la red neuronal artificial, previamente programada en MATLAB, la cual consiste en una sola capa oculta de neuronas sigmoide y una capa de salida con neuronas softmax. Esta red neuronal preserva las características de entrenamiento que la red desarrollada para el conjunto de imágenes capturadas por la cámara CCD, es decir, los pesos se optimizaron usando el método del descenso del gradiente buscando minimizar la entropía cruzada, así como sus hiperparámetros (35 neuronas en la capa oculta, una taza de aprendizaje de 0.00005 y 10000 épocas de entrenamiento). El conjunto de imágenes capturadas por la cámara del dispositivo móvil con sistema operativo Android se pre-procesó utilizando la aplicación generada en Kotlin para dicha tarea y se dividió el conjunto de forma aleatoria utilizando 80 % para el entrenamiento y un 20 % para las pruebas.

Con la arquitectura mencionada, se obtuvo un porcentaje general de reconocimiento del 93.8 % con un valor de entropía cruzada de 0.0118. Este porcentaje de reconocimiento es similar al porcentaje obtenido utilizando el conjunto de datos generado por la cámara CCD (95.5 %). Para la etapa de entrenamiento se obtuvo un 94.6 % de reconocimiento y para la etapa de prueba se obtuvo un 93 %. La red neuronal se entrenó en repetidas ocasiones con los mismos hiperparámetros separando el conjunto de datos de forma aleatoria en cada ocasión para comprobar que el porcentaje de reconocimiento no presentara grandes variaciones. En cada uno de esos entrenamiento y pruebas el porcentaje de reconcomiendo fue entre un 90-95 %.

A partir del entrenamiento y prueba de la red neuronal se generó la matriz de confusión similar a la generada por MATLAB para observar el porcentaje de reconocimiento de cada clase. Las matrices para los datos de entrenamiento y de pruebas se observan en la figura 4.3. La clase de la red neuronal a la que pertenece cada sustancia es la siguiente:

- Clase 1: 1-propanol
- Clase 2: agua
- Clase 3: etanol

- Clase 4: metanol

En la figura 4.3 se puede observar que el porcentaje de reconocimiento para la clase 3 (metanol) fue el más bajo de todas las clases con un 92.3 % en el entrenamiento y un 88.5 % en la prueba. Las clases restantes resentaron porcentajes de reconocimiento mayor al 90 %.

Figura 4.3: Matriz de confusión generada por la red *patternet* con 10 neuronas en la capa oculta y 20000 épocas utilizando el conjunto de datos generado con la cámara CCD.

4.4. Aplicación para dispositivos Android

La aplicación desarrollada se codificó en IDE Android Studio utilizando el lenguaje Kotlin para la identificación de mezclas bioquímicas binarias de agua-alcohol. Esta aplicación contiene la red neuronal previamente entrenada. En la figura 4.4 se puede observar la pantalla principal de la aplicación llamada *Red neuronal*. En la pantalla se muestran dos opciones para obtener la

imagen del patrón de difracción de la mezcla bioquímica a identificar: la cámara y la galería. Se muestra un mensaje de *Bienvenido*, el cual cambia cuando se identifica alguna imagen. En la parte inferior se muestra el menú de navegación, el cual cuenta con 3 botones: *Red neuronal*, *Instrucciones* y *Acerca de*.

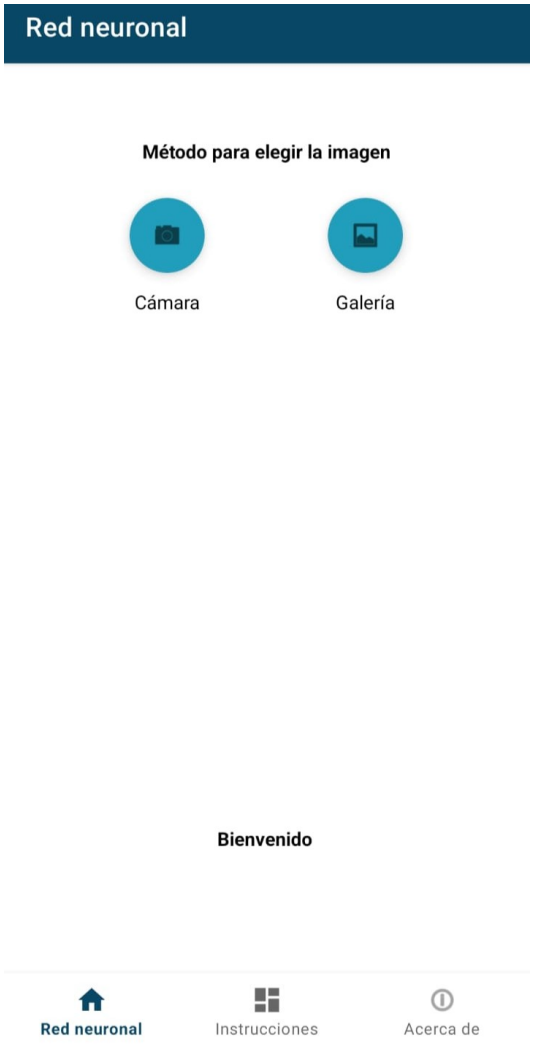


Figura 4.4: Interfaz de la aplicación generada para la identificación de mezclas binarias bioquímicas.

En la figura 4.5 se muestra cómo cambia la pantalla al seleccionar el botón de *Cámara*. En el centro de la pantalla aparece un recuadro mostrando la vista previa de la fotografía a tomar y debajo de este recuadro aparecen dos botones. El botón *TOMAR FOTOGRAFÍA* permite capturar la imagen mostrada en el recuadro y el botón *CANCELAR* regresa la pantalla a su estado inicial, es decir, a la pantalla mostrada en la figura 4.4.

Al capturar la fotografía, los botones *TOMAR FOTOGRAFÍA* y *CANCELAR* desaparecen y la aplicación comienza con el reconocimiento del patrón de difracción. La imagen capturada es guardada en la galería del dispositivo y se muestra en el recuadro del centro. La imagen se pre-

procesa, es decir, se le aplica un cambio de color RGB a escala de grises, y un downsampling, para que posteriormente la información se almacene en una matriz. Esta matriz es ingresada a la red neuronal programada y la mezcla bioquímica binaria es identificada. La figura 4.6 muestra la interfaz cuando la fotografía es tomada, debajo de la fotografía se muestra un texto indicando a qué clase pertenece la mezcla bioquímica binaria: etanol, agua, metanol o 1-propanol.

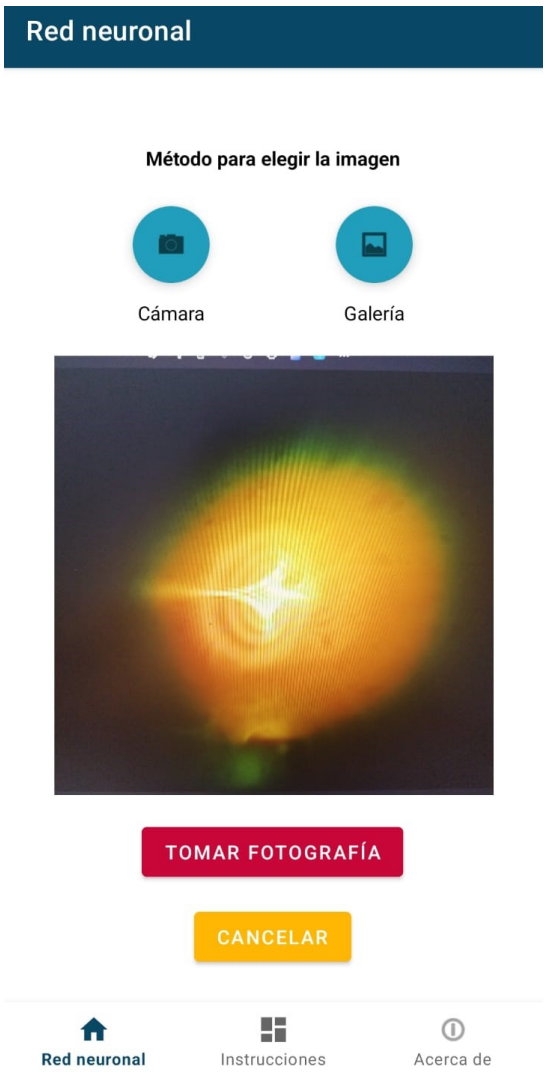


Figura 4.5: Interfaz de la aplicación generada mostrando la cámara.

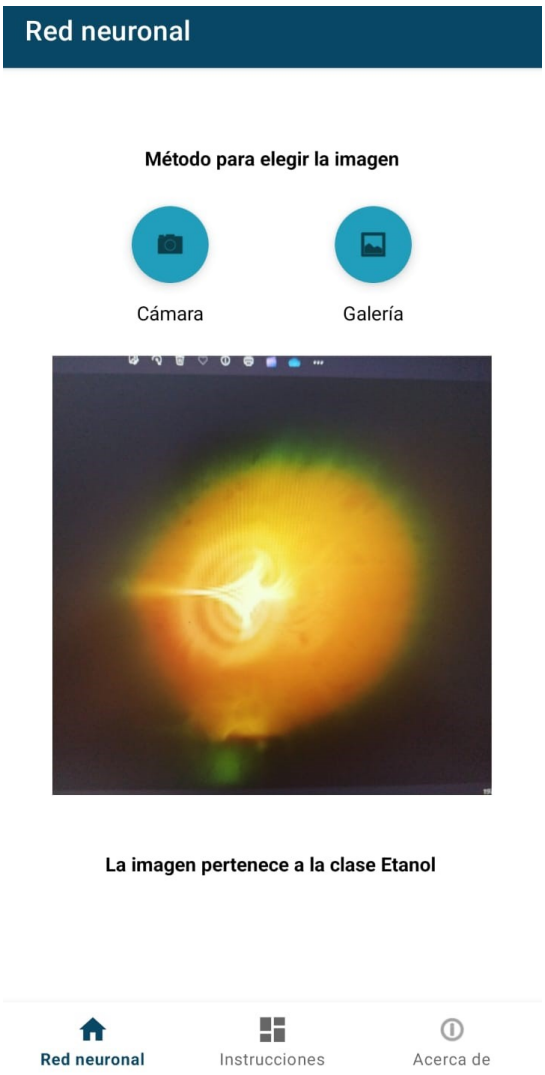


Figura 4.6: Interfaz de la aplicación al tomar la fotografía e identificar la mezcla.

En la figura 4.7 se muestra la interfaz al seleccionar el botón de *Galería*. La aplicación permite elegir entre la galería disponible en el dispositivo Android y permite elegir una única imagen a reconocer. Una vez seleccionada la imagen, se comienza el mismo proceso de pre-procesamiento y reconocimiento que el implementado para la opción de *Cámara*. En la figura 4.8 se muestra la interfaz de la aplicación después de reconocer la imagen seleccionada desde la galería. La imagen seleccionada aparece en el recuadro del centro y debajo aparece el texto indicando a qué clase pertenece la mezcla bioquímica binaria: etanol, agua, metanol o 1-propano.

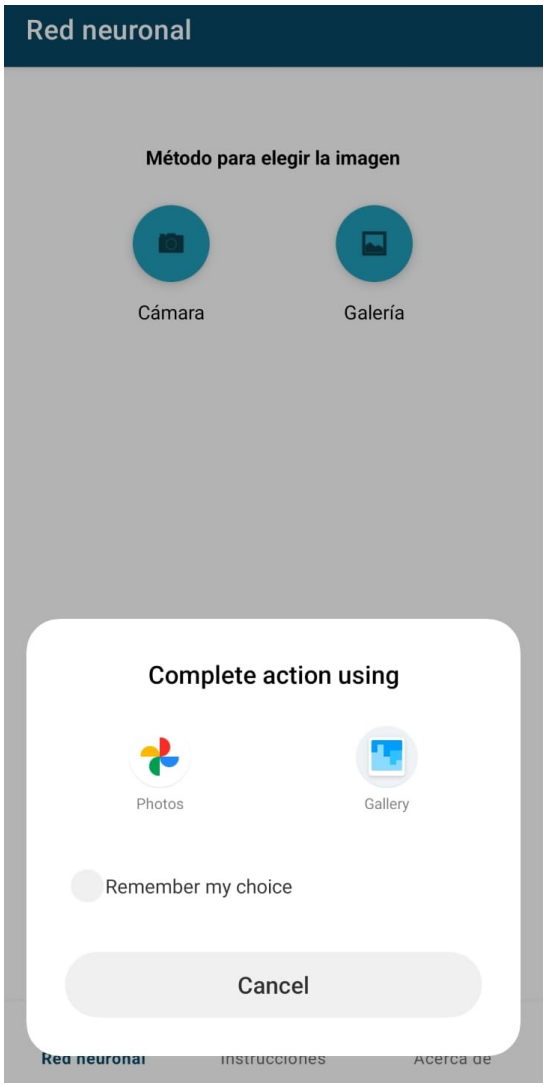


Figura 4.7: Interfaz de la aplicación al seleccionar la opción de *Galería*.

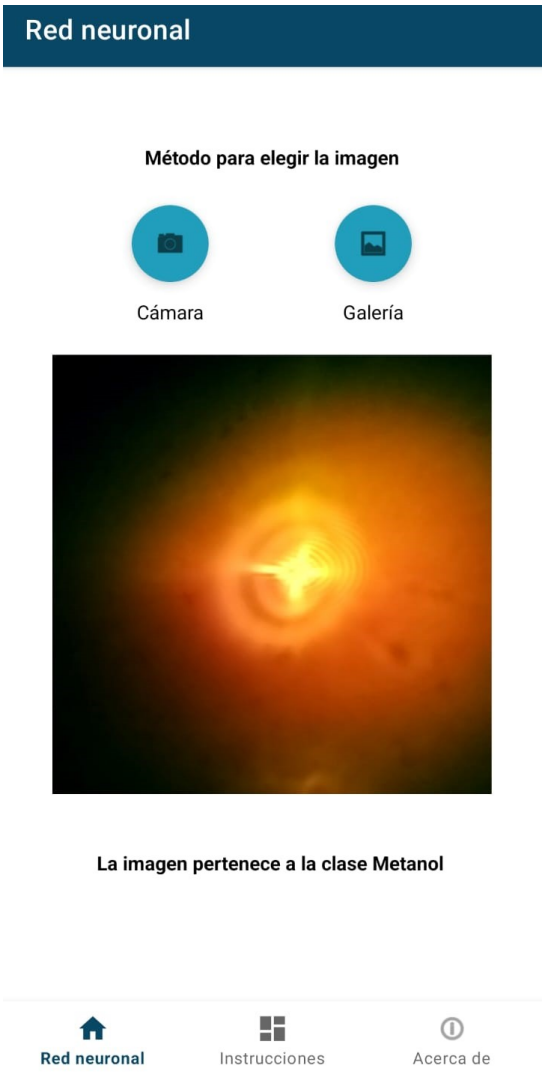


Figura 4.8: Interfaz de la aplicación al seleccionar la imagen de la galería e identificar la mezcla.

En la figura 4.9 se muestra la pantalla de *Instrucciones*. En esta se describe brevemente cómo utilizar la aplicación. Se explican las dos opciones para elegir la imagen a identificar y lo que sucede una vez identificada la imagen. En la figura 4.10 se muestra el resumen de la tesis, así como información sobre el desarrollo y los logos de las instituciones que contribuyeron al desarrollo de este proyecto: la Facultad de Ingeniería (FI) de la Universidad Autónoma de Querétaro (UAQ) y el Centro de Física Aplicada y Tecnología Avanzada (CFATA) de la Universidad Nacional Autónoma de México (UNAM).

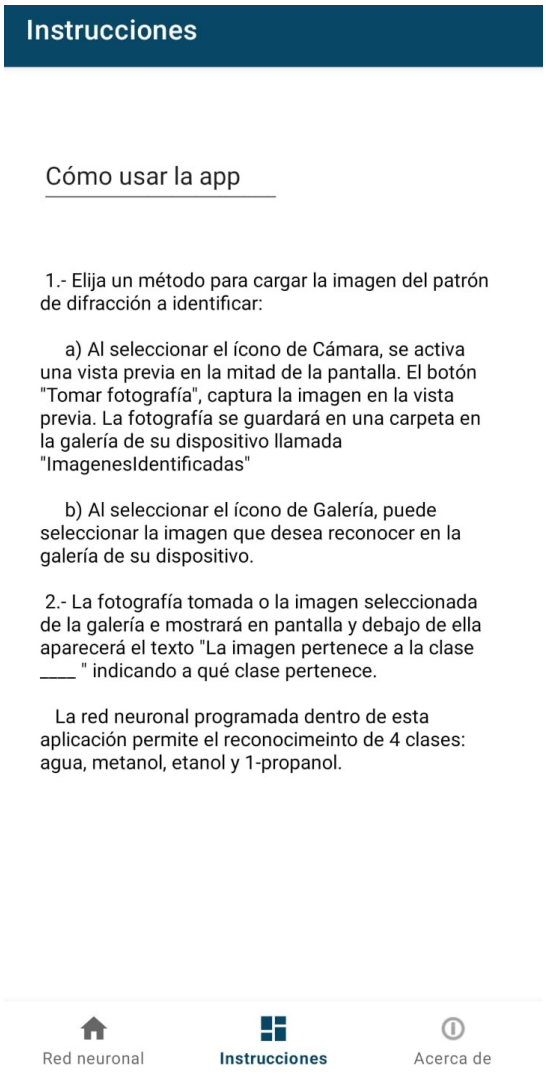


Figura 4.9: Interfaz de la aplicación.

Figura 4.10: Interfaz de la aplicación.

La aplicación generada se instaló en tres dispositivos Android diferentes para comprobar su correcto funcionamiento.

Capítulo 5

Conclusiones

Las conclusiones obtenidas a partir del desarrollo y evaluación de los diferentes modelos computacionales y bases de datos presentados en este estudio demuestran la viabilidad y las limitaciones del uso de redes neuronales artificiales para la identificación de mezclas bioquímicas a partir de patrones de difracción en el campo lejano.

Primero, la creación de una base de datos robusta utilizando imágenes capturadas con una cámara CCD y un cuidadoso pre-procesamiento permitió la obtención de un porcentaje de reconocimiento promedio de 95 % al identificar mezclas bioquímicas binarias de agua-alcohol. Este alto porcentaje de acierto pone de manifiesto la eficacia de las técnicas de pre-procesamiento, como el recorte de imágenes, la conversión a escala de grises y la aplicación de downsampling, en la reducción de la dimensionalidad del espacio de características sin perder información relevante para el entrenamiento de la red neuronal.

El desarrollo de un modelo computacional en MATLAB para entrenar una red neuronal con una sola capa oculta de neuronas sigmoides y neuronas softmax en la capa de salida también mostró ser eficiente al minimizar la entropía cruzada, lo cual es clave para mejorar la precisión en la clasificación de las sustancias bioquímicas. La importancia de la calidad de las imágenes para el entrenamiento de modelos de aprendizaje automático se destaca aquí como un factor crucial, sugiriendo que un mayor número de imágenes por clase o imágenes más enfocadas podrían mejorar significativamente la precisión.

Además, se implementó un modelo computacional de la red neuronal en lenguaje Kotlin para un sistema operativo Android, permitiendo la identificación de sustancias bioquímicas directamente desde una aplicación móvil. Para esto se construyó una base de datos conformada por imágenes de los patrones de difracción en el régimen de campo lejano adquiridos por una cámara de celular para diferentes sustancias bioquímicas: etanol, metano y 1-propanol. La aplicación demuestra una efectividad del 93 % al identificar patrones de difracción, su rendimiento podría mejorarse mediante la optimización de la captura de imágenes, como la implementación de en-

foque manual y ajustes de iluminación. Esto indica que, aunque la portabilidad de la red neuronal a dispositivos móviles es factible, existen desafíos adicionales en cuanto a la calidad de los datos de entrada en entornos menos controlados. Cabe mencionar que la aplicación tiene una interfaz de usuario amigable que permite al usuario gestionar la adquisición de imágenes de los patrones de difracción así como su identificación.

Bibliografía

Abiodun, O. I., Jantan, A., Omolara, A. E., Dada, K. V., Umar, A. M., Linus, O. U., Arshad, H., Kazaure, A. A., Gana, U., y Kiru, M. U. (2019). Comprehensive review of artificial neural network applications to pattern recognition. *IEEE access*, 7:158820–158846.

Aceves Fernández, M. (2021). *Inteligencia Artificial para Programadores con Prisa*. Amazon Digital Services LLC - KDP Print US.

Adhikari, S., Spaeth, P., Kar, A., Baaske, M. D., Khatua, S., y Orrit, M. (2020). Photothermal microscopy: Imaging the optical absorption of single nanoparticles and single molecules. *ACS Nano*, 14(12):16414–16445.

Aggarwal, C. (2018). *Neural networks and deep learning: a textbook*. Spinger.

Barlow, H. B. (1989). Unsupervised learning. *Neural computation*, 1(3):295–311.

Baxt, W. G. (1995). Application of artificial neural networks to clinical medicine. *The lancet*, 346(8983):1135–1138.

BBVA (2019). *'Machine learning': ¿qué es y cómo funciona?* `https://www.bbva.com/es/machine-learning-que-es-y-como-funciona/`.

Bedner, M., Bedner, M., Murray, J. A., Urbas, A. A., MacCrehan, W. A., y Wilson, W. B. (2021). *A comparison of measurement methods for alcohol-based hand sanitizers*. US Department of Commerce, National Institute of Standards and Technology.

Bialkowski, S. E., Astrath, N. G., y Proskurnin, M. A. (2019). *Photothermal spectroscopy methods*. John Wiley & Sons.

Bravo, F. C. y Sotelo, J. A. L. (2009). *Una aproximación práctica a las redes neuronales artificiales*. Universidad del Valle, 1 edición.

Brown, T., LeMay, H., Bursten, B., Hernández, A., Murphy, C., y Woodward, P. (2014). *Química: la ciencia central*. Colección Ciencias. Pearson Educación, 12 edición.

Cai, S., Wang, Z., Wang, S., Perdikaris, P., y Karniadakis, G. E. (2021). Physics-informed neural networks for heat transfer problems. *Journal of Heat Transfer*, 143(6):060801.

Callen, W., Huth, B., y Pantell, R. (1967). Optical patterns of thermally self-defocused light. *Applied physics letters*, 11(3):103–105.

Calvo, J. (2022). *La importancia de las funciones de activación en una red neuronal*.

Chablani, M. (2017). *Deep learning concepts — PART 1*. Towards Data Science.

Chakraborty, S., Rawat, A. K., Mishra, A. K., y Goswami, D. (2023). Quality assessment of the commercially available alcohol-based hand sanitizers with femtosecond thermal lens spectroscopy. *PeerJ Analytical Chemistry*, 5:e25.

Chen, C., Shimizu, H., y Kitamori, T. (2021). Review of ultrasensitive readout for micro-/nanofluidic devices by thermal lens microscopy. *Journal of Optical Microsystems*, 1(2).

Chhikara, P., Tekchandani, R., Kumar, N., Chamola, V., y Guizani, M. (2020). Dcnn-ga: A deep neural net architecture for navigation of uav in indoor environment. *IEEE Internet of Things Journal*, 8(6):4448–4460.

Cuomo, S., Di Cola, V. S., Giampaolo, F., Rozza, G., Raissi, M., y Piccialli, F. (2022). Scientific machine learning through physics–informed neural networks: Where we are and what's next. *Journal of Scientific Computing*, 92(3):88.

Dabby, F., Gustafson, T., Whinnery, J., Kohanzadeh, Y., y Kelley, P. (1970). thermally self-induced phase modulation of laser beams. *Applied Physics Letters*, 16(9):362–365.

Developers. (2023). *Descripción general de Kotlin*. `https://developer.android.com/kotlin/overview?hl=es-419`.

Domínguez-Juárez, J., Quiroz-Juárez, M., Aragón, J., Quintero-Bermúdez, R., y Quintero-Torres, R. (2023a). Complete numerical description of the laser-induced thermal profile in a liquid, to explain complex self-induced diffraction patterns. *Laser Physics Letters*, 20(3):036003.

Domínguez-Juárez, J. L., Quintero-Torres, R., Cardoso-Duarte, M. A., Quiroz-Juárez, M. A., Aragón, J. L., y Villatoro, J. (2023b). Unveiling the properties of liquids via photothermal-induced diffraction patterns. *Communications Physics*, 6(1):154.

Dominguez-Juarez, J. L., Vallone, S., Lempel, A., Moocarme, M., Oh, J., Gafney, H. D., y Vuong, L. T. (2015). Influence of solvent polarity on light-induced thermal cycles in plasmonic nanofluids. *Optica*, 2(5):447–453.

Escobar-Ruiz, A., Jiménez-Lara, L., Juárez-Flores, P., Montoya-Molina, F., Moreno-Sáenz, J., y Quiroz-Juárez, M. (2024). Data-driven reconstruction of chaotic dynamical equations: The hénon–heiles type system. *Chaos, Solitons & Fractals*, 184:115025.

Fadlalla, A. y Lin, C.-H. (2001). An analysis of the applications of neural networks in finance. *Interfaces*, 31(4):112–122.

Goodfellow, I., Bengio, Y., y Courville, A. (2016). *Deep learning*. MIT press.

Goodman, J. W. (2005). *Introduction to Fourier optics*. Roberts and Company publishers.

Gordon, J., Leite, R., Moore, R., Porto, S., y Whinnery, J. (1965). Long-transient effects in lasers with inserted liquid samples. *Journal of Applied Physics*, 36(1):3–8.

Greenfield, E., Rotschild, C., Szameit, A., Nemirovsky, J., El-Ganainy, R., Christodoulides, D. N., Saraf, M., Lifshitz, E., y Segev, M. (2011). Light-induced self-synchronizing flow patterns. *New Journal of Physics*, 13(5):053021.

Hagan, M., Demuth, H., Beale, M., y De Jesús, O. (2014). *Neural Network Design*. Martin Hagan.

Hammond, C. (2015). *The basics of crystallography and diffraction*, volumen 21. International Union of Crystallography texts on crystallography.

Haykin, S. (2009). *Neural networks and learning machines*. Pearson Education India, 3 edición.

Hecht, E. (2016). *Optics, Global Edition*. Pearson Education.

Hertzberg, O., Bauer, A., Küderle, A., Pleitez, M. A., y Mäntele, W. (2017). Depth-selective photothermal ir spectroscopy of skin: potential application for non-invasive glucose measurement. *Analyst*, 142:495–502.

Hodson, T. O. (2022). Root mean square error (rmse) or mean absolute error (mae): When to use them or not. *Geoscientific Model Development Discussions*, 2022:1–10.

IBM (2020). *Redes neuronales*. `https://www.ibm.com/mx-es/cloud/learn/neural-networks`.

IBM (2021). *El modelo de redes neuronales*. `https://www.ibm.com/docs/es/spss-modeler/saas?topic=networks-neural-model`.

Intel (s.f.). *Image Downsampling*. `https://www.intel.com/content/www/us/en/docs/ipp/developer-reference/2021-7/image-downsampling.html`.

Jiang, W. y Luo, J. (2022). Graph neural network for traffic forecasting: A survey. *Expert systems with applications*, 207:117921.

Kim, B. y Park, Y. H. (2018). Beginner's guide to neural networks for the mnist dataset using matlab. *Korean Journal of Mathematics*, 26(2):337–348.

Kim, P. (2017). *MATLAB Deep Learning: With Machine Learning, Neural Networks and Artificial Intelligence*. Apress.

Kitamori, T. (2019). Thermal lens microscope and microchip chemistry. *Bulletin of the Chemical Society of Japan*, 92(2):469–473.

Koech, K. E. (2020). *Cross-Entropy Loss Function*. Towards Data Science.

LeCun, Y., Bengio, Y., y Hinton, G. (2015). Deep learning. *nature*, 521(7553):436–444.

MacDonald, S. A. y Bureau, B. (2003). Fourier transform infrared attenuated total reflection and transmission spectra studied by dispersion analysis. *Applied spectroscopy*, 57(3):282–287.

Magolan, K. M. (2005). *Analysis of methanol, ethanol and propanol in aqueous environmental matrices*. Tesis doctoral, University of North Carolina at Wilmington.

Mall, P. K., Singh, P. K., Srivastav, S., Narayan, V., Paprzycki, M., Jaworska, T., y Ganzha, M. (2023). A comprehensive review of deep neural networks for medical image processing: Recent developments and future opportunities. *Healthcare Analytics*, p. 100216.

Mao, A., Mohri, M., y Zhong, Y. (2023). Cross-entropy loss functions: Theoretical analysis and applications. En *International conference on Machine learning*, pp. 23803–23828. PMLR.

Martín del Brío, B. y Sans Molina, A. (2006). *Redes neuronales y sistemas borrosos*. Textos Universitarios. Ra-Ma S.A. Editorial y Publicaciones.

Mathworks (s.f.). *Mathworks*. `https://la.mathworks.com/help/index.html?s_tid=CRUX_lftnav`.

McNelis, P. D. (2005). *Neural networks in finance: gaining predictive edge in the market*. Elsevier.

Nielsen, M. A. (2015). *Neural networks and deep learning*, volumen 25. Determination press San Francisco, CA, USA.

Penny, W. y Frost, D. (1996). Neural networks in clinical medicine. *Medical Decision Making*, 16(4):386–398.

Pleitez, M. A., Hertzberg, O., Bauer, A., Seeger, M., Lieblein, T., Lilienfeld-Toal, H. v., y MÃ¤ntele, W. (2015). Photothermal deflectometry enhanced by total internal reflection enables non-invasive glucose monitoring in human epidermis. *Analyst*, 140:483–488.

Ponce, P. (2010). *Inteligencia artificial: con aplicaciones a la ingeniería*. Alpha Editorial.

Proskurnin, M., Volkov, D., Gor'kova, T., Bendrysheva, S., Smirnova, A., y Nedosekin, D. (2015). Advances in thermal lens spectrometry. *Journal of Analytical Chemistry*, 70:249–276.

Proskurnin, M. A., Khabibullin, V. R., Usoltseva, L. O., Vyrko, E. A., Mikheev, I. V., y Volkov, D. S. (2022). Photothermal and optoacoustic spectroscopy: state of the art and prospects. *Physics-Uspekhi*, 65(3):270.

Rosenblatt, F. (1961). Principles of neurodynamics. perceptrons and the theory of brain mechanisms. Technical report, Cornell Aeronautical Lab Inc Buffalo NY.

Rumelhart, D. E., Hinton, G. E., y Williams, R. J. (1986). Learning representations by back-propagating errors. *nature*, 323(6088):533–536.

Russell, S. J. y Norvig, P. (2016). *Artificial intelligence: a modern approach*. Pearson.

Santillana Quesada, S. (2022). *Introducción a las redes neuronales para el tratamiento de imágenes*.

Serway, R. y Jewett, J. (2009). *Física Para Ciencias E Ingeniería, Volumen 2*. Física para ciencias e ingeniería. CENGAGE Learning.

Shimizu, H., Chen, C., Tsuyama, Y., Tsukahara, T., y Kitamori, T. (2022). Photothermal spectroscopy and micro/nanofluidics. *Journal of Applied Physics*, 132(6).

Sutton, R. S. y Barto, A. G. (2018). *Reinforcement learning: An introduction*. MIT press.

Tedjopurnomo, D. A., Bao, Z., Zheng, B., Choudhury, F. M., y Qin, A. K. (2020). A survey on modern deep neural network for traffic prediction: Trends, methods and challenges. *IEEE Transactions on Knowledge and Data Engineering*, 34(4):1544–1561.

Tsenang, M., Pheko-Ofitlhile, T., Mokgadi, J., Masamba, W., y Phokedi, G. N. (2023). Quantification of ethanol and identification of other chemical constituents in homemade morula beer using gas chromatography-mass spectrometry (gc-ms).

Villegas, A., Quiroz-Juárez, M. A., U'Ren, A. B., Torres, J. P., y León-Montiel, R. d. J. (2022). Identification of model particle mixtures using machine-learning-assisted laser diffraction. En *Photonics*, volumen 9, p. 74. MDPI.

Wakisaka, A. y Matsuura, K. (2006). Microheterogeneity of ethanol–water binary mixtures observed at the cluster level. *Journal of molecular liquids*, 129(1-2):25–32.

Weatherly, C. A., Woods, R. M., y Armstrong, D. W. (2014). Rapid analysis of ethanol and water in commercial products using ionic liquid capillary gas chromatography with thermal conductivity detection and/or barrier discharge ionization detection. *Journal of agricultural and food chemistry*, 62(8):1832–1838.

Capítulo 6

Anexo: Constancia del Congreso Nacional de Física

CONGRESO NACIONAL DE FÍSICA
Morelia, Michoacán, 2023

La Sociedad Mexicana de Física

Agradece la asistencia y participación de:

EVELYN ZUÑIGA CORNEJO

UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

en el **LXVI Congreso Nacional** de **Física**

Centro de Convenciones y Exposiciones del 8 al 13 de octubre de 2023

MORELIA, MICHOACÁN

Dr. Julio G. Mendoza Alvarez
PRESIDENTE

Capítulo 7

Anexo: Actividades previas en MATLAB

7.0.1. Red ADALINE

Compuerta lógica OR

Para familiarizarse con el lenguaje y el entorno de trabajo de MATLAB, así como con el funcionamiento de las redes neuronales, el primer código realizado consistió en una red neuronal ADALINE para obtener la salida de una compuerta lógica OR dados los valores de dos entradas binarias. Se estableció la matriz con los valores de entrada, incluyendo el bias, y la matriz con los valores objetivo, es decir, las salidas esperadas de la compuerta.

Con esta red neuronal se obtuvo un porcentaje de reconocimiento del 100 %. Se graficó una recta con los pesos obtenidos y los valores de las entradas, las combinaciones de uno y cero, y se observó como esta red clasifica las entradas y devuelve un uno cuando se tiene al menos un uno en una de las entradas (7.1). El código empleado se encuentra en el anexo 8.1.

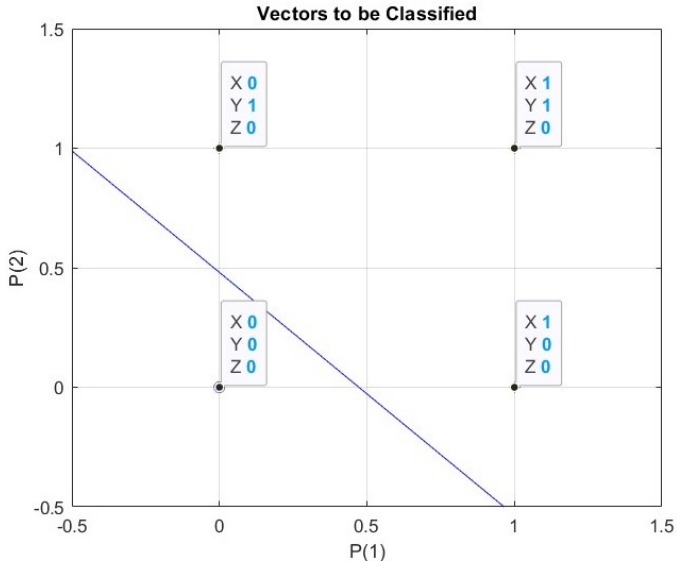
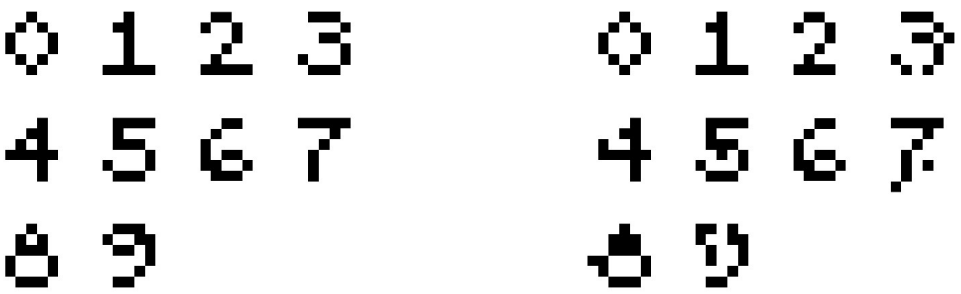


Figura 7.1: Representación gráfica de la clasificación que realiza la red ADALINE para la compuerta lógica OR. Las coordenadas X y Y representan las dos entradas que lee la red.

Reconocimiento de números

Para familiarizarse con el manejo de matrices el siguiente código a realizar fue una red ADALINE para la identificación de imágenes con números del 0 al 9. Las imágenes se generaron utilizando matrices de 9x9, colocando ceros (píxel en blanco) y unos (píxel en negro) para generar el número deseado como se muestra en la figura 7.2a . Se utilizó la función de activación *hardlim*, se utilizaron 1000 épocas para el entrenamiento y un tamaño de paso de 5×10^{-6}. Con esto se obtuvo un porcentaje de reconocimiento del 100 %. Posteriormente se realizaron pruebas a la red generando ruido en las imágenes de los números mediante la función *imnoise* (figura 7.2b. La red neuronal presentó un reconocimiento del 70 % para estas imágenes. El código generado para el reconociemiento se encuentra en el anexo 8.2.



(a) Números originales

(b) Números con ruido

Figura 7.2: Números generados para el reconocimiento de patrones utilizando una red ADALINE.

7.0.2. Implementación del algoritmo Backpropagation para el reconocimiento de patrones numéricos

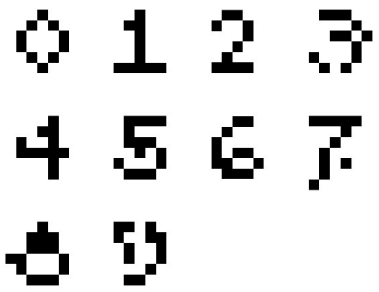
Para comprobar el funcionamiento del algoritmo Bakcpropagation desarrollado, se programó un perceptrón multicapa. Para esta red neuronal se utilizó una única capa oculta con 15 neuronas y la arquitectura de la red *patternet*, es decir:

- Función de costo: entropía cruzada
- Función de salida de la capa oculta: función sigmoide.
- Función de salida de la capa de salida: función softmax.

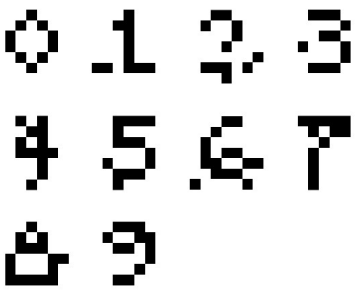
Como conjunto de entrenamiento se utilizaron las matrices de 9x9 generadas para el reconocimiento de números y como conjunto de prueba se utilizaron las matrices de 9x9 de números con ruido. Para programar esta red neuronal se utilizaron las ecuaciones 3.28 y 3.31.

Para elegir el número de neuronas en la capa oculta se realizaron pruebas con 10, 15 y 20 neuronas. Así mismo, se probaron diferentes tazas de aprendizaje. Con 100 épocas, 20 neuronas y una taza de aprendizaje de 0.01 se obtuvo un porcentaje de reconocimiento del 100 % para el conjunto de entrenamiento. Se probó esta red con el conjunto de prueba (figura 7.3b) y se obtuvo un porcentaje de reconocimiento del 90 %).

Se observó que el perceptrón multicapa con el algoritmo backpropagation presenta el mismo porcentaje de reconocimiento que la red ADALINE, sin embargo, las imágenes utilizadas para probar el perceptrón multicapa presentan mayor ruido (Figura 7.3). Esto nos demuestra la mayor capacidad que presenta un perceptrón multicapa contra una red ADALINE para el reconocmiento de patrones. El código empleado para el percentrón multicapa se encuentra en el anexo 8.6.



(a) Números empleados para la red ADALINE



(b) Números empleados para el Perceptrón multicapa

Figura 7.3: Números con ruidos generados para el reconocimiento de patrones .

Capítulo 8

Anexo: Códigos utilizados

8.1. Compuerta lógica OR

```
clear,clc

%Datos:
In=[1 0 0;1 0 1;1 1 0;1 1 1]; %Entradas: b, x1, x2
Out=[0;1;1;1]; % Salida de la compuerta OR
%W=rand(1,3); %Pesos iniciales aleatorios
W = [1 1 1]; %Pesos iniciales
epsilon=0.3; %Tolerancia
alfa=0.1; %Taza de aprendizaje

ECM = 0; %Error cuadratico medio
[f,c]=size(In); %f=filas y c=columnas (tama o de In)
h=100; % pocas

%Adaline
for p=1:h %Para cada  poca
    EMC = 0;
    for n=1:f %Para cada fila

        y=W*In(n,:)'; %Calculamos z(n) para la fila n,
           transponemos In para que se puedan multiplicar
        Er=Out(n)-y; %Error
        W=W+(alfa*Er*In(n,:)); %Ajuste de pesos
        ECM=ECM+(Er^2);
    end

```

```
26      ECM=ECM/f;
27      MSB(p)=ECM;
28      if MSB(p)<epsilon
29          break
30      end
31  end
32
33  Epoca=1:p;
34  figure;
35  plot(Epoca,MSB,'*-r')
36  xlabel('Epoca')
37  ylabel('Error Cuadratico Medio')
38  disp(p)
39
40  for j=1:f
41      y=W*In(j,:)';
42      if y> 0.5
43          Y(j,1)=1;
44      else
45          Y(j,1)=0;
46      end
47  end
48
49  u=0;
50  for q=1:f
51      if Y(q,1)==Out(q)
52          u=u+1;
53      else
54          der=1;
55      end
56  end
57  por=(u*100)/f; %Porcentaje de reconociemiento
58  disp(['El porcentaje de reconocimiento es del ', num2str(por),'%'
        ]);
59  disp(W);
60
61  figure;
62  InR =[0 0;0 1;1 0;1 1];
63  plotpv(InR',Out');
64  grid on;
```

```
We=[-W(2) -W(3)];
plotpc(We,W(1));
```

8.2. Reconocimiento de números

```
clear,clc
%DATOS
R=0.06;
uno=[1 1 1 0 1 1 1;1 1 0 0 1 1 1;1 1 1 0 1 1 1;1 1 1 0 1 1 1;1 1 1 0 1 1 1;1 0 0 0 0 0 1;1 1 1 1 1 1 1];
dos=[1 1 0 0 1 1 1;1 0 1 1 0 1 1;1 1 1 1 0 1 1;1 1 1 0 1 1 1;1 1 0 1 1 1 1;1 0 0 0 0 0 1;1 1 1 1 1 1 1];
trs=[1 1 0 0 0 1 1;1 1 1 1 1 0 1;1 1 1 0 0 1 1;1 1 1 1 1 0 1;1 0 1 1 1 0 1;1 1 0 0 0 1 1;1 1 1 1 1 1 1];
cua=[1 1 1 1 0 1 1;1 1 1 0 0 1 1;1 1 0 1 0 1 1;1 0 0 0 0 0 1;1 1 1 1 0 1 1;1 1 1 1 1 1 1];
cin=[1 1 0 0 0 0 1;1 1 0 1 1 1 1;1 1 0 0 0 1 1;1 1 1 1 1 0 1;1 0 1 1 1 0 1;1 1 0 0 0 1 1;1 1 1 1 1 1 1];
six=[1 1 1 0 0 1 1;1 1 0 1 1 1 1;1 0 1 1 1 1 1;1 0 1 0 0 1 1;1 0 0 1 1 0 1;1 1 0 0 0 1 1;1 1 1 1 1 1 1];
sev=[1 0 0 0 0 0 1;1 1 1 1 0 1 1;1 1 1 0 1 1 1;1 1 0 1 1 1 1;1 1 0 1 1 1 1;1 1 0 1 1 1 1;1 1 1 1 1 1 1];
och=[1 1 1 0 1 1 1;1 1 0 1 0 1 1;1 1 0 0 0 1 1;1 0 1 1 1 0 1;1 0 1 1 1 0 1;1 1 0 0 0 1 1;1 1 1 1 1 1 1];
nue=[1 1 0 0 0 1 1;1 0 1 1 0 0 1;1 1 0 0 1 0 1;1 1 1 1 1 0 1;1 1 1 1 0 1 1;1 1 0 0 1 1 1;1 1 1 1 1 1 1];
cro=[1 1 1 0 1 1 1;1 1 0 1 0 1 1;1 0 1 1 1 0 1;1 0 1 1 1 0 1;1 1 0 1 0 1 1;1 1 1 0 1 1 1;1 1 1 1 1 1 1];
subplot(3,4,1);imshow(cro);
subplot(3,4,2);imshow(uno);
subplot(3,4,3);imshow(dos);
subplot(3,4,4);imshow(trs);
subplot(3,4,5);imshow(cua);
subplot(3,4,6);imshow(cin);
subplot(3,4,7);imshow(six);
subplot(3,4,8);imshow(sev);
subplot(3,4,9);imshow(och);
subplot(3,4,10);imshow(nue);

```

```
25 patron=[[cro(1,:),cro(2,:),cro(3,:),cro(4,:),cro(5,:),cro(6,:),cro
       (7,:)];
26         [uno(1,:),uno(2,:),uno(3,:),uno(4,:),uno(5,:),uno(6,:),uno
             (7,:)];
27         [dos(1,:),dos(2,:),dos(3,:),dos(4,:),dos(5,:),dos(6,:),dos
             (7,:)];
28         [trs(1,:),trs(2,:),trs(3,:),trs(4,:),trs(5,:),trs(6,:),trs
             (7,:)];
29         [cua(1,:),cua(2,:),cua(3,:),cua(4,:),cua(5,:),cua(6,:),cua
             (7,:)];
30         [cin(1,:),cin(2,:),cin(3,:),cin(4,:),cin(5,:),cin(6,:),cin
             (7,:)];
31         [six(1,:),six(2,:),six(3,:),six(4,:),six(5,:),six(6,:),six
             (7,:)];
32         [sev(1,:),sev(2,:),sev(3,:),sev(4,:),sev(5,:),sev(6,:),sev
             (7,:)];
33         [och(1,:),och(2,:),och(3,:),och(4,:),och(5,:),och(6,:),och
             (7,:)];
34         [nue(1,:),nue(2,:),nue(3,:),nue(4,:),nue(5,:),nue(6,:),nue
             (7,:)]];
35  %Entradas con ruido
36   R0=imnoise(cro,'salt & pepper',R);
37   R1=imnoise(uno,'salt & pepper',R);
38   R2=imnoise(dos,'salt & pepper',R);
39   R3=imnoise(trs,'salt & pepper',R);
40   R4=imnoise(cua,'salt & pepper',R);
41   R5=imnoise(cin,'salt & pepper',R);
42   R6=imnoise(six,'salt & pepper',R);
43   R7=imnoise(sev,'salt & pepper',R);
44   R8=imnoise(och,'salt & pepper',R);
45   R9=imnoise(nue,'salt & pepper',R);
46
47 patru=[1,R0(1,:),R0(2,:),R0(3,:),R0(4,:),R0(5,:),R0(6,:),R0(7,:);
48        1,R1(1,:),R1(2,:),R1(3,:),R1(4,:),R1(5,:),R1(6,:),R1(7,:);
49        1,R2(1,:),R2(2,:),R2(3,:),R2(4,:),R2(5,:),R2(6,:),R2(7,:);
50        1,R3(1,:),R3(2,:),R3(3,:),R3(4,:),R3(5,:),R3(6,:),R3(7,:);
51        1,R4(1,:),R4(2,:),R4(3,:),R4(4,:),R4(5,:),R4(6,:),R4(7,:);
52        1,R5(1,:),R5(2,:),R5(3,:),R5(4,:),R5(5,:),R5(6,:),R5(7,:);
53        1,R6(1,:),R6(2,:),R6(3,:),R6(4,:),R6(5,:),R6(6,:),R6(7,:);
54        1,R7(1,:),R7(2,:),R7(3,:),R7(4,:),R7(5,:),R7(6,:),R7(7,:);
```

```
55          1,R8(1,:),R8(2,:),R8(3,:),R8(4,:),R8(5,:),R8(6,:),R8(7,:);
56          1,R9(1,:),R9(2,:),R9(3,:),R9(4,:),R9(5,:),R9(6,:),R9(7,:)];
57
58       figure(2);
59  subplot(3,4,1),imshow(R0)
60  subplot(3,4,2),imshow(R1)
61  subplot(3,4,3),imshow(R2)
62  subplot(3,4,4),imshow(R3)
63  subplot(3,4,5),imshow(R4)
64  subplot(3,4,6),imshow(R5)
65  subplot(3,4,7),imshow(R6)
66  subplot(3,4,8),imshow(R7)
67  subplot(3,4,9),imshow(R8)
68  subplot(3,4,10),imshow(R9)
69
70  salida=[1 0 0 0 0 0 0 0 0 0;0 1 0 0 0 0 0 0 0 0;0 0 1 0 0 0 0 0 0
       0;0 0 0 1 0 0 0 0 0 0;0 0 0 0 1 0 0 0 0 0;
71          0 0 0 0 0 1 0 0 0 0;0 0 0 0 0 0 1 0 0 0;0 0 0 0 0 0 0 1 0
              0;0 0 0 0 0 0 0 0 1 0;0 0 0 0 0 0 0 0 0 1];
72
73
74  W=ones(50,10);                          %Pesos iniciales
75  epsilon=0.000005;
76  alfa=0.035;
77
78  %Acondicionamiento
79  %[f c]=size(E);
80  u=0;
81  h=1000;
82
83  %Adaline
84  for p=1:h
85      for n=1:10
86          X=[1,patron(n,:)];
87          y=X*W;
88          ER=salida(n,:)-y;
89          W=W+(alfa*X'*ER);
90          ECM(n)=(ER*ER');
91      end
92
```

```
93      E(p)=(sum(ECM))/10;
94
95      if E(p)<=epsilon
96          break
97      end
98  end
99
100 Epoca=1:p;
101 figure;
102 plot(Epoca,E,'*-r')
103 xlabel('Epoca')
104 ylabel('Error Cuadratico Medio')
105
106
107 for j=1:10
108     y=[1,patron(j,:)]*W;
109     Y=hardlim(y);
110     if salida(j,:)==Y
111         u=u+1;
112     end
113 end
114 por=(u*100)/10;
115 disp(['El porcentaje de reconocimiento en etapa de Entrenamiento '
        , num2str(por),'%']);
116
117 u=0;
118 for j=1:10
119    y=patru(j,:)*W;
120      Y=hardlim(y);
121    if salida(j,:)==Y
122        u=u+1;
123        disp(['El numero:', num2str(j),' SI']);
124    else
125        disp(['El numero:', num2str(j),' NO']);
126    end
127 end
128 por=(u*100)/10;
129 disp(['El porcentaje de reconocimiento en etapa de Prueba ',
       num2str(por),'%']);
```

8.3. Red patternnet

```
%% Pruebas para encontrar los hiperpar metros opitmos
perf=[];
k=1;
for j=[0.01 0.005]
    l=1;
    for i = [1 2 3 4 5]
        Net = patternnet(class*i, 'traingd');
        Net.trainParam.lr = j;
        Net.trainParam.epochs = 5000;
        Net.divideFcn = 'dividerand';  % Divide data randomly
        Net.divideMode = 'sample';  % Divide up every sample
        Net.divideParam.trainRatio = 70/100;
        Net.divideParam.valRatio = 15/100;
        Net.divideParam.testRatio = 15/100;

        [NetTrained,tr] = train(Net,All_Images,Target);
        disp(tr);
        perf(k,l)=tr.best_tperf;
        l=l+1;
    end
    k=k+1;
end

%% Red con los hiperparametros elegidos

Net = patternnet(class*5, 'traingd');
Net.trainParam.lr = 0.005;
Net.trainParam.epochs = 5000;
Net.divideFcn = 'dividerand';  % Divide data randomly
Net.divideMode = 'sample';  % Divide up every sample
Net.divideParam.trainRatio = 70/100;
Net.divideParam.valRatio = 15/100;
Net.divideParam.testRatio = 15/100;

[NetTrained,tr] = train(Net,All_Images,Target);
disp(tr);
```

8.4. Downsampling

```
function [FieldD]=DownSampling_v2(FieldFinal,SquareSize)
[OutPixel,OutPixel2]=size(FieldFinal);
res=mod(OutPixel,SquareSize);
Limit=(OutPixel-res)/SquareSize;

res2=mod(OutPixel2,SquareSize);
Limit2=(OutPixel2-res2)/SquareSize;

    %Down sampling
    for m=1:Limit
        for p=1:Limit2
            ini_m=SquareSize*(m-1)+1;
            fin_m=SquareSize*(m);
            ini_n=SquareSize*(p-1)+1;
            fin_n=SquareSize*(p);
            Im=FieldFinal(ini_m:fin_m,ini_n:fin_n);
            FieldD(m,p)=sum(sum(Im))/(SquareSize^2);
        end
    end

end
```

8.5. Pre-procesamiento

```
clear,clc
DownDiv=20;  %Pixeles/DownDiv= No. de pixeles para Downsampling
All_Images =[];  %Matriz para todas las im genes
Num_images=[]; %N mero de imagenes en c/folder
imgType='*.jpg'; %Tipo de im gen

%Folder con los subfolders
topLevelFolder = fullfile('D:\Tesis\Proyecto');
figure()
plot(Entropia)
% Lista de todos los subfolders.
allSubFolders = genpath(topLevelFolder);
```

```
13
14 %Obtenemos el nombre de cada folder
15 remain = allSubFolders;
16 listOfFolderNames = {};
17 while true
18   %Separamos cada folder con ';'
19   [singleSubFolder, remain] = strtok(remain, ';');
20   if isempty(singleSubFolder)
21     break; %Cuando se acaben los folders se rompe el ciclo
22   end
23   %matriz con los nombres de cada folder, incluyendo el principal
24   listOfFolderNames = [listOfFolderNames singleSubFolder];
25 end
26
27 %No. de subfolders = no de clases
28 numberOfFolders = length(listOfFolderNames);
29 class=numberOfFolders-1;
30
31 %Para cada subfolder...
32 for l = 2 :numberOfFolders
33   thisFolder = listOfFolderNames{l};
34   imgPath =strcat(thisFolder,'\');
35   images  = dir([imgPath imgType]); %Ruta del folder
36   [ren,col]=size(images); %No de im genes y  de caracter sticas
37   Num_images(l-1)=ren;
38
39   if not(l == 2)
40       Data=zeros(ren1*col1,ren);
41   end
42
43   %Leemos cada imagen del subfolder actual
44   for k=1:ren
45     disp([l-1,k,ren]) %Mostramos en qu  no. de folder e imagen va
         el c digo
46
47     %Extraemos los valores de la concentracion
48     aux=str2double(regexp(images(k).name,'\d+[\.]?\d*','match'));
49     C(k)=aux(end);
50
51     %Leer imagen
```

```
52      NameImages=[imgPath,images(k).name];
53      I=imread(NameImages);
54      Cut=I(1330:2330,2030:3030); %Recortamos las im genes
55      Im = im2gray(Cut); %Escala de grises
56      Id=double(Im);
57
58      %Downsampling
59      I_down=DownSampling_v2(Im,DownDiv);
60      [ren1,col1]=size(I_down);
61
62
63      if k==20
64          figure; imshow(I)
65          figure; imshow(Im)
66          figure;imagesc(Id)
67          figure;imagesc(I_down)
68      end
69
70      %Juntamos todos los pixeles de una imagen en una sola columna
71      Data(:,k)=reshape(I_down,[ren1*col1,1]);
72    end
73
74    %Unimos las im genes de cada folder
75    All_Images=cat(2,All_Images,Data);
76
77  end
78
79  %Creamos la matriz Target
80  [ren_all,col_all]=size(All_Images);
81  Target=zeros(class,col_all);
82  s=1;
83  for i=1:class
84      n=Num_images(i);
85      Target(i,s:n+s-1)=1;
86      s=n+s;
87  end
```

8.6. Reconocimiento de números utilizando Backpropagation

```
%%Entrenamiento
%Datos
clear,clc

uno=[1 1 1 0 1 1 1;1 1 0 0 1 1 1;1 1 1 0 1 1 1;1 1 1 0 1 1 1;1 1 1
    0 1 1 1;1 0 0 0 0 0 1;1 1 1 1 1 1 1];
dos=[1 1 0 0 1 1 1;1 0 1 1 0 1 1;1 1 1 1 0 1 1;1 1 1 0 1 1 1;1 1 0
    1 1 1 1;1 0 0 0 0 0 1;1 1 1 1 1 1 1];
trs=[1 1 0 0 0 1 1;1 1 1 1 1 0 1;1 1 1 0 0 1 1;1 1 1 1 1 0 1;1 0 1
    1 1 0 1;1 1 0 0 0 1 1;1 1 1 1 1 1 1];
cua=[1 1 1 1 0 1 1;1 1 1 0 0 1 1;1 1 0 1 0 1 1;1 0 0 0 0 0 1;1 1 1
    1 0 1 1;1 1 1 1 0 1 1;1 1 1 1 1 1 1];
cin=[1 1 0 0 0 0 1;1 1 0 1 1 1 1;1 1 0 0 0 1 1;1 1 1 1 1 0 1;1 0 1
    1 1 0 1;1 1 0 0 0 1 1;1 1 1 1 1 1 1];
six=[1 1 1 0 0 1 1;1 1 0 1 1 1 1;1 0 1 1 1 1 1;1 0 1 0 0 1 1;1 0 0
    1 1 0 1;1 1 0 0 0 1 1;1 1 1 1 1 1 1];
sev=[1 0 0 0 0 0 1;1 1 1 1 0 1 1;1 1 1 0 1 1 1;1 1 0 1 1 1 1;1 1 0
    1 1 1 1;1 1 0 1 1 1 1;1 1 1 1 1 1 1];
och=[1 1 1 0 1 1 1;1 1 0 1 0 1 1;1 1 0 0 0 1 1;1 0 1 1 1 0 1;1 0 1
    1 1 0 1;1 1 0 0 0 1 1;1 1 1 1 1 1 1];
nue=[1 1 0 0 0 1 1;1 0 1 1 0 0 1;1 1 0 0 1 0 1;1 1 1 1 1 0 1;1 1 1
    1 0 1 1;1 1 0 0 1 1 1;1 1 1 1 1 1 1];
cro=[1 1 1 0 1 1 1;1 1 0 1 0 1 1;1 0 1 1 1 0 1;1 0 1 1 1 0 1;1 1 0
    1 0 1 1;1 1 1 0 1 1 1;1 1 1 1 1 1 1];
subplot(3,4,1);imshow(cro);
subplot(3,4,2);imshow(uno);
subplot(3,4,3);imshow(dos);
subplot(3,4,4);imshow(trs);
subplot(3,4,5);imshow(cua);
subplot(3,4,6);imshow(cin);
subplot(3,4,7);imshow(six);
subplot(3,4,8);imshow(sev);
subplot(3,4,9);imshow(och);
subplot(3,4,10);imshow(nue);

patron=[[cro(1,:),cro(2,:),cro(3,:),cro(4,:),cro(5,:),cro(6,:),cro
   (7,:)];
```

```
27          [uno(1,:),uno(2,:),uno(3,:),uno(4,:),uno(5,:),uno(6,:),uno
               (7,:)];
28          [dos(1,:),dos(2,:),dos(3,:),dos(4,:),dos(5,:),dos(6,:),dos
               (7,:)];
29          [trs(1,:),trs(2,:),trs(3,:),trs(4,:),trs(5,:),trs(6,:),trs
               (7,:)];
30          [cua(1,:),cua(2,:),cua(3,:),cua(4,:),cua(5,:),cua(6,:),cua
               (7,:)];
31          [cin(1,:),cin(2,:),cin(3,:),cin(4,:),cin(5,:),cin(6,:),cin
               (7,:)];
32          [six(1,:),six(2,:),six(3,:),six(4,:),six(5,:),six(6,:),six
               (7,:)];
33          [sev(1,:),sev(2,:),sev(3,:),sev(4,:),sev(5,:),sev(6,:),sev
               (7,:)];
34          [och(1,:),och(2,:),och(3,:),och(4,:),och(5,:),och(6,:),och
               (7,:)];
35          [nue(1,:),nue(2,:),nue(3,:),nue(4,:),nue(5,:),nue(6,:),nue
               (7,:)]];
36
37  salid=[1 0 0 0 0 0 0 0 0 0;0 1 0 0 0 0 0 0 0 0;0 0 1 0 0 0 0 0 0 0
       0;0 0 0 1 0 0 0 0 0 0;0 0 0 0 1 0 0 0 0 0;
38          0 0 0 0 0 1 0 0 0 0;0 0 0 0 0 0 1 0 0 0;0 0 0 0 0 0 0 1 0
               0;0 0 0 0 0 0 0 0 1 0;0 0 0 0 0 0 0 0 0 1];
39
40  Input=patron';
41  Target=salid';
42  epsilon=1e-7;
43
44
45  %% Training
46  %Hyperparameters
47  gamma=0.01;
48  NeuronHiddenLayer=20;
49  epoch = 100;
50  [ren,col]=size(Input);
51  [ren1,col1]=size(Target);
52
53  W_Lm1=(2*(rand(NeuronHiddenLayer,ren)-0.5));
54  W_L=(2*(rand(ren1,NeuronHiddenLayer)-0.5));
55
```

```
WLm1=W_Lm1;
WL=W_L;

%Train
j=1;
 for k = 1:epoch
    for n=1:col
        Z_Lm1=W_Lm1*Input(:,n);
        A_Lm1=sigmoide(Z_Lm1);
        Z_L=W_L*A_Lm1;
        A_L=salida(Z_L);

        Delta_L=A_L-Target(:,n);
        aux=transpose(W_L)*Delta_L;
        Delta_Lm1=aux.*A_Lm1.*(1-A_Lm1);

        W_L=W_L-gamma*Delta_L*transpose(A_Lm1);
        W_Lm1=W_Lm1-gamma*Delta_Lm1*transpose(Input(:,n));

        Entropia(j)= -(1/length(A_L))*sum(Target(:,n).*log(A_L)
           +(1-Target(:,n)).*log(1-A_L));
        j = j+1;
    end
    E(k)=sum(Entropia)/length(Entropia);
    if E(k)<=epsilon
       break
    end
 end
figure()
plot(Entropia)
disp(Entropia(j-1))

u=0;

for n=1:col
    Z_Lm1=W_Lm1*Input(:,n);
    A_Lm1=sigmoide(Z_Lm1);
    Z_L=W_L*A_Lm1;
    A_L=salida(Z_L);
    [M,I] = max(A_L);
```

```
95      if Target(I,n)==1
96          u = u+1;
97      end
98  end
99
100 por=(u*100)/10;
101 disp(['El porcentaje de reconocimiento en etapa de Entrenamiento '
        , num2str(por),'%']);
102
103
104 %% Test
105
106 R=0.09;
107 %Entradas con ruido
108   R0=imnoise(cro,'salt & pepper',R);
109   R1=imnoise(uno,'salt & pepper',R);
110   R2=imnoise(dos,'salt & pepper',R);
111   R3=imnoise(trs,'salt & pepper',R);
112   R4=imnoise(cua,'salt & pepper',R);
113   R5=imnoise(cin,'salt & pepper',R);
114   R6=imnoise(six,'salt & pepper',R);
115   R7=imnoise(sev,'salt & pepper',R);
116   R8=imnoise(och,'salt & pepper',R);
117   R9=imnoise(nue,'salt & pepper',R);
118
119   patru=[R0(1,:),R0(2,:),R0(3,:),R0(4,:),R0(5,:),R0(6,:),R0(7,:);
120          R1(1,:),R1(2,:),R1(3,:),R1(4,:),R1(5,:),R1(6,:),R1(7,:);
121          R2(1,:),R2(2,:),R2(3,:),R2(4,:),R2(5,:),R2(6,:),R2(7,:);
122          R3(1,:),R3(2,:),R3(3,:),R3(4,:),R3(5,:),R3(6,:),R3(7,:);
123          R4(1,:),R4(2,:),R4(3,:),R4(4,:),R4(5,:),R4(6,:),R4(7,:);
124          R5(1,:),R5(2,:),R5(3,:),R5(4,:),R5(5,:),R5(6,:),R5(7,:);
125          R6(1,:),R6(2,:),R6(3,:),R6(4,:),R6(5,:),R6(6,:),R6(7,:);
126          R7(1,:),R7(2,:),R7(3,:),R7(4,:),R7(5,:),R7(6,:),R7(7,:);
127          R8(1,:),R8(2,:),R8(3,:),R8(4,:),R8(5,:),R8(6,:),R8(7,:);
128          R9(1,:),R9(2,:),R9(3,:),R9(4,:),R9(5,:),R9(6,:),R9(7,:)];
129
130       figure(2);
131   subplot(3,4,1),imshow(R0)
132   subplot(3,4,2),imshow(R1)
133   subplot(3,4,3),imshow(R2)
```

```
134   subplot(3,4,4),imshow(R3)
135   subplot(3,4,5),imshow(R4)
136   subplot(3,4,6),imshow(R5)
137   subplot(3,4,7),imshow(R6)
138   subplot(3,4,8),imshow(R7)
139   subplot(3,4,9),imshow(R8)
140   subplot(3,4,10),imshow(R9)
141
142 Input2=patru';
143
144 u=0;
145
146 for n=1:col
147     Z_Lm1=W_Lm1*Input2(:,n);
148     A_Lm1=sigmoide(Z_Lm1);
149     Z_L=W_L*A_Lm1;
150     A_L=salida(Z_L);
151     [M,I] = max(A_L);
152     if Target(I,n)==1
153         u = u+1;
154     end
155 end
156
157 por=(u*100)/10;
158 disp(['El porcentaje de reconocimiento en etapa de Prueba ',
       num2str(por),'%']);
159
160 %% Funciones
161 function f=sigmoide(h)
162     f=1./(1+exp(-h));
163 end
164
165 function f=salida(h)
166     sum_exp=sum(exp(h));
167     f=exp(h)./sum_exp;
168 end
```

8.7. Red neuronal para el reconocimiento de mezclas bioquímicas binarias

```
%% Separaci n de datos
%s = RandStream('mlfg6331_64');
[Im_Train,idx_train] = datasample(All_Images,round(col_all*0.7),2,
   'Replace',false);
Tar_Train= Target(:,idx_train);

Tar_Test=Target;
Tar_Test(:,idx_train )=[];

Im_Test=All_Images;
Im_Test(:,idx_train )=[];

%% Test 7

%Hyperparameters
epsilon=0.005;        %
gamma=0.00005;           %Taza de aprendizaje
NeuronHiddenLayer=35;
epoch = 3000;
Input1=Im_Train;
Target1=Tar_Train;
[ren,col]=size(Input1);
[ren1,col1]=size(Target1);

W_Lm1=(2*(rand(NeuronHiddenLayer,ren)-0.5));
W_L=(2*(rand(ren1,NeuronHiddenLayer)-0.5));

WLm1=W_Lm1;
WL=W_L;

%Train
j=1;
 for k = 1:epoch
    for n=1:col
        Z_Lm1=W_Lm1*Input1(:,n);
        A_Lm1=sigmoide(Z_Lm1);
```

```
        Z_L=W_L*A_Lm1;
        A_L=salida(Z_L);

        Delta_L=A_L-Target1(:,n);
        aux=transpose(W_L)*Delta_L;
        Delta_Lm1=aux.*A_Lm1.*(1-A_Lm1);

        W_L=W_L-gamma*Delta_L*transpose(A_Lm1);
        W_Lm1=W_Lm1-gamma*Delta_Lm1*transpose(Input1(:,n));

        Entropia(j)= -(1/length(A_L))*sum(Target(:,n).*log(A_L)
           +(1-Target(:,n)).*log(1-A_L));
        j = j+1;
    end
    E(k)=sum(Entropia)/length(Entropia);
    if E(k)<=epsilon
      break
    end
    disp([k, E(k)]);
 end
figure()
plot(E)
   %%
Input2=Im_Test;
Target2=Tar_Test;
[ren2,col2]=size(Input2);

 u=0;
 class_t=zeros(5,1);
 class_o=zeros(5,1);
 class_p=zeros(5,1);

 for n=1:col2
     Z_Lm1=W_Lm1*Input2(:,n);
     A_Lm1=sigmoide(Z_Lm1);
     Z_L=W_L*A_Lm1;
     A_L=salida(Z_L);
     [M,I] = max(A_L);
     if Target2(I,n)==1
         u = u+1;
```

```
75          class_t(I,1)=class_t(I,1)+1;
76      end
77
78      for m=1:5
79          class_o(I,1)=Target2(m,n)+class_o(I,1);
80      end
81
82  end
83  por=(u*100)/n;
84  disp(por);
85  class_p=(class_t./class_o)*100;
86 disp(class_p);
87
88 %Perdida
89 %% Functions
90 function f=sigmoide(h)
91     f=1./(1+exp(-h));
92 end
93
94 function f=salida(h)
95     sum_exp=sum(exp(h));
96     f=exp(h)./sum_exp;
97 end
```

8.8. Pre-procesamiento Kotlin

```
1 private fun handleImageSelection(imageUri: Uri) {
2     GlobalScope.launch(Dispatchers.IO) {
3         // Obtener el Bitmap desde la URI
4         val inputStream = contentResolver.openInputStream(imageUri
              )
5         val originalBitmap = BitmapFactory.decodeStream(
              inputStream)
6         var grayscaleBitmap: Bitmap // Variable para almacenar el
              bitmap en escala de grises
7         var downsampledBitmap: Bitmap // Variable para almacenar
              el bitmap downsampleado
8         var procesedBitmap: Bitmap
9
```

```
10          // Convertir a blanco y negro
11          grayscaleBitmap = convertToGrayscale(originalBitmap)
12          if (grayscaleBitmap.height == 4000 &&  grayscaleBitmap.
               width==6000){
13              procesedBitmap =Bitmap.createBitmap(grayscaleBitmap,
                   2030,  1330, 1000, 1000)
14              downsampledBitmap = downsamplingGrayscaleBitmap(
                   procesedBitmap, 50)
15              images.add(downsampledBitmap)
16              val tam= images.size
17              println("imagenes $tam")
18          } else if (grayscaleBitmap.height == 720 &&
               grayscaleBitmap.width==1280){
19              procesedBitmap =Bitmap.createBitmap(grayscaleBitmap,
                   290,  20, 700, 700)
20              downsampledBitmap = downsamplingGrayscaleBitmap(
                   procesedBitmap, 14)
21              images.add(downsampledBitmap)
22              val tam= images.size
23              println("imagenes $tam")
24          } else if (grayscaleBitmap.height == 1088 &&
               grayscaleBitmap.width==1920){
25              procesedBitmap =Bitmap.createBitmap(grayscaleBitmap,
                   750,  300, 700, 700)
26              downsampledBitmap = downsamplingGrayscaleBitmap(
                   procesedBitmap, 14)
27              images.add(downsampledBitmap)
28              val tam= images.size
29              println("imagenes $tam")
30          } else{
31              runOnUiThread {
32                  binding.TextView.text = "Tama o de imagen no
                       configurado para procesamiento"
33              }
34          }
35
36
37          // Limpieza de recursos
38          inputStream?.close()
39      }
```

```
40 }
41
42
43 private fun convertToGrayscale(originalBitmap: Bitmap): Bitmap {
44     val width = originalBitmap.width
45     val height = originalBitmap.height
46
47     val grayscaleBitmap = Bitmap.createBitmap(width, height,
       Bitmap.Config.ARGB_8888)
48     val canvas = Canvas(grayscaleBitmap)
49     val paint = Paint()
50
51     val colorMatrix = ColorMatrix().apply {
52         setSaturation(0f) // 0 para escala de grises
53     }
54
55     val colorFilter = ColorMatrixColorFilter(colorMatrix)
56     paint.colorFilter = colorFilter
57
58     canvas.drawBitmap(originalBitmap, 0f, 0f, paint)
59
60     return grayscaleBitmap
61 }
62
63 private fun saveBitmapToStorage(bitmap: Bitmap) {
64     val picturesDirectory = Environment.
       getExternalStoragePublicDirectory(Environment.
       DIRECTORY_PICTURES)
65     val directory = File(picturesDirectory, "ImagenesPreProcesadas
       ")
66     if (!directory.exists()) {
67         directory.mkdirs()
68     }
69
70     val fileName = "grayscale_${System.currentTimeMillis()}.png"
71     val file = File(directory, fileName)
72
73     try {
74         val fileOutputStream = FileOutputStream(file)
```

```
75          bitmap.compress(Bitmap.CompressFormat.PNG, 100,
               fileOutputStream)
76          fileOutputStream.flush()
77          fileOutputStream.close()
78
79          //runOnUiThread {
80          //    binding.TextView.text = "Bitmap guardado en $
               directory"
81          //}
82
83          // Escanear el nuevo archivo para que aparezca en la
               galer a de medios
84          scanMediaFile(file)
85
86      } catch (e: Exception) {
87          e.printStackTrace()
88          runOnUiThread {
89              binding.TextView.text = "Bitmap guardado catch"
90          }
91      }
92  }
93
94  private fun scanMediaFile(file: File) {
95      // Escanear el archivo para que aparezca en la galer a de
           medios
96      MediaScannerConnection.scanFile(
97          this,
98          arrayOf(file.path),
99          null
100     ) { _, uri ->
101         // Puedes manejar la URI si es necesario
102         //runOnUiThread {
103         //    binding.TextView.text = "Bitmap guardado en $uri"
104         //}
105     }
106 }
107
108
109
110 private fun saveTextToFile(grayscaleValues: Array<IntArray>) {
```

```
111     val intent = Intent(Intent.ACTION_CREATE_DOCUMENT).apply {
112         addCategory(Intent.CATEGORY_OPENABLE)
113         type = "text/plain"
114         putExtra(Intent.EXTRA_TITLE, "Matrices_${System.
               currentTimeMillis()}.txt")
115     }
116     startActivityForResult(intent, REQUEST_SAVE_FILE)
117 }
118
119 override fun onActivityResult(requestCode: Int, resultCode: Int,
       data: Intent?) {
120     super.onActivityResult(requestCode, resultCode, data)
121
122     if (requestCode == REQUEST_SAVE_FILE && resultCode == Activity
           .RESULT_OK) {
123         data?.data?.let { uri ->
124             applicationContext.contentResolver.openOutputStream(
                   uri)?.use { outputStream ->
125                 val bufferedWriter = BufferedWriter(
                       OutputStreamWriter(outputStream))
126                 try {
127                     for (row in grayArray!!) {
128                         for (value in row) {
129                             bufferedWriter.append("$value ")
130                         }
131                         bufferedWriter.append("\n")
132                     }
133                     bufferedWriter.flush()
134                 } catch (e: IOException) {
135                     e.printStackTrace()
136                     binding.TextView.text = "Error al guardar el
                           archivo"
137                 } finally {
138                     bufferedWriter.close()
139                 }
140                 runOnUiThread {
141                     binding.TextView.text = "$totalImagesSaved
                           matrices guardadas"
142                 }
143             }
```

```
144         }
145     }
146 }
147
148 private fun bitmapToGrayscaleArray2D(bitmap: Bitmap): Array<
    IntArray> {
149     val width = bitmap.width
150     val height = bitmap.height
151     val pixels = IntArray(width * height)
152
153     // Obtener los p xeles del Bitmap
154     bitmap.getPixels(pixels, 0, width, 0, 0, width, height)
155
156     // Crear un nuevo array bidimensional para los valores de
        intensidad de gris
157     val grayscaleValues = Array(height) { IntArray(width) }
158
159     var index = 0
160     for (y in 0 until height) {
161         for (x in 0 until width) {
162             val pixel = pixels[index]
163
164             // Extraer componentes de color
165             val red = Color.red(pixel)
166             val green = Color.green(pixel)
167             val blue = Color.blue(pixel)
168
169             // Calcular el valor de intensidad de gris (promedio
                de los componentes)
170             val grayscale = 0.299 * red + 0.587 * green + 0.114 *
                blue
171
172             // Almacenar el valor de intensidad de gris en el
                array bidimensional
173             grayscaleValues[y][x] = grayscale.toInt()
174
175             index++
176         }
177     }
178
```

```
179     return grayscaleValues
180 }
181
182
183 private fun downsamplingGrayscaleBitmap (bitmap: Bitmap,
        squareSize: Int): Bitmap {
184     val width = bitmap.width
185     val height = bitmap.height
186     val limitX = width / squareSize
187     val limitY = height / squareSize
188
189     val downsampledBitmap = Bitmap.createBitmap(limitX, limitY,
            Bitmap.Config.ARGB_8888)
190
191     for (x in 0 until limitX) {
192         for (y in 0 until limitY) {
193             var sum = 0
194             var count = 0
195
196             for (i in x * squareSize until (x + 1) * squareSize) {
197                 for (j in y * squareSize until (y + 1) *
                        squareSize) {
198                     val pixel = bitmap.getPixel(i, j)
199                     val intensity = pixel and 0xFF // Extrae la
                            intensidad en escala de grises
200                     sum += intensity
201                     count++
202                 }
203             }
204
205             val avgIntensity = sum / count
206             val color = 0xFF shl 24 or (avgIntensity and 0xFF) or
                    ((avgIntensity and 0xFF) shl 8) or ((avgIntensity
                    and 0xFF) shl 16)
207             downsampledBitmap.setPixel(x, y, color)
208         }
209     }
210     return downsampledBitmap
211 }
```

8.9. Red neuronal en Kotlin

```kotlin
fun neuralNetwork(context: MainActivity, imageMatrix: Array<DoubleArray>): Array<Array<Double>> {
    val wl = readTxtToMatrix(context, R.raw.w_l)
    val wlm1 = readTxtToMatrix(context, R.raw.w_lm1)

    // Imprimir dimensiones y matrices
    println("wl dimensions: ${wl.size} x ${wl[0].size}")
    //println("wl matrix:")
    //printMatrix(wl)

    println("wlm1 dimensions: ${wlm1.size} x ${wlm1[0].size}")
    //println("wlm1 matrix:")
    //printMatrix(wlm1)

    val imageColumn = matrizColumna(imageMatrix)

    println("imageColumn dimensions: ${imageColumn.size} x ${imageColumn[0].size}")
    //println("imageColumn matrix:")
    //printMatrix(imageColumn)

    val zlm1 = multMatrix(wlm1, imageColumn)
    println("zlm1 dimensions: ${zlm1.size} x ${zlm1[0].size}")
    //println("zlm1 matrix:")
    //printMatrix(zlm1)

    val alm1 = sigmoid(zlm1)
    println("alm1 dimensions: ${alm1.size} x ${alm1[0].size}")
    //println("alm1 matrix:")
    //printMatrix(alm1)

    val zl = multMatrix(wl, alm1)
    println("zl dimensions: ${zl.size} x ${zl[0].size}")
    //println("zl matrix:")
    //printMatrix(zl)

    val al = salida(zl)
```

```
36      println("al dimensions: ${al.size} x ${al[0].size}")
37      //println("al matrix:")
38      //printMatrix(al)
39
40      return al
41  }
42
43  fun sigmoid(matrix: Array<Array<Double>>): Array<Array<Double>> {
44      val result = Array(matrix.size) { Array(matrix[0].size) { 0.0
            } }
45
46      for (i in matrix.indices) {
47          for (j in matrix[i].indices) {
48              result[i][j] = 1.0 / (1.0 + exp(-matrix[i][j]))
49          }
50      }
51
52      return result
53  }
54
55  fun salida(h: Array<Array<Double>>): Array<Array<Double>> {
56      val sumExp = h.flatten().map { exp(it) }.sum()
57
58      val result = Array(h.size) { Array(h[0].size) { 0.0 } }
59
60      for (i in h.indices) {
61          for (j in h[i].indices) {
62              result[i][j] = exp(h[i][j]) / sumExp
63          }
64      }
65
66      return result
67  }
68
69
70  fun readTxtToMatrix(context: MainActivity, resourceId: Int): Array
        <Array<Double>> {
71      val matrix = mutableListOf<Array<Double>>()
72
73      try {
```

```
74          val inputStream = context.resources.openRawResource(
               resourceId)
75
76          BufferedReader(InputStreamReader(inputStream)).use {
               reader ->
77              reader.readLines().forEach { line ->
78                  // Modifica esta l nea seg n el formato de tu
                       archivo de texto
79                  val values = line.split(",").map {
80                      try {
81                          it.toDouble()
82                      } catch (e: NumberFormatException) {
83                          // Manejar casos donde la conversi n a
                               Double falla
84                          // Podr as imprimir un mensaje de error o
                               manejarlo de otra manera
85                          e.printStackTrace()
86                          0.0 // O alg n valor predeterminado
87                      }
88                  }.toTypedArray()
89
90                  // Aseg rate de que haya al menos un valor antes
                       de agregar la fila a la matriz
91                  if (values.isNotEmpty()) {
92                      matrix.add(values)
93                  }
94              }
95          }
96      } catch (e: Exception) {
97          e.printStackTrace()
98      }
99
100     return matrix.toTypedArray()
101 }
102
103
104 fun multMatrix(
105     MatrixA: Array<Array<Double>>,
106     MatrixB: Array<Array<Double>>
107 ): Array<Array<Double>> {
```

```
108
109     val rowsA = MatrixA.size
110     val columnsA = MatrixA[0].size
111     val rowsB = MatrixB.size
112     val columnsB = MatrixB[0].size
113
114     println("Multiplying matrices with dimensions: ($rowsA x
            $columnsA) * ($rowsB x $columnsB)")
115
116     val product = Array(rowsA) { Array(columnsB) { 0.0 } }  //New
            matrix with 0.0 in its elements
117
118     if (columnsA != rowsB) {
119         println("Multiplication not possible: columnsA ($columnsA)
                != rowsB ($rowsB)")
120         return product
121     }
122
123     for (i in 0 until rowsA) {
124         for (j in 0 until columnsB) {
125             for (k in 0 until columnsA) {
126                 product[i][j] += MatrixA[i][k] * MatrixB[k][j]
127             }
128         }
129     }
130     return product
131 }
132
133 fun printMatrix(matrix: Array<Array<Double>>) {
134     for (row in matrix) {
135         println(row.joinToString(", ") { it.toString() })
136     }
137 }
138
139 fun matrizColumna(matriz: Array<DoubleArray>): Array<Array<Double
        >> {
140     val numeroFilas = matriz.size
141     val numeroColumnas = matriz.firstOrNull()?.size ?: 0
142
143     if (numeroColumnas == 0) {
```

```
144         // Manejar caso de matriz vac a o sin columnas
145         return emptyArray()
146     }
147 
148     val matrizResultante = Array(numeroFilas * numeroColumnas) {
        Array(1) { 0.0 } }
149 
150     for (columna in 0 until numeroColumnas) {
151         for (fila in 0 until numeroFilas) {
152             val valor = matriz[fila][columna]
153             matrizResultante[columna * numeroFilas + fila][0] =
                valor
154         }
155     }
156 
157     return matrizResultante
158 }
159 
160 
161 fun matrixArrayintToArraydouble(arrayInt: Array<IntArray>): Array<
    DoubleArray> {
162     return Array(arrayInt.size) { filaInt ->
163         DoubleArray(arrayInt[filaInt].size) { columnaInt ->
164             arrayInt[filaInt][columnaInt].toDouble()
165         }
166     }
167 }
168 
169 fun findMaxVal(matriz: Array<Array<Double>>): Triple<Double, Int,
    Int> {
170     var maxValor = Double.MIN_VALUE
171     var filaPosicion = -1
172     var columnaPosicion = -1
173 
174     for (i in matriz.indices) {
175         for (j in matriz[i].indices) {
176             if (matriz[i][j] > maxValor) {
177                 maxValor = matriz[i][j]
178                 filaPosicion = i
179                 columnaPosicion = j
```

```
180                 }
181             }
182         }
183
184         return Triple(maxValor, filaPosicion, columnaPosicion)
185 }
186
187 fun printImage(matrix: Array<DoubleArray>) {
188         for (row in matrix) {
189             println(row.joinToString(", ") { it.toString() })
190         }
191 }
```