UNIVERSIDAD DE PIURA

FACULTAD DE INGENIERÍA

Predicción de parámetros de calidad de la harina de pescado utilizando Imágenes Hiperespectrales y Redes Neuronales Artificiales

Trabajo de Investigación para optar el Grado de
Bachiller en Ingeniería Mecánico - Eléctrica

Isabel del Pilar Moscol Albañil
Gleen Peltroche Saavedra
Víctor Augusto Ruesta García

Asesor:
Dr. Ing. William Ipanaqué Alama

Piura, junio de 2021

UNIVERSITAS STUDIORUM PIURENSIS

Resumen

Las Imágenes Hiperespectrales son utilizadas para detectar parámetros que muestran más información que los instrumentos convencionales. Esta información se representa como una firma espectral, cuyos valores de reflectancia pueden ser utilizados como entrada a una Red Neuronal Artificial para la detección, clasificación y estimación de los parámetros principales en la calidad de la harina de pescado.

Actualmente, en muchas empresas de harina de pescado se carece de un control automático y en tiempo real de los parámetros de calidad: proteína, humedad, ceniza, grasa, entre otros. El problema se establece a la salida del proceso de secado, donde la optimización de los parámetros de calidad final es posible a través del reingreso controlado del producto al secador para reducir la humedad hasta que alcance un porcentaje óptimo, de forma homogénea y sin quemaduras en el producto. La presente investigación plantea una metodología que combina las Imágenes Hiperespectrales y Redes Neuronales Artificiales para intentar resolver esta problemática. Los algoritmos desarrollados estiman los principales parámetros para determinar el nivel de calidad de la harina de pescado de forma no invasiva para su eventual uso en las etapas de secado.

Como resultado, se obtuvo una buena correlación entre la reflectancia, proporcionada por la Imagen Hiperespectral, y los principales parámetros de calidad tras implementar un algoritmo de Red Neural Perceptrón Multicapa.

UNIVERSITAS STUDIORUM PIURENSIS

Tabla de contenido

Lista de tablas

UNIVERSITAS STUDIORUM PIURENSIS

Lista de figuras

Introducción

El Perú es actualmente se encuentra entre los principales productores de harina de pescado, fabricada a base de anchoveta, este producto se utiliza como alimento para animales debido a su gran valor nutricional, que está asociado a la calidad de la harina de pescado.

En la última década, la industria alimentaria ha sufrido un gran incremento, por ello, la alta demanda actual de harina de pescado y la creciente competencia del mercado internacional requiere de nuevas tecnologías para cumplir con los estándares de calidad cada vez más exigentes del cliente.

Se establece una problemática en la etapa del secado, uno de los procesos más importantes para determinar la calidad de la harina de pescado. Presenta consumos de vapor muy variables que afectan su costo y calidad del producto final. El objetivo de la presente investigación es el diseño de un modelo para la predicción de humedad, proteína, grasa y ceniza de la harina de pescado. Conocer estos parámetros permite cuantificar el material necesario a secar, esto beneficiaria a la obtención de una harina de pescado de mejor calidad.

En la presente investigación se plantea mejorar la calidad de la harina de pescado a través de la automatización del control de calidad mediante la predicción de sus parámetros utilizando imágenes hiperespectrales y redes neuronales. Ambas son nuevas tecnologías que hoy en día es posible usar gracias al desarrollo y abaratamiento de *hardware* capaz de procesar esta información en tiempo real.

UNIVERSITAS STUDIORUM PIURENSIS

Capítulo 1
Antecedentes

1.1. Problema y Justificación

Actualmente, la medición de las propiedades finales de la harina de pescado se realiza durante el proceso de envasado en sacos donde se toma una muestra de esta para la medición de parámetros de calidad físico, químicos y microbiológicos. Entre los parámetros químicos se encuentran: proteína, humedad, grasa y cenizas. Los resultados de las pruebas de laboratorio se obtienen después de que el lote ya se encuentra envasado y listo para su envío, siendo inmodificable cualquier factor químico. Una de las etapas más relevantes es la de secado, donde se disminuye gradualmente la concentración de humedad, vinculada a los índices de proteína, factor importante en la clasificación de calidad. Por ello, es necesario hacer énfasis en el control y monitoreo de la harina en dicha etapa.

Por ello, este proyecto de investigación plantea medir los parámetros de calidad más relevantes a la salida del proceso de secado de manera no invasiva y en un tiempo de adquisición de resultados más bajo respecto al método tradicional sin necesidad de mano de obra constante durante el proceso, para ello, se desarrollarán algoritmos de Redes Neuronales Artificiales con los datos obtenidos mediante la tecnología de Imágenes hiperespectrales, cuya adecuada implementación a futuro permitiría automatizar el proceso de secado. Entre las ventajas se encuentra la medición de parámetros de calidad en tiempo real y con mayor precisión; las industrias podrían reducir costos, tiempo y mejorar la calidad del producto final.

1.2. Objetivos y Alcance

1.2.1. *Objetivo general*

1. Desarrollar modelos personalizados con la estructura de una Red Neuronal Perceptrón Multicapa para cada uno de los principales parámetros: proteína, grasa, humedad y cenizas para lograr la correcta clasificación de la calidad de la harina de pescado según los estándares internacionales.

1.2.2. *Objetivos específicos*

1. Buscar bibliografía sobre el desarrollo actual de redes neuronales.
2. Relacionar firma espectral de la harina de pescado con los parámetros de salida.

3. Desarrollar un algoritmo de red neuronal para estimar cada uno de los cuatro parámetros seleccionados de la harina de pescado.
4. Validar el modelo con nueva data tomada en las mismas condiciones que la utilizada para entrenamiento y validación.

1.3. Estado del arte

1.3.1. *Producción de la harina de pescado*

La calidad nos permite conocer el conjunto de características y prioridades que generan un grado o nivel de superioridad en un producto, con la finalidad de expandirse en el mercado por encima de la competencia, siempre buscando cumplir con los requerimientos del cliente, el grado de calidad es el pilar fundamental de las industrias modernas. En caso de la harina de pescado siempre se busca que tenga el mayor grado de pureza posible, un excelente valor nutritivo, con una baja composición de agentes químicos que puedan evitar su descomposición, todo esto con la finalidad de producir un buen impacto en la industria alimentaria (animal) siempre buscando un equilibrio entre consumidor, productor y además siendo amigables con el ambiente.

La harina de pescado producida en Perú está hecha a base de anchoveta; esta especie puede alcanzar una longitud de 12 cm, siendo esta una medida estándar para su pesca. Existen diferentes calidades en la harina de pescado, esto es producto del tipo de pez usado para la fabricación, en algunas zonas de américa la harina de pescado está hecha a base de los residuos del procesamiento de fábricas de conservas o muelles; mientras que en países europeos se hace uso de otras especies, tales como el Capelán o el Arenque.

La harina de pescado producida en Perú es una de las más comercializadas en mundo ya que está hecha a base de anchoveta, la cual es abundante en el mar nacional; según la IFFO (organización mundial de ingredientes marinos) la harina de pescado a nivel comercial tiene una composición de proteína entre 64% y 67%, 12% de grasa, entre 9% y 10% de humedad y bajo contenido de sal que redondea el 3%; todo esto variando según el estándar de calidad, la harina de carácter especial puede llegar a contener entre 68% y 72% de proteína. La harina de pescado producida en Perú tiene una composición de proteína de 64% y 68%, 12% de grasa, entre 6% y 10% de humedad, entre 12% y 18% de ceniza y 2.98% de sal aproximadamente.

1.3.2. *Estándares actuales en la industria*

GMP+B2

La norma GMP+B2 está basada en la norma HACCP (es un proceso que garantiza la seguridad alimentaria) y plantea las condiciones necesarias que debe cumplir el producto base para la producción de aceite y harina de pescado de manera industrial, así mismo plantea las condiciones y métodos en las que debe ser almacenado el producto obtenido después de todo el proceso de secado y pulverizado. (Obtenido de Memorial Anual COPEINCA).

GMP B3

La norma GMP+ B3 da a conocer los requisitos necesarios para poder realizar comercio, recolección, almacenamiento y transbordo, asegurando que el producto no es peligroso para el consumo y además garantiza que se encuentra en buen estado. (Obtenido de GMP Certificación).

IFFO

IFFO contiene una serie de certificados que avalan el tipo de materia prima usada para la producción de harina y aceite de pescado, así mismo verifica que los parámetros necesarios para poder clasificar la calidad de la harina de pescado se encuentren dentro de los márgenes internacionales, también verifica que el abastecimiento va a la par de la supervisión de FAO, además busca que las condiciones de salubridad del producto sean óptimas para el consumo animal. (Obtenido de IFFO)

1.3.3. *Métodos tradicionales para la medición de parámetros*

La industria de la harina de pescado usa una serie de estándares guía que establecen los parámetros necesarios para clasificar la harina y aceite de pescado como especial o de baja calidad, generalmente para determinar el grado de calidad se toman en cuenta muchos factores que varían desde el tipo de materia prima usada para la producción hasta el tipo de procesos usados durante y después de haber obtenido el producto final.

Generalmente se usan parámetros como la frescura de la materia prima a usar, la humedad relativa en el producto obtenido, la cantidad de proteína, la cantidad de grasa obtenida y la calidad de esta misma, cantidad de ceniza, entre otros; así mismo se hace uso de distintos métodos que pueden variar según el nivel de producción de la empresa y el tipo de equipos disponibles para realizar pruebas, en esta breve descripción se dará a conocer el porqué de los parámetros establecidos y además se mencionara los métodos usados con mayor frecuencia.

Frescura

La frescura es un parámetro usado para la clasificación de la materia prima usada para la producción de harina y aceite de pescado, esto debido a que en algunos países la materia prima base son los restos de peces que ya han sido usados en su mayor parte para el consumo humano, estos restos pueden provenir de muelles, fábricas de envasado, entre otros; normalmente de este tipo de materia prima se obtiene “harina blanca” la cual tiene un contenido medio de proteína en comparación con la harina hecha de la pesca industrial de pescado.

Se denomina “harina especial” al tipo de harina producto de usar materia prima “fresca”, normalmente hecha a base de arenque o anchoveta, ya que estos tipos de peces tienden a tener un tiempo de vida muy corto y además su reproducción en su medio natural

es a gran escala, estos peces al pasar directamente de la red al proceso de producción no presentan degradación severa en su composición, por ende contienen un nivel más elevado de proteína, grasas, entre otros; también se aplica procesos de refrigeración, comúnmente son 3:

- Conservación por nitritos: no es el más usado ya que tienden a tener efectos en la materia prima.
- Conservación por agua salada: es de uso común pero su uso provoca que los niveles de sal se eleven.
- Conservación por agua dulce mezclada con hielo: es el más recomendado ya que no tiene influencia directa sobre la materia prima y además permite ser usado por largos periodos de tiempo.

Proteína

La proteína es uno de los parámetros más importantes para conocer la calidad de la harina de pescado, como ya se mencionó antes, este factor es tanto dependiente del tipo de pescado, la frescura y estado de la materia prima y no menos importante, el tipo de refrigeración; el porcentaje de proteína permite clasificar la harina en "harina blanca" con un contenido de 65%; "harina de arenque" con 72% de proteína y " harina de Sudamérica" con un contenido de 65%; la principal diferencia entre la harina blanca y de Sudamérica radica en el contenido de ceniza, ya que la harina blanca contiene 20% de ceniza mientras que la sudamericana tiene entre 12% y 16%.

El método más común usado para medir el porcentaje de proteína consta de una primera etapa en donde se pesa una muestra seleccionada con cautela sin alterar el producto en lo más mínimo, luego haciendo uso de un tubo "Kjeldahl" y ácido sulfúrico se realiza un proceso denominado digestión, en donde mediante la aplicación de calor sobre esta muestra permite la cuantificación del nitrógeno, el cual es multiplicado por un factor que varía según el tipo de materia prima usada.

Humedad

Según parámetros internacionales el contenido de humedad debe estar entre 5% y 10%, la humedad es un agente que influye de manera directa en la degradación de la materia prima y del producto obtenido, permitiendo la proliferación de agentes como hongos o bacterias, los cuales contaminan en su totalidad el producto; generalmente en la harina de calidad especial, los límites de humedad redondean entre 7% y 10%.

El método más común usado en la industria consiste en someter calor una muestra de aproximadamente 10 gramos ya homogeneizada, esta muestra es previamente pesada en una balanza de alta precisión, en donde después de la aplicación de calor se vuelve a pesar y la

variación entre el peso inicial y el peso final es la cantidad de humedad que había en la muestra, que luego mediante una simple relación se estima el porcentaje total.

Grasa

La grasa en la harina de pescado es un factor comúnmente buscado ya que esta es una buena fuente de energía para los consumidores (animales), lo cual permite que tengan un mejor desarrollo físico y una mayor actividad hablando en términos de producción; la harina de pescado en promedio contiene entre 10% y 12% de grasa, no se busca sobrepasar estos valores ya que afecta la fluidez del producto y este es un factor en contra durante el proceso de producción.

El método común para lograr llevar un control de los niveles de grasa en la harina de pescado es mediante la aplicación de antioxidantes (controlados y establecidos por norma); así mismo la medida de este parámetro se da mediante dos métodos:

- Método de la acetona:

Mediante el aparato extractor "Gold Fish" se somete a una extracción de 16 horas haciendo uso de acetona, el líquido obtenido se mezcla con acetona para realizar un proceso de "limpieza", la mezcla es sometida a calor (aproximadamente 80° C) y provocara la evaporación total de la acetona, este proceso se puede llegar a repetir entre 2 a 3 veces añadiendo otros agentes químicos que facilitan el proceso de filtrado y destilado, posteriormente el líquido obtenido se pesa y mediante una relación matemática se cuantifica el porcentaje.

- Método de *Bligh* and *Dryer*

Este método consiste es realizar un ajuste de humedad, posteriormente mediante la aplicación de cloroformo y metanol se extrae la humedad de la muestra; este proceso se puede repetir entre 2 a 3 veces, variando según el tipo de materia prima, posteriormente mediante un proceso de filtrado y separación se eliminan las partículas sólidas de la muestra; para luego aplicar calor al líquido obtenido en donde el cloroformo se evaporara dejando únicamente la grasa , el porcentaje se calcula mediante la diferencia de peso inicial y final.

Cenizas

Como ya se mencionó antes el contenido de ceniza en la harina de pescado es un factor dependiente del tipo de materia prima usada para su producción, la tolerancia de ceniza en la harina de pescado esta entre 10% y 45%, generalmente este factor tiene esas variaciones debido a los niveles de fosforo, calcio, entre otros; estos a su vez son producto de los restos óseos, debido a que la materia prima usada en ocasiones son restos de pescado, la relación existente entre los niveles de "carne o masa" y hueso tiene una influencia directa en el producto, es esta una de las razones principales de la diferencia de calidad entre la harina producida en Sudamérica y otros países como Venezuela o países europeos.

El método usado para cuantificar el porcentaje de ceniza empieza por el pesado de una muestra en una balanza de alta precisión, posteriormente es sometida a altas temperaturas durante un prolongado periodo de tiempo, con esto se asegura que no hay humedad en la muestra; este proceso se repite 2 a 3 veces y se cada vez a niveles de temperatura mayor, con la finalidad de eliminar todo rastro de materia orgánica; finalmente es pesado y mediante relación matemática se calcula el porcentaje de ceniza.

Nitrógeno volátil total

Este parámetro permite conocer el nivel de degradación de la materia prima, como ya se mencionó antes, es posible calcularlo en el proceso de obtención de parámetro de la proteína.

Aminas Biogénicas

Parámetro usado para determinar la frescura de la materia prima, es el parámetro más usado para determinar este factor.

Actualmente los estándares de calidad para la producción de comercialización de aceite y harina de pescado se han expandido, ya que ahora se consideran importantes otros factores como digestibilidad, aminas biogénicas, score biotoxicológico.

Como ya se mencionó antes las aminas biogénicas es un parámetro usado para determinar el nivel de frescura de la materia prima, el porcentaje de aminas en los pescados “frescos” es considerado de carácter despreciable, según estudios realizados a la sardinas; su estado de descomposición empieza cuando es capturado en las redes; a partir de aquí la flora bacteriana toma lugar en todo el cuerpo del pez; la formación de aminas biogénicas se da cuando los aminoácidos propios de los músculos del pez comienzan a degradarse, esto desencadena un estado denominado “hidrolisis”, esto posteriormente da lugar a la descomposición de algunas enzimas y da lugar a la flora bacteriana; este proceso es inmediato a temperatura ambiente; se calcula que el crecimiento de esta flora se da de manera exponencial; algunos factores que están relacionados con la formación de aminas biogénicas:

- Método por el cual el pez ha sido extraído del mar.
- Modo de almacenamiento para llevarlos a planta.
- Tipo de refrigeración utilizada en planta.
- Tipo de pez y época del año.
- Temperatura ambiental.

Las aminas biogénicas no oscilan con a temperatura; es decir son indicadores acumulativos, que según el tiempo de exposición del pez y su nivel de descomposición se irán acumulando y permitirán conocer el estado del pez que será usado para la producción de aceite y harina de pescado.

Estos factores usados para la medición del índice de frescura de la materia primera se pueden relacionar mediante el "Índice de Bai", es cual está hecho para los productos sacados del mar; mediante relaciones matemáticas se logra llegar a una ecuación que indica el índice de frescura de la materia, teniendo así:

$$IF = \frac{100Ca + Pu + 10Hi + 20Ty + 10Ag}{1000}$$

1.3.4. *Machine Learning*

Actualmente, *machine learning* es muy aplicado a la industria alimenticia con el soporte de las imágenes hiperespectrales quienes permiten obtener procesos automatizados para el control de calidad de muy buena precisión. En Sun (2010) se hace el análisis de calidad en alimentos mediante imágenes hiperespectrales e inteligencia artificial. Uno de los alimentos evaluados fue el filete de pescado mediante un sistema NIR. Se analizó cada píxel de la imagen hiperespectral para predecir la concentración de agua y grasa, multiplicando el espectro de cada píxel con el vector de coeficiente obtenido a partir del modelo PLSR (*Partial Least Squares Regression*).

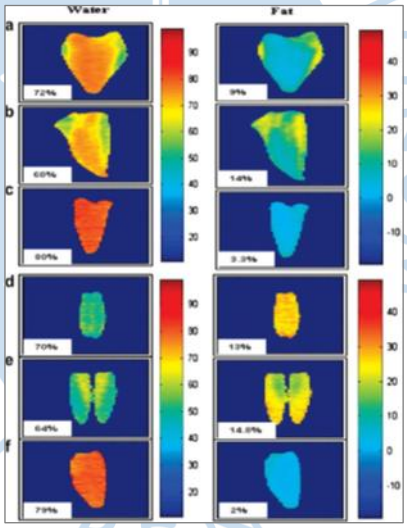

Figura 1. Distribución de agua y grasa en filetes de pescado.

Fuente: Tomado de (Sun, D., 2010).

Los valores porcentuales de la parte inferior izquierda hacen referencia al promedio de agua y grasa de cada filete. Se observa cómo la concentración de agua y grasa varía drásticamente en distintas zonas del mismo filete analizado.

Las imágenes hiperespectrales son utilizadas para hacer estudios en alimentos de manera no destructiva ni invasiva, por ello, en Sun (2010) se analizaron las imágenes hiperespectrales de la zona lateral de un pescado (caballa) con la finalidad de medir la reflectancia espectral VIS/NIR (Espectroscopia del infrarrojo cercano) media. A continuación,

se muestra la firma espectral tras 1, 2, 4 y 6 días de muerto (d.p.m = *days post mortem*) para estudiar cómo varía el grado de frescura en la carne del pescado. La espectroscopía NIR está despertando interés en el sector industrial debido al análisis rápido y de coste muy reducido con preparación de muestra prácticamente nula y sin generación de residuos (Peguero, 2010).

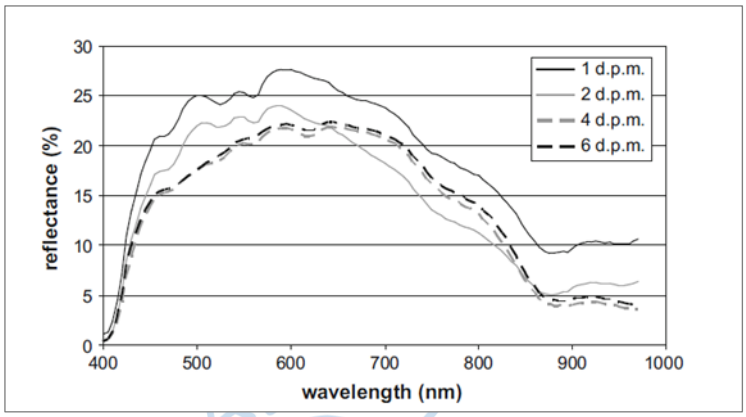

Figura 2. Firma espectral promedio en días post mortem de una Caballa tras conservase por congelación.

Fuente: Tomado de (Sun, D, 2010).

1.3.5. *Imágenes Hiperespectrales*

Este trabajo se centrará en la tecnología hiperespectral aplicada a determinar los parámetros de calidad de la harina de pescado y a su vez con estos automatizar el proceso de secado. Por ello se han revisado artículos que analizan información espectral.

En (Mundaca, G., Soto, J., & Ipanaqué, W, 2015) se muestra que hay una gran eficacia con el uso de las técnicas de procesamiento de imágenes hiperespectrales al evaluar con la calidad de granos de cacao permitiendo a través de los análisis de índices espectrales correlacionar el índice de antocianina AR12.

Figura 3. Imagen Hiperespectral en NIR y RGB.

Fuente: Tomado de (Mundaca et al., 2015).

Se establece un procedimiento para identificar parámetros fisicoquímicos de harina de pescado aplicando la tecnología de Imágenes Hiperespectrales mediante técnicas de visión artificial, procesamiento de señales y modelos de regresión basados en algoritmos de *Machine*

Learning con aprendizaje supervisado. El procedimiento se aplicó a la salida del proceso de producción de la harina de pescado. (Cherre, 2019).

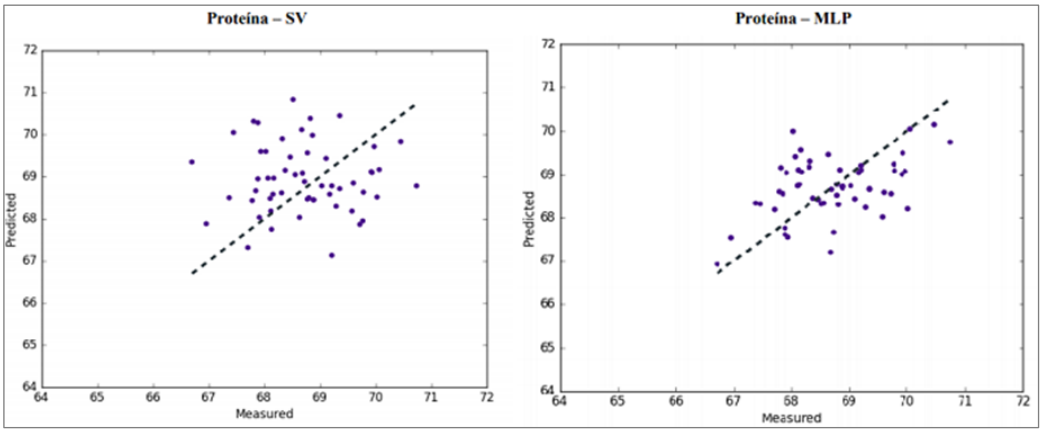

Figura 4. Comparación entre los modelos *Support Vector Regression* y *Multilayer* Perceptron para estimar el parámetro de la proteína.

Fuente: Tomado de (Cherre, 2019).

Se comprueba una mayor calidad de predicción con el algoritmo de MLP (*Multilayer Perceptron*) por encima del algoritmo SV (*Support Vector Regression*). (Cherre, 2019).

Para el uso de la correcta información que se obtiene de las imágenes hiperespectrales es indispensable que las imágenes sean obtenidas de la mejor manera posible. Los métodos para el procesamiento y análisis de imágenes digitales RGB en su mayoría pueden utilizarse en el procesamiento y análisis de las imágenes hiperespectrales.

En (United States Patente nº US 9,176,110 B1, 2015) se plantea un modelo para determinar la histamina en pescados usando índices espectrales y de manera no destructiva (ver Figura 5).

En (Mutlu A. H.-A., 2011) se comenta que "el uso de las imágenes tiene grandes aportes para el análisis no destructivo prediciendo parámetros de maíz usando el Multiplayer Perceptron, teniendo la capacidad de predecir numerosos parámetros como la proteína, humedad, zeleny, entre otros. Este método no invasivo es de gran importancia debido a que la materia no sufre ningún daño y se obtiene una reducción de costos.

1.3.6. *Redes Neuronales*

Las redes neuronales se describen como modelos de computación y aprendizaje, son uno de los elementos que nos permite clasificar de una mejor manera cualquier patrón en el cual vayamos a trabajar dentro del mundo de la toma de decisiones, además una característica importante es que la propia red es capaz de aprender durante las fases de entrenamiento, mediante lo cual posteriormente se podrá extrapolar en decisiones para las cuales no ha sido entrenada con algún tipo de patrón especial.

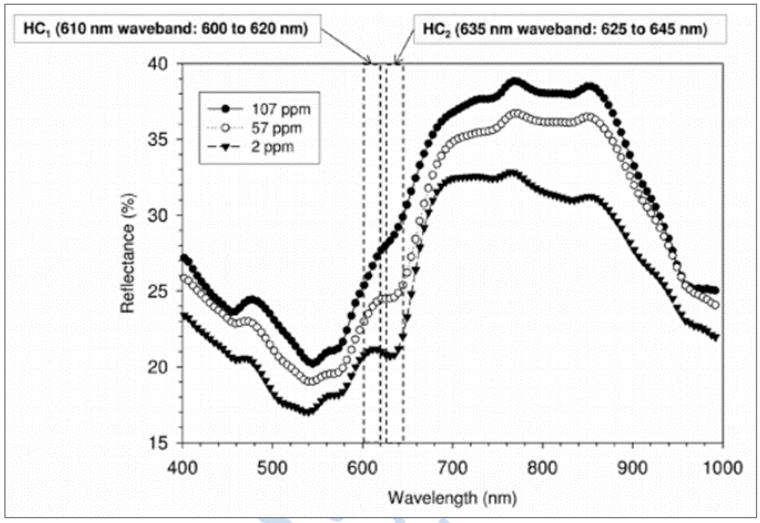

Figura 5. Identificación con firma espectral.

Fuente: Tomado de (United States Patente nº US 9,176,110 B1, 2015).

Las redes neuronales no son antiguas, a mediados del siglo pasado apareció por primera vez un modelo de una neurona, la cual era un intentaba imitar el comportamiento de las neuronas del cerebro humano, buscando imitar su capacidad de procesar y compartir información con otras neuronas; además de otorgar la capacidad de la toma de decisiones.

- 1943: McCullogh y Pitts; se modeló por primera vez una neurona.
- 1949: Hebb; planteó el ajuste de pesos; concepto fundamental para la implementación y entrenamiento de una red neuronal.
- 1958: Rosemblat- Perceptrón; se planteó por primera vez el principio de funcionamiento del perceptrón y además su desarrollo.
- 1969: Minsky y Papert; se realizó la crítica XOR; en donde se decía que el perceptron no podía clasificar todas las cosas haciendo uso de ecuaciones lineales.
- 1974: P.Werbos- solución PCM; enuncia los principios del perceptrón multicapa; en donde se hace uso de varias neuronas agrupadas por capas y conectadas entre si.
- 1986: Rumelhart y McCelland; se planteó el algoritmo de retro propagación, usado en el entrenamiento de las redes neuronales.

Haciendo uso de entradas denominadas “X” se realiza un análisis mediante una sumatoria de las entradas, modulando las entradas mediante los pesos “W” los cuales permiten conocer la intensidad con la que la variable afecta a la neurona y además se le puede añadir un término independiente (BIAS) que puede modificar la salida; la salida de la neurona

va a ser el resultado de los productos de las entradas por sus pesos más sus términos independientes.

Tipos de Redes Neuronales

Adaline, perceptrón, madaline: el perceptrón busca dividir y clasificar a partir del aprendizaje de patrones sencillos; hace uso de una función de activación que busca adaptar el sumatorio de las entradas. Se demuestra mediante la aplicación del perceptrón simple, es decir, la neurona tiene separabilidad lineal (AND, OR).

Perceptrón multicapa: La crítica antes mencionada XOR planteaba que al intentar hacer que el perceptrón solucione una función mediante un método no lineal se producía una falla, además el unir varias neuronas sin una función de activación se produce un fallo ya que matemáticamente una neurona es un modelo de regresión lineal y al estar agrupadas en capas y de manera secuencial (*feed forward*) se producirá una sumatoria de ecuaciones lineales dando como resultado otra ecuación lineal.

La solución planteada por P.Werbos fue el perceptrón multicapa en el cual se lograra realizar una separación haciendo uso de varias neuronas en varias capas interconectadas entre sí, para entrenar el perceptrón multicapa se hace uso del algoritmo de retro propagación, el cual busca modificar los pesos ideales de los cuales se obtiene la salida de la neurona modificando el resultado que se obtiene en el entrenamiento, es decir ese peso después del entrenamiento es función del peso anterior y de la salida esperada menos la diferencia de la salida obtenida.

Redes neuronales profundas; esta es una red de fácil manipulación y muy adaptable, con ella podemos procesar desde textos números hasta imágenes, su característica principal es que entre las capas de entrada y salida existen muchas capas ocultas, y en ocasiones su procesamiento requiere de un equipo con la capacidad para realizar múltiples operaciones.

Redes neuronales convolucionales; este tipo de redes neuronales están especialmente diseñadas con la finalidad de realizar procesamiento de imágenes, su principal característica es que en las capas ocultas se realizan convoluciones y clasificación de datos (*pooling*), las convoluciones se traducen como una reducción en la calidad de las imágenes procesadas; son lineales o varían en el tiempo, en caso de imágenes lo que realiza en las capas ocultas es una reducción progresiva de estas mismas en donde se identifican los puntos más importantes.

Redes neuronales recurrentes: son usadas para datos de tipo secuencial; en donde hay muchos parámetros que serán analizados por mis capas ocultas, se produce un efecto de memoria corta a largo plazo, todo esto se puede reducir al término “predicción”, en donde mediante un proceso de retroalimentación acumula datos a los cuales recurrir.

Algunas características de las redes neuronales artificiales:

- Alto paralelismo; ya que mientras se está llevando a cabo el proceso también se está tratando de manera paralela todos los sumatorios que están entrando a las neuronas.
- Fácil implementación HW; su uso actualmente es muy explotado ya que permite una gran adaptabilidad a distintos campos de desarrollo tomando como base modelos ya establecidos, en donde en ocasiones solo es necesario establecer nuestros parámetros o introducir nuestras variables.
- Robustez frente a fallas; en algún momento fallan algunas de las neuronas, en ocasiones producto de una falta de entrenamiento sin embargo normalmente las salidas siguen siendo aceptables.
- Capacidad de generalización; permiten extrapolar la información recopilada, a otros elementos los cuales no han sido entrenados.
- Naturaleza adaptativa (aprendizaje); debido al ajuste de los pesos y regulación del bias en cada iteración de la red neuronal.

Algoritmo de retropropagación

Tras haber realizado previamente ensayos o fases de entrenamiento, el algoritmo de retro propagación le permitirá a la red neuronal buscar a adaptar la salida de esta misma modificando los pesos que se le otorgan a cada variable independiente; esto se puede expresar mediante la siguiente ecuación:

$$W_{t+1} = W_t + \eta X_K \, (Y_K - W_T \, X_k)$$

Donde,

W_t= peso anterior

ŋ = coeficiente de aprendizaje variable

Y_K = valor esperado de la salida

X_K = valor de la entrada

$W_T \, X_k$ =sumatorio de los pesos por sus entradas

Fases de diseño de una red neuronal

Inicialmente se define su arquitectura en la cual vemos el número de capas, numero de neuronas luego se define cuáles son los patrones y las salidas teniendo en cuenta que hay patrones que se usaran para la fase de entrenamiento y validación, la fase de validación nos permitirá comprobar que la red está funcionando de la manera esperada con patrones que no han sido usado para el entrenamiento de la red, una vez hecho esto se selecciona el algoritmo de entrenamiento, luego mediante el algoritmo de retropropragación se realiza el

entrenamiento de la red, por último se pasa a la explotación mediante el uso de un hardware adecuado o como software dentro de un sistema de apoyo a la toma de decisiones.

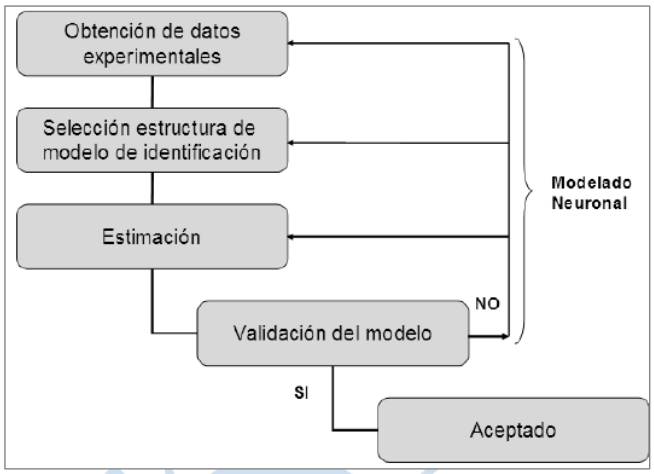

Figura 6. Etapas del diseño de una red neuronal.

Fuente: Tomado de (Scielo, 2012).

Aplicaciones

- Problemas con variables heterogéneas, por ejemplo: conceder un crédito bancario; en donde la institución bancaria usa filtros de edad, trabajo o historial crediticio.
- Problemas con valores variables en entornos continuos; la cual es la principal aplicación del uso de redes neuronales en este trabajo.
- Necesidad de gran número de patrones de entrada y respuesta, incluso nos permite realizar "predicciones" por ejemplo la bolsa de valores, en donde se busca prever el mercado de inversiones mundial.
- Interpolación para un dominio concreto

Avance en la actualidad

Las redes neuronales artificiales permitieron dar un gran avance en la ingeniería moderna, tras su desarrollo se ha buscado diferentes medios donde las redes neuronales tengan una aplicación óptima. Actualmente, las redes neuronales son la base de casi todo lo que nos rodea; siendo desarrolladas a partir de principios matemáticos (estadística).

Con la mejora de los procesadores y equipos de desarrollo de las redes neuronales, se está buscando el desarrollo de redes neuronales cada vez más complejas, basándose en lo que se conoce como aprendizaje profundo, el cual nos permite controlar procesos y predecir comportamientos o variables de carácter complejo; con el inicio de la industria 4.0 se busca reducir costos, aumentar la producción y mejorar la calidad de los productos y los servicios ofrecidos por las compañías; las redes neuronales nos permiten hacer un seguimiento

continuo de los procesos en ejecución, evitando posibles fallas (predicción) y que haya riesgos humanos o económicos de por medio.

Un claro ejemplo de la mejora de las redes neuronales buscando adaptarlas a las necesidades crecientes del día a día, es la red neuronal de Google denominada *Reformer*; la cual nos permite hacer uso de una gran variedad de términos lingüísticos de diferentes idiomas y procesar imágenes ocupando menos memoria de lo normal; Google antes de la llegada de *Reformer* hacía uso del modelo *Transformer*, el cual permitía traducir textos haciendo uso de las reglas gramaticales del idioma y de registros de memoria; con la implementación de *Reformer* se logró tener un mejor desempeño a la hora de traducción ya que se puede procesar 1000000 de palabras con solo 16 GB de memoria, esto mediante el uso de capas residuales y hash sensible a la localidad, las fases de entrenamiento de esta red neuronal permitieron que el procesamiento de imágenes con pocos pixeles o sin algunos de ellos, logrando identificar y completar las imágenes satisfactoriamente.

Otro ejemplo que permitirá un gran avance en el procesamiento de datos, es un algoritmo desarrollado en el MIT (*Massachusetts Institute of Technology*), este algoritmo permite realizar operaciones de manera muy eficiente y rápida (150-200 veces más rápido); como ya se sabe, no existe máquina o programa con la capacidad de igualar la capacidad de procesamiento del cerebro humano, en caso de las redes neuronales el procesamiento de una gran cantidad de datos conlleva una gran cantidad de tiempo y espacio de memoria, es decir, son directamente proporcionales y además de ser necesario el procesamiento de todos estos datos, se busca la implementación de nuevos o más equipos con la suficiente capacidad. Algunos algoritmos de Google tienen la capacidad de procesar mucha información, pero a costo de periodos muy grandes de tiempo (48000 horas) incluso teniendo la capacidad de ejecutar decenas de unidades de procesamiento gráfico, sin pasar por alto la gran cantidad y calidad de los equipos usados.

Capítulo 2
Estudio Teórico

2.1. Marco teórico

2.1.1. *Harina de pescado*

La harina de pescado en el Perú contiene, en promedio, entre 60% y 72% de proteína, un 9% de humedad y entre 10% a 12% de grasa. El principal uso de la harina de pescado se da en la formulación de alimentos de alta calidad en la acuicultura (la principal), avicultura, ganadería, entre otros.

Parámetros de calidad

Los parámetros de calidad son los encargados de darle competitividad a la Harina de pescado. Su clasificación se basa en la calidad del producto según estándares internacionales.

Tabla 1. *Parámetros de calidad de la harina de pescado*.

Parámetros			**Calidades**					
			Premium	**Super prime**	**Prime**	**Taiwan**	**Thailand**	**Standard**
Proteína	%	min	70	68	67	67	67	65/64
Grasa	%	max	10	10	10	10	10	10
Humedad	%	max	10	10	10	10	10	10
FFA	%	max	7	7.5	10	10	10	12
Cenizas	%	max	14	14	15	17	17	–
Arena y sal	%	max	4	4	5	5	5	5
TVN	100mg /100gr	max	85	100	120	120	150	–
Histamina	ppm	max	100	500	1000	–	–	–
Antioxidante	ppm	min	150	150	150	150	150	150

Fuente: Tomado de (Sociedad Nacional de Pesquería, 2018b)

2.1.2. *Proceso de producción*

Una vez que la anchoveta es extraída del mar, es llevada a las plantas de procesamiento, luego pasa por diversas etapas para convertirse en Harina de Pescado.

El proceso para la elaboración de la harina de pescado, posteriormente a adquirir la materia prima, comprende un conjunto de operaciones:

a) Recepción y almacenamiento de la materia prima

Primero se pesa la materia prima y se analizan las muestras respectivas, posteriormente se transportan a las tolvas de almacenamiento. (Corporación Pesquera Inca S.A.C, 2012).

b) Cocción

La materia Prima es ingresada y se le aplica un proceso térmico con vapor indirecto con el fin de detener la degradación de las proteínas en la fase sólida. (Farro, 1996).

Este proceso calienta la harina a temperaturas que oscilan los 95-96 grados centígrados. El objetivo de esta etapa es ayudar a solidificar las proteínas y desinfectar el producto. (Corporación Pesquera Inca S.A.C, 2012).

c) Desaguado

En esta fase se retira una porción del fluido resultante de la materia cocida, posteriormente tiene ingreso al equipo de recuperación de sólidos, este tiene paredes internas especiales para la fácil evacuación de los licores en donde posteriormente será combinado con el licor resultante de la cocción. La principal meta de esta etapa es lograr facilitar la etapa de prensado.

d) Prensado

Esta fase implica un procedimiento de prensado mecánico a la materia prima obtenida de la cocción, los tornillos comprimen fuertemente la masa, eliminando el Licor resultante del prensado a través de las rejillas, y una Torta de prensa por el extremo (masa sólida). (SAC, 2013).

e) Separado y centrifugado

El licor obtenido del prensado y el extracto obtenido del cocinador son transportados hacia los separadores con la finalidad de apartar la fase sólida de la líquida, buscando así en la parte liquida dividir el agua y el aceite por medio de centrifugado, además la parte sólida es sumada en el queque de la prensa, esto es en la etapa de secado (Farro, 1996).

f) Evaporado de agua de cola

Esta etapa consiste en la evaporación del agua de cola, persiguiendo la concentración para conseguir recuperar los sólidos restantes y la proteína soluble. En este proceso, la torta

prensada y el concentrado resultante se transmiten a los secadores. (Corporación Pesquera Inca S.A.C, 2012)

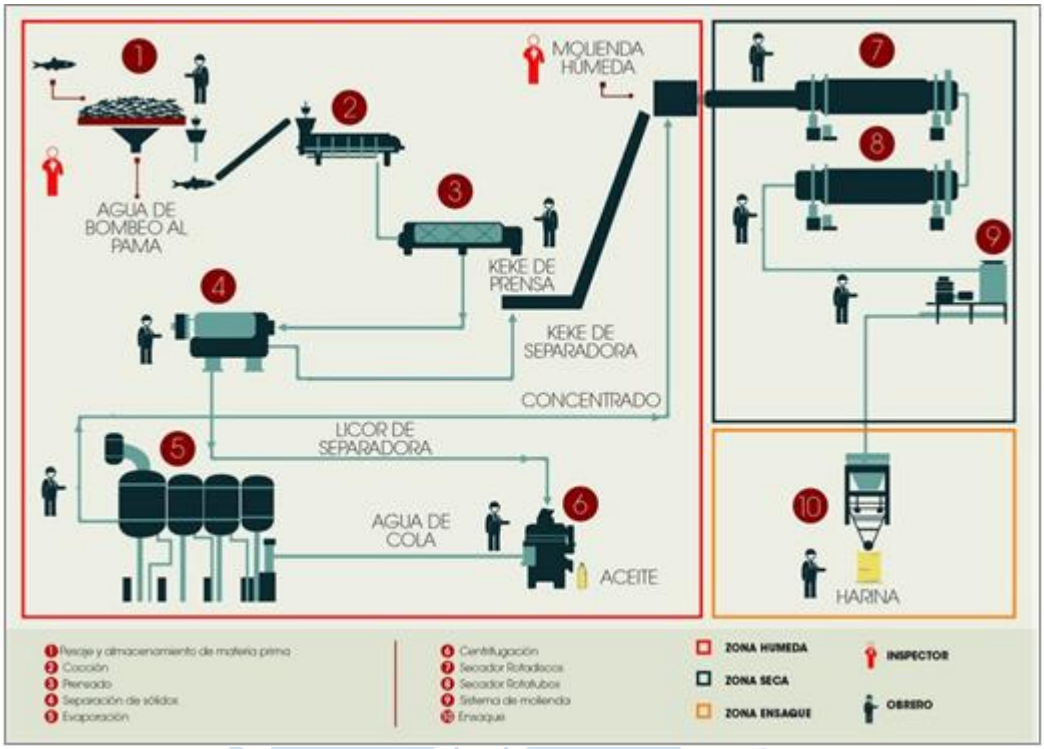

Figura 7. Proceso productivo de la harina y aceite de pescado.

Fuente: Tomado de (Sociedad Nacional de Pesquería, 2018a).

g) Secado

El propósito de este proceso es transformar la torta de prensa, el producto obtenido del decantador de lodos y además el licor de la etapa de evaporación en una harina consumible y seca. “Las Plantas utilizan un sistema de secado multietapa para obtener un nivel óptimo de humedad. La harina requiere un sistema de secado indirecto de vapor y/o aire caliente, esta misma se seca a temperaturas más bajas obteniéndose un contenido proteico mayor” (Corporación Pesquera Inca S.A.C, 2012).

La etapa de secado es el proceso que influye determinantemente en el nivel de calidad de la harina de Pescado, debido a la descomposición de los aminoácidos por la temperatura y además por la reducción de la humedad del “queque” de prensa. (Farro, 1996).

h) Molido

Los sólidos finos y gruesos se envían hacia un molino donde se pulveriza la harina obteniéndose el producto con gran granulometría. (Farro, 1996).

i) Adición de Antioxidante

Se realiza para generar una estabilidad en la harina y regular la oxidación del porcentaje de grasa, además de evitar el sobrecalentamiento. (Pizardi C., 1992).

j) Pesado y Envasado

La Harina de Pescado se empaca y pesa en sacos de poliuretano de 50 kg para luego ser almacenados en ambientes limpios y frescos formando rumas. (INDECOPI, 1986).

Tabla 2. *Muestras de harina de pescado.*

Fecha Producción	Proteína (Dumas)	Grasa	Humedad	Cenizas	Arena	Tbvn	Histamina
07/04/18	70.79	7.47	6.45	15.32	0.07	80.74	38.01
07/04/18	70.67	7.74	7.17	14.76	0.07	83	89.53
07/04/18; 08/04/18; 09/04/18	69.87	7.97	7.49	14.96	0.07	92.83	204.97
08/04/18	69.33	7.89	7.42	15.64	0.07	91.55	67.65
08/04/18	68.81	8.3	7.36	15.64	0.07	84.1	56.87
08/04/18; 09/04/18	69.13	7.84	7.7	15.72	0.08	93.53	157.12
08/04/18	69.86	8.43	7.01	15.06	0.07	99.89	145.37
08/04/18	69.69	8.12	7.18	15.38	0.07	99.69	180.03
08/04/18; 09/04/18	69.36	8.54	6.7	15.36	0.08	88.31	120.29

Fuente: Empresa peruana productora de harina de pescado.

Antes de la etapa de pesado y envasado se toman muestras de las rumas de harina para su medición. En la muestra 1 de la Tabla 2 se tiene buenos parámetros de proteína, grasa, humedad, arena, histamina y TBVN como para posicionarla como calidad PREMIUM, pero, la ceniza de 15.32 excede el máximo permitido de las calidades PREMIUM, SUPER PRIME y PRIME dejando esta ruma con calidad TAIWAN. En cambio, en la muestra 2 la ruma entra a la calidad de PRIME por tener una ceniza de 14.76. Podemos concluir que este factor ha jugado mucho en contra sobre la calidad de la harina en estas rumas.

2.1.3. *Problemática del secado*

En la producción, el secado es uno de los procesos más determinantes en la calidad de la harina y presenta consumos de vapor muy variables que afectan su costo y calidad del

producto final. Muchas veces se debe reprocesar el queque para obtener una harina de buena calidad, sin embargo, el reingreso no controlado adecuadamente del producto en proceso al secador podría generar que este se queme, no sea homogéneo y, además, por realizar nuevamente el proceso al mismo lote del producto se incrementa el costo de producción. Algunas causas son:

- Falta de mediciones en el secador.
- Pocas soluciones tecnológicas en el mercado nacional.
- Falta de control automático que se enfoque en el volumen (entrada) y temperatura (entrada y salida).

El objetivo es el diseño de un modelo para la predicción de parámetros de calidad (humedad, proteína, grasa, ceniza, arena) para así obtener datos en la etapa del proceso de secado.

Al conocer el porcentaje de humedad en la harina de pescado sería de conocimiento la disminución en el porcentaje de humedad necesario permitiendo la reintroducción controlada de la harina al secador rotatubos. De esta manera, se obtiene una harina de pescado con menor humedad y más homogénea que permita obtener los parámetros de calidad (proteína, cenizas, grasa, humedad, entre otros) en los indicadores de calidad óptimos para competir en el mercado internacional de la industria pesquera enfocada en la harina de pescado.

2.1.4. *Clasificación de imágenes*

2.1.4.1. RGB. Es un modelo aditivo constituido por los colores primarios rojo, verde y azul, basados en la luz visible por el ojo humano que únicamente detecta una región del espectro electromagnético (ver Figura 8). La luz visible corresponde a las longitudes de onda desde 400 nm a 700 nm. (Navarro J. C., 2007).

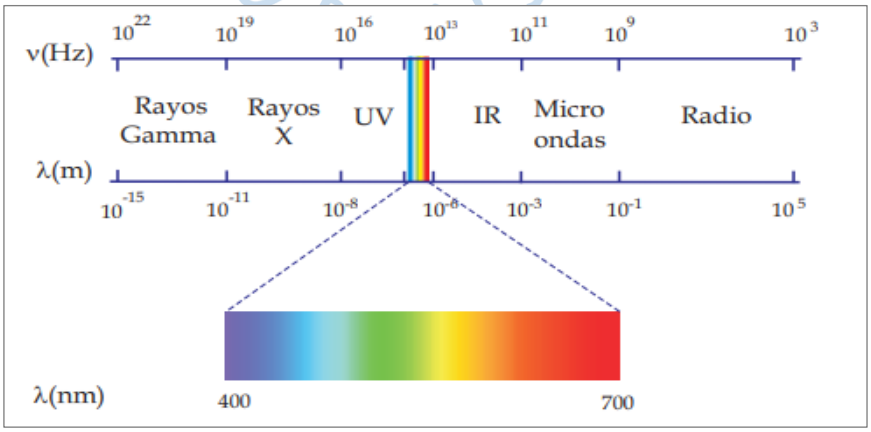

Figura 8. Espectro Electromagnético con énfasis en el espectro visible.

Fuente: Tomado de (Navarro, 2007).

2.1.4.2. Imagen Espectral. Las imágenes espectrales se dividen en imagen multiespectral e hiperespectral, cuya principal diferencia radica en el número de bandas del espectro electromagnético. En los sistemas multiespectrales se obtiene valores de intensidad en las longitudes de onda discretas y en los sistemas hiperespectrales se abarca el espectro continuo, es decir, todas las bandas del espectro electromagnético, de esta manera se obtiene la firma espectral del objeto de análisis en función de la reflectancia respecto a cada longitud de onda. (ElMasry, 2010).

2.1.4.3. Imagen Hiperespectral. Las imágenes hiperespectrales, como se mencionó previamente, son una clasificación de las imágenes espectrales, cuyo rango cubre todo el espectro electromagnético. Con estas imágenes se logra acumular información de los espectros electromagnéticos producto de la incidencia de la luz sobre los cuerpos, conocida como reflectancia. Los espectros son invisibles para el ojo humano a excepción del rango del rango del espectro visible que conforma las imágenes RGB. El estudio de las imágenes hiperespectrales resulta muy útil ya que permite conocer la variación en la composición en una serie de muestras.

La imagen hiperespectral, conocida igualmente como espectrometría de imagen; es una práctica cada vez más abordada, ésta mezcla las ventajas del enfoque por computador habitual y la espectroscopia para conseguir información espacial y espectral de un cuerpo simultáneamente. (Pu Y. Y., 2015)

2.1.5. *Firma Espectral*

Debido a la diferencia en la estructura química y física de los cuerpos, la capacidad de dispersión de la luz varia, como resultado la emisión y absorción de energía electromagnética se da a distintas longitudes de onda, generando así una firma espectral una huella única de los objetos, dependiendo de la materia y condiciones. (ElMasry et al., 2009)

2.1.6. *Instrumentación para medición de parámetros*

Cámara hiperespectral Resonon PIKA – II

El modelo Resonon Pika – II está constituido un sistema integral de hardware y software que permite la captura y el análisis de imágenes hiperespectrales, cuyo rango en el espectro electromagnético incluye al espectro visible y parte del infrarrojo cercano, tomando así un rango de longitudes de onda desde 400nm a 900 nm. Esta cámara realiza la captura por medio de un escaneo en línea. (Maza G. V., 2018).

El equipo Pika II (ver Figura 11) es un dispositivo de fácil uso, compacto y de gran capacidad, permite obtener imágenes de alta calidad, buena proporción de señal y ruido también presenta ligeras distorsiones y presencia de muy ligera luz parásita. (Maza G. V., 2018).

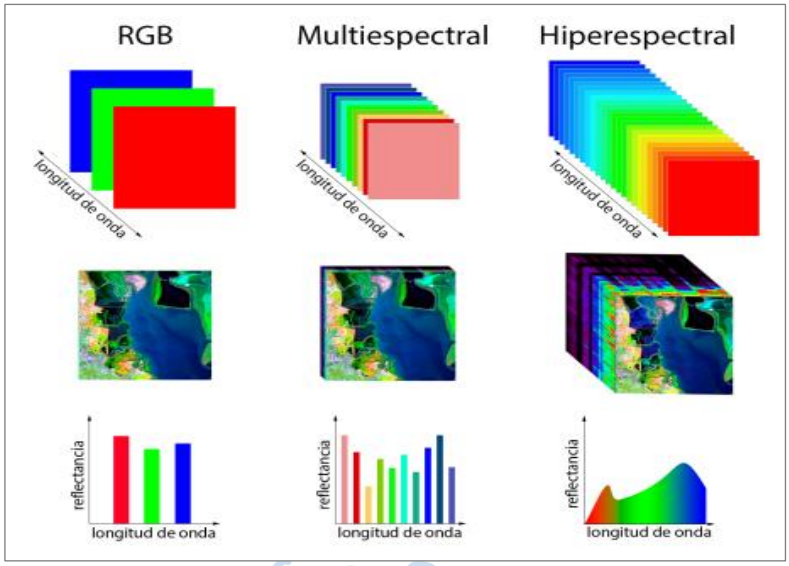

Figura 9. Comparativa entre distintos tipos de imágenes en función del número de bandas espectrales capturadas.

Fuente: Tomado de (Municio Durán, s. f.).

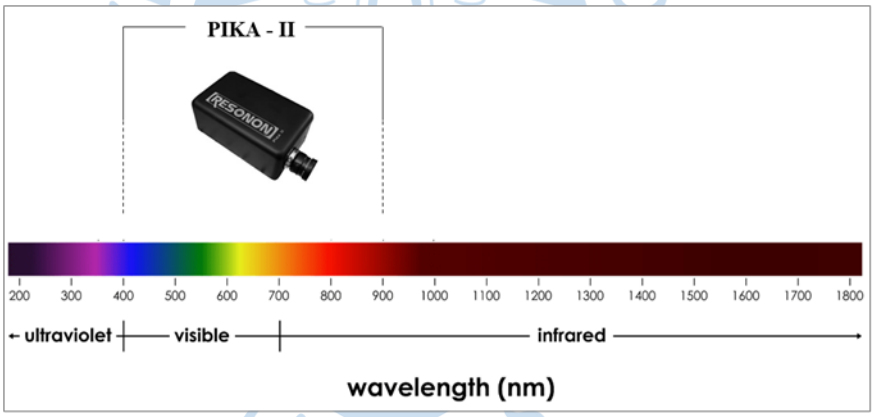

Figura 10. Rango en el espectro electromagnético de la cámara Pika-II.

Fuente: Elaboración propia a partir de (Resonon, s.f.).

Las imágenes hiperespectrales generadas presentan alta calidad de imagen debido a la alta intensidad luminosa, bajas distorsiones, alta relación de señal a ruido (SNR, *Signal-to-Noise Ratio*) y escasa luz difusa (Ruiz, 2016).

Figura 11. Cámara hiperespectral modelo Resonon PIKA – II.

Fuente: Elaboración propia.

La toma de imagen se realiza a modo de escaneo, a medida que el material es desplazado por el stage control. La imagen hiperespectral presenta píxeles, considerados como vectores, donde cada uno puede ser visto como una firma espectral o "huella digital" del material subyacente en el píxel (Gonzáles, 2012). La firma espectral indica la relación entre la reflectancia según cada longitud de onda, en esta investigación se trabajará con la firma espectral promedio de un conjunto de píxeles de una determinada región de la imagen hiperespectral de cada cubo hiperespectral según la muestra evaluada. El análisis se desarrolla mediante el software SpectrononPro (ver Figura 12), cuya interfaz permite visualizar la firma espectral y la imagen en formato RGB y en diferentes longitudes de onda.

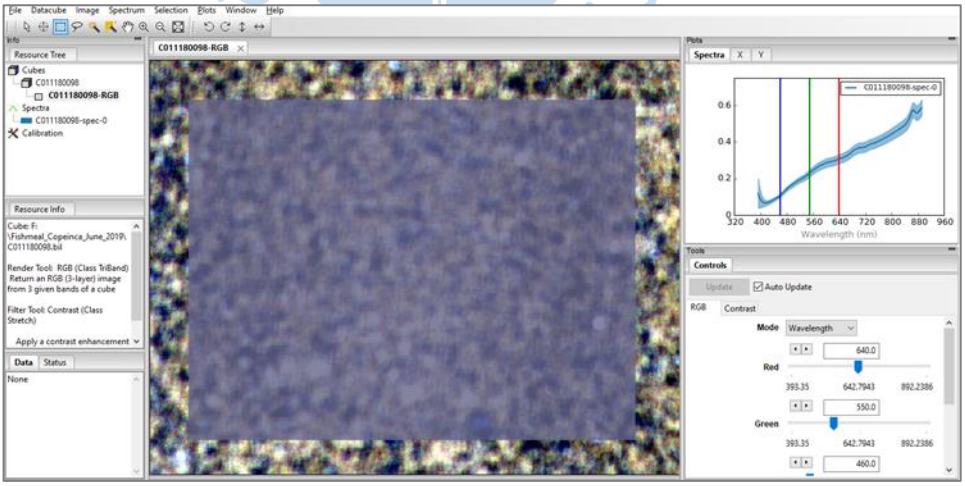

Figura 12. Firma espectral promedio de región de píxeles de la imagen hiperespectral.

Fuente: Elaboración propia.

La cámara incluye una cámara CCD (*charge-coupled device*) de alto rendimiento (López, 2019), inventado a mediados de los años 60 en los laboratorios Bell, quienes descubrieron su potencial para captar la luz y convertirla en una señal eléctrica, dicha cámara cuenta con filtros infrarrojo y RGB. Además, cuenta con un espectrógrafo unido a un lente zoom encargado de medir la radiación electromagnética de la materia en un rango del espectro electromagnético como se indicó previamente. El sistema incluye una fuente iluminación con 4 lámparas halógenas constituidas por cuarzo, en lugar de vidrio, debido a que soporta elevadas temperaturas como la generada por el ciclo halógeno que permite alta luminosidad, reducción de tamaño de las lámparas, mayor vida útil por tratarse de un ciclo y proporciona iluminación adecuada en todas las longitudes de onda para adquirir datos hiperespectrales de alta calidad, las luces son controladas por una fuente de alimentación estabilizada que minimiza la variación debido a las fluctuaciones de iluminación (Resonon Inc, 2020). Presenta una plataforma de desplazamiento (*stage*) y una torre de montaje de aluminio acanalada para graduar la posición del soporte de la cámara y las lámparas (Viera-Maza, 2018, p.54).

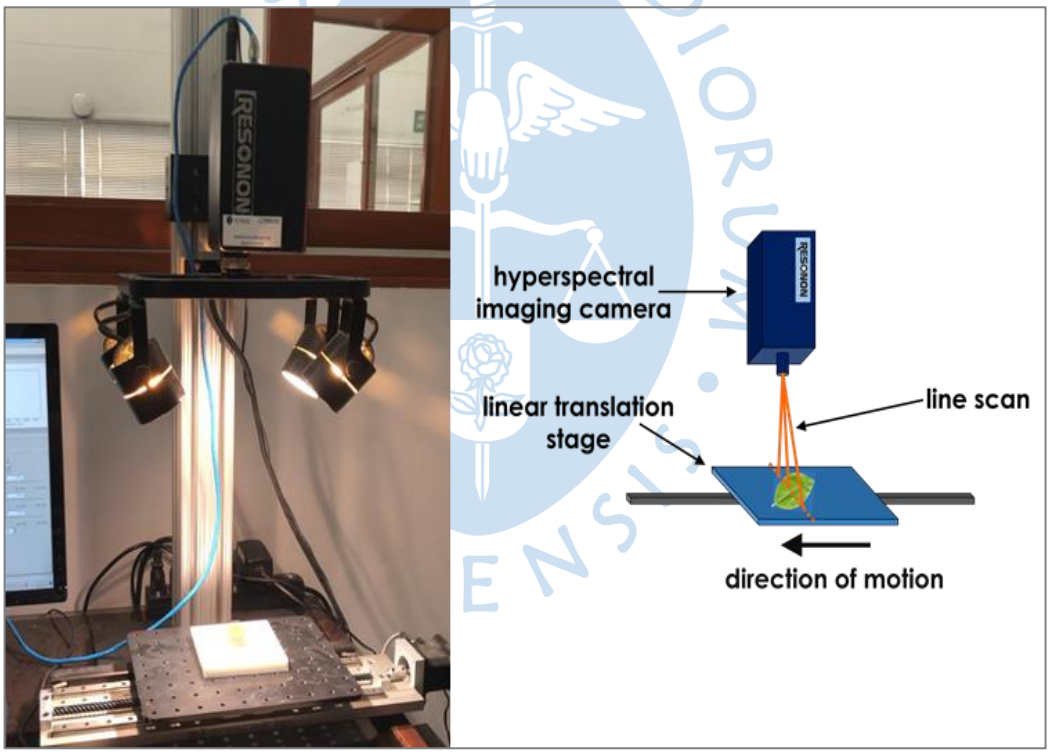

Figura 13. (a) Sistema Resonon PIKA - II en funcionamiento. (b) Representación de la captura en línea de una imagen hiperespectral.

Fuente: (a) Elaboración propia. (b) Tomado de (Resonon, s.f.-b).

A continuación, se detallará las especificaciones técnicas de operación del sistema.

Tabla 3. *Especificaciones técnicas de la cámara hiperespectral Resonon Pika-II*

Parámetro	Unidades	Valor
Spectral Range	nm	400 - 900
Spectral Resolution	nm	2.1
Spectral Channels	-	240
Spatial Channels	-	640
Max Frame Rate	Fps	145
Bit Depth	-	12
Connection Options	-	GigE
Power Requirements	V, W	8-30 V, <2.5 W
Weight	lbs/kg	2.8 / 1.3
Dimensions	in/cm	3.8 x 6.6 x2.5 / 9.7 x 16.8 x 6.4
Operating Temperature Range	F/C	46-90 / 8-32
f/#	-	f/3.0
Average RMS Spot Radius	µm	7
Smile, Peak-to-Peak	µm	5
Keystone, Peak-to-Peak	µm	7
Pixel Size	µm	7.4

Fuente: Elaboración propia a partir de (Resonon, s.f.-a).

2.1.7. *Machine Learning*

Tras el surgimiento de la Inteligencia Artificial se crearon varios métodos lograr que las computadoras aprendan por sí solas, el más conocido actualmente es el Aprendizaje Automático (*Machine learning*). (Amores, 2017, p.25)

El aprendizaje automático consiste en que las computadoras adapten sus acciones de tal forma que se vuelvan cada vez más precisas (Marsland, 2015) para cumplir un determinado propósito. En *machine learning* se tiene un modelo definido por unos parámetros y se busca optimizar los parámetros predecidos por el modelo, es decir, que exista el mínimo error entre las variables predecidas y las reales (Alpaydin, 2020). Con el fin de incrementar el rendimiento, el aprendizaje automático se realiza utilizando un conjunto de datos o experiencia pasada.

2.1.7.1. Sistemas de aprendizaje automático. En *machine learning* existen cuatro métodos de aprendizaje automático según la forma de desarrollar el algoritmo, es decir, la manera en que el sistema aprende de acuerdo con el tipo de supervisión que requiere. A continuación, se detallarán las características de cada uno.

2.1.7.1.1. *Sistema de aprendizaje automático supervisado.* El sistema de aprendizaje automático supervisado está compuesto de una serie de algoritmos que construyen modelos matemáticos a partir de un determinado grupo de datos que contienen tanto variables de

entrada (*inputs*) como de salida (*outputs*). Los datos de entrada son llamados datos de entrenamiento (*training data*) (Mining, 2020). En este método de aprendizaje automático todos los datos de entrenamiento proporcionados incluyen sus respectivos valores de salida conocidos como etiquetas (*labels*)**.**

El algoritmo utiliza la optimización iterativa, es decir, a través de iteraciones se optimiza el modelo de una función objetivo, cuya finalidad es predecir parámetros de salida establecidos. Una vez definido el modelo, este es asociado a nuevos valores de entrada (*validation data*) con la finalidad de validar el mismo. (Mining, 2020)

El aprendizaje supervisado puede ser realizado mediante clasificación o regresión algorítmica (Mining, 2020). La clasificación algorítmica es aplicada para generar outputs restringidos a un conjunto limitado de valores, es decir, agrupados según la salida. Por otro lado, en este proyecto se utilizará regresión algorítmica, donde cada output corresponde a un determinado valor según el input. En esta investigación se trabajará con cinco outputs para cada input correspondientes a proteína, humedad, grasa, arena y cenizas de la harina de pescado mediante algoritmos de Redes Neuronales.

Algunos algoritmos de regresión pueden ser utilizados en clasificación como es el caso de la Regresión Logística (*Logistic Regression*), comúnmente utilizada para clasificación debido a que puede generar un valor que corresponda a la probabilidad de encontrarse dentro de clase determinada. (Géron, 2017, p.26)

2.1.7.1.2. ***Sistema de aprendizaje automático no supervisado.*** Esta metodología de *machine learning* es una herramienta muy útil al momento de detectar relaciones desconocidas dentro de un conjunto de datos sin conocer la respuesta del sistema, es decir, los datos utilizados en el aprendizaje no supervisado no están asignados a variables de salida (Gutierrez, 2015)**.**

El objetivo es hallar regularidades dentro del grupo de datos de entrada (Alpaydin, 2020). Por ello, su aplicación principal es identificar patrones desconocidos en un conjunto de datos mediante la agrupación de datos similares o la detección de valores atípicos (Gutierrez, 2015). Estos mecanismos de clasificación se basan en la búsqueda de las clases con suficiente separabilidad mediante algoritmos que permiten realizar la categorización, para conseguir diferenciar unos objetos de otros. (Amores, 2017).

Uno métodos más utilizados es el *clustering* o agrupamiento, donde el objetivo es encontrar patrones para generar grupos dentro de los datos de entrada (Alpaydin, 2020, p.11), uno de los pasos más importantes en este algoritmo es definir la cercanía en los datos.

2.1.7.1.3. ***Sistema de aprendizaje automático mixto o semi supervisado.*** El sistema de aprendizaje automático semi supervisado resulta de la mezcla del aprendizaje supervisado y el no supervisado, es decir, no todos los datos de entrada son asignados un valor de salida.

La aplicación de este sistema se debe a que no siempre es posible o práctico etiquetar toda la data requerida para el desarrollo del algoritmo**.**

Se busca mejorar la ejecución de los métodos de aprendizaje supervisado utilizando conjunto de datos no etiquetados. Como se explicó en el aprendizaje supervisado, la data etiquetada es utilizada para aprender desde las características de los datos de entrada, los valores deseados de salida, sin embargo, la computadora no puede aprender dicho proceso con datos no etiquetados debido a la ausencia de outputs, pero estos sí proveen información sobre la distribución de los datos de entrada (SemiSaeed, 2018). La información proporcionada es útil para desarrollar un modelo siempre que la distribución de las características de los datos etiquetados y no etiquetados sean relevantes para el problema de clasificación, en este caso los datos no etiquetados mejorarán el rendimiento del aprendizaje supervisado.

Etiquetar millones de datos no solo lleva mucho tiempo, sino que también puede ser extremadamente costoso. Como afirma Smith (2019): "ofrecer una modesta colección de datos etiquetados durante el proceso de aprendizaje y probar con datos no etiquetados parece producir resultados más efectivos y precisos" siempre que se cumpla lo anteriormente establecido.

2.1.7.1.4. ***Sistema de aprendizaje automático reforzado.*** En algunas aplicaciones, la salida de un sistema es una secuencia de acciones. En tal caso, una sola acción no resulta importante; lo es la secuencia de correctas acciones para lograr el objetivo. El programa debería ser capaz de aprender de la secuencia de acciones para generar una política que lo llevará a maximizar la señal de recompensa (Alpaydin, 2020, p.12). La retroalimentación es necesaria para permitir a la máquina saber si está progresando o no y redirigir su camino cuando sea necesario (Smith, 2019). Tales métodos de aprendizaje automático son llamados algoritmos de aprendizaje por reforzamiento (reinforcement learning)**.**

El concepto base de este método de machine learning es muy similar al aprendizaje no supervisado en el que se le otorga una gran parte del control al software y las máquinas para determinar cuál sería la acción adecuada y como consecuencia generar un modelo, pero en vez de enfocarse en encontrar clases dentro de un conjunto de datos, el aprendizaje por reforzamiento se centra en aumentar la señal de recompensa que permita alcanzar el objetivo.

Se aplica cuando los modelos exactos no son factibles. El uso más común del aprendizaje por reforzamiento es en juegos donde hay jugadores simulados por la computadora y juega contra oponentes humanos. En estos "jugadores de computadora", el aprendizaje por reforzamiento les permite responder de una manera que no es exacta y precisa en todo momento, sin embargo, desafía al ser humano por ser poco predecible (Mining, 2020). También es utilizado para el desarrollo de autos autónomos.

2.1.8. *Redes neuronales artificiales (RNA)*

Una red neuronal artificial es un sistema computacional inspirado en una red neuronal biológica y busca modelar las características más relevantes de la red neuronal biológica.

Formalmente podemos definir a una red neuronal artificial como "redes interconectadas masivamente en paralelo de elementos simples (usualmente adaptativos) y con organización jerárquica, las cuales intentan interactuar con los objetos del mundo real del mismo modo que lo hace el sistema nervioso biológico." (Basogain Olabe, 2005)

Una de las características del cerebro que se desea emular es la del aprendizaje adaptativo, es decir que, mediante el entrenamiento de nuestra red, esta pueda ser capaz de detectar y diferenciar patrones. (Matich, 2001) Esto lo hacen mediante el autoajuste de los pesos asignados a cada valor de entrada en una neurona, cada peso determina que tanto influye ese valor en la neurona, mientras mayor sea el peso, mayor influencia tendrá la neurona anterior sobre la neurona siguiente de la red.

Otra característica importante es la de auto organización. "Las redes neuronales emplean su capacidad de aprendizaje adaptativo para auto organizar la información que reciben durante el aprendizaje y/o la operación. Mientras que el aprendizaje es la modificación de cada elemento procesal, la auto organización consiste en la modificación de la red neuronal completa para llevar a cabo un objetivo específico." (Matich, 2001)

La RNA organiza de forma automática la información que recibe durante el proceso de aprendizaje, modificando toda la red neuronal, hasta alcanzar la organización requerida.

Mediante la auto organización, la RNA puede responder apropiadamente ante datos o situaciones a la que no se le había expuesto anteriormente, a esto se le conoce como generalización. Al generalizar podemos llegar a una conclusión a pesar de que los datos de entrada no sean muy claros o si los datos de entrada estan incompletos, esto nos permite una tolerancia a fallos. (Matich, 2001)

Las RNA poseen un margen tolerancia respecto a errores en los datos, es decir, tienen la capacidad de aprender a reconocer patrones distorsionados, incompletos o con ruido, esta es la característica de tolerancia de fallos o abstracción de datos. Si es que se altera parte del algoritmo, Las RNA pueden seguir funcionando, aunque con resultados menos precisos, a la pérdida de neuronas en la red se le conoce como degradación de la red. (Matich, 2001)

2.1.8.1. Estructura de la RNA. "Cada neurona es una simple unidad procesadora que recibe y combina señales desde y hacia otras neuronas. Si la combinación de entradas es suficientemente fuerte la salida de la neurona se activa. El cerebro consiste en uno o varios billones de neuronas densamente interconectadas. El axón (salida) de la neurona se ramifica y está conectada a las dendritas (entradas) de otras neuronas a través de uniones llamadas

sinapsis. La eficacia de la sinapsis es modificable durante el proceso de aprendizaje de la red." (Basogain Olabe, 2005)

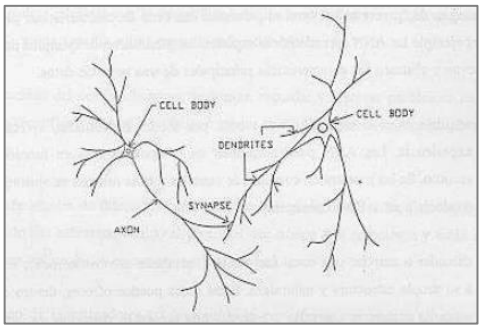

Figura 14. Neurona biológica.

Fuente: Tomado de (Basogain Olabe, 2005).

La neurona recibe "estímulos" en su entrada (input) y cuando llega a un determinado umbral de activación, la neurona se activa, trasmitiendo una señal hacia el axón. Al familiarizarse con el trabajo de una neurona biológica se puede representar el modelo de la neurona artificial.

2.1.8.2. Funcionamiento de la Neurona artificial

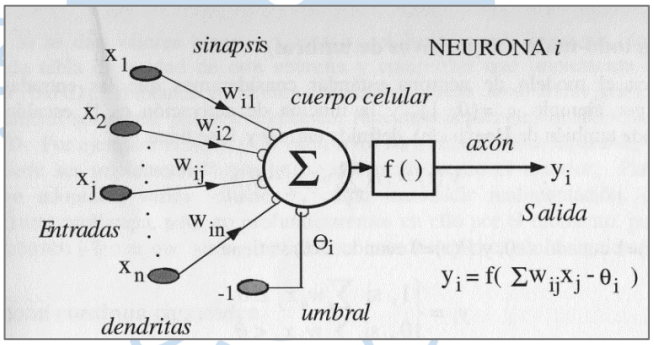

Figura 15. Modelo matemático de una neurona.

Fuente: Tomado de (Basogain Olabe, 2005).

Los valores X representan los valores de entrada en la neurona o inputs. La variable W representa el peso asignado a cada input X, el subíndice i representa el número de la neurona dentro de toda la red neuronal y el subíndice que lo acompaña indica la neurona a la que le corresponde ese peso.

El valor independiente o bias es representado como θ_i, este índice el umbral que debe superar la neurona para que la información pase hacia la siguiente neurona. $f(\sum W_{ij} X_j - \theta_i)$ es una composición de funciones, $f( )$ es la función de activación por la que pasa la neurona para evitar la linealización de toda la red.

2.1.8.2.1. *La función de activación.* "Conviene destacar que la mejora de las redes multicapa estriba en la función de activación no lineal entre capas, pudiéndose llegar al caso de diseñar una red de una capa simple equivalente a una red multicapa si no se utiliza la función no lineal de activación entre capas." (Basogain Olabe, 2005)**.**

Si no utilizamos una función de activación no lineal, toda la red se podría reducir a una solo función lineal, por lo tanto, a una sola neurona. La función de activación es importante porque permite resolver problemas linealmente no separables.

Las funciones de activación más comunes son la sigmoide, tangente hiperbólica y la unidad lineal rectificada (ReLu).

2.1.8.2.2. *Organización de la red neuronal artificial.* Una red neuronal artificial no es más que una red conformada por neuronas interconectadas unas con otras de manera sucesiva, estructuradas de forma vertical en capas**.**

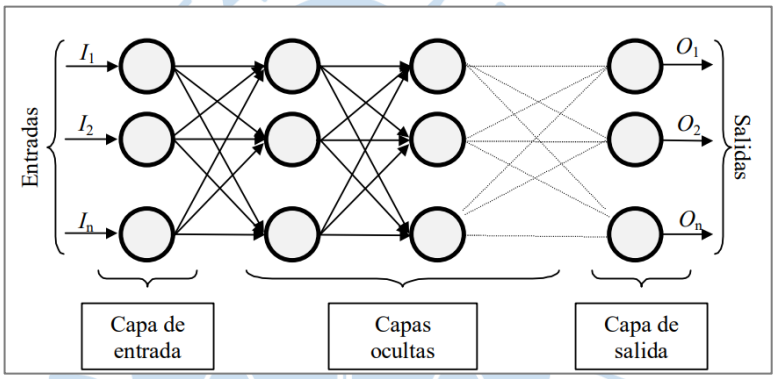

Figura 16. Organización de una red neuronal artificial multicapa.

Fuente: Tomado de (Basogain Olabe, 2005).

La figura anterior muestra una red neuronal artificial multicapa ya que posee más de una capa oculta. Todas las neuronas, a excepción de las neuronas en la capa de entrada, reciben información de las neuronas anteriores.

La red neuronal multicapa que usaremos es una *feedforward* la cual "se caracteriza por ser supervisado pues los parámetros de la red, conocidos como pesos, son estimados a partir de un conjunto de patrones de entrenamiento compuesto por patrones de entrada y salida." (Salas, 2004).

Capa de entrada: la primera columna de neuronas (capa de entrada) se encargará de recibir la información de entrada (inputs o entradas).

Capas ocultas: la información de entrada es procesada por las capas ocultas, se le asignará un peso a cada input y será multiplicada por la función de activación correspondiente, luego este proceso se repite en cada capa oculta hasta llegar a la capa de salida.

Capas de salida: en esta capa, para una red neuronal con aprendizaje supervisado, el valor de salida será comparada con el valor real de salida y mediante el algoritmo de *backpropagation* se corregirá el error, corrigiendo el valor de los pesos de cada neurona.

Algoritmo de *Backpropagation*

"El algoritmo *Backpropagation* se encuentra dentro de los algoritmos de aprendizaje supervisado, de arquitectura multicapa y con conexiones hacia adelante." (Samaniego, 2009).

Mediante el algoritmo de *backpropagation*, el error resultante será propagado "hacía atrás", asignándole un porcentaje del error total a cada neurona de acuerdo con la influencia de esa neurona en la neurona subsiguiente, propagando así el error hasta la primera capa. Luego se modificarán los pesos de acuerdo al porcentaje de error que posea.

Explicación matemática del algoritmo de *backpropagation* en una red neuronal perceptrón multicapa

Como ya se detalló anteriormente, una neurona no es más que una suma ponderada de todas las neuronas comprendidas en la capa anterior más un parámetro llamado ballas.

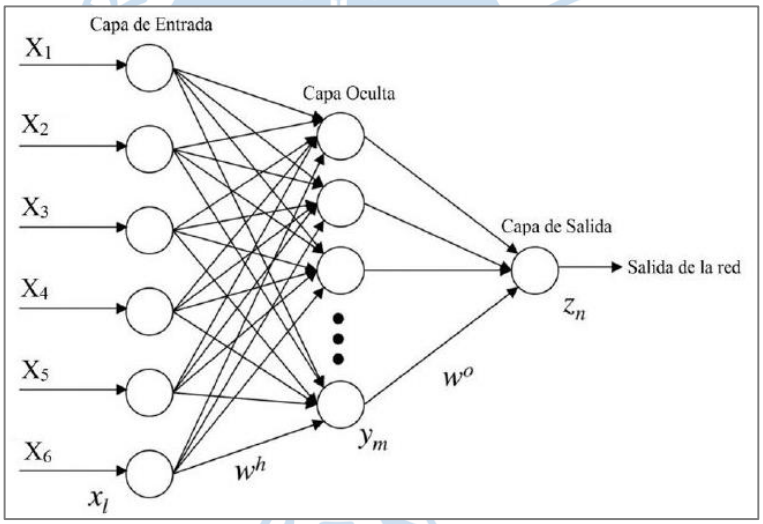

Figura 17. Estructura de una RNA perceptrón multicapa con una neurona de salida.

Fuente: Tomado de (FPUNE Scientific, s.f.).

Este proceso se realiza en toda la red neuronal, por tanto, todas las neuronas de una capa anterior representan los valores de entrada de las neuronas de la capa siguiente.

Los cálculos los realizaremos de manera general para una sola neurona de salida, pero este a partir de este proceso se pueden inferior los cálculos para estructuras con más de una neurona en la capa final o de salida.

Sabemos que en la suma ponderada de la neurona que se encuentra en la última capa puede representarse por la siguiente función matemática:

$$z^n = w_i{}^n a_i{}^{n-1} + b^n$$

Al aplicarle la función de activación obtenemos la salida de la última capa.

$$a^n(z^n)$$

Sabemos que la función de *backpropagation* propaga el error al comparar el parámetro de salida en la red con el valor que deseamos obtener, esto lo realiza mediante la técnica del gradiente descendente. mediante las derivadas parciales, el gradiente descendente busca la zona donde la función de coste se reduzca, y según el nuevo valor hallado, actualiza los pesos y el parámetro de ballas. Entonces lo que se busca son las derivadas parciales de la función de coste.

La función de coste de la última capa se define se define como la siguiente composición de funciones:

$C\big(a(z^n)\big)$ = error de salida

Los valores que constantemente cambian en toda la red son los pesos y los parámetros de ballas, por lo tanto, deberemos encontrar las derivadas parciales de la función de error respecto a ellos.

Al aplicar el método de la regla de la cadena, obtenemos las siguientes derivadas parciales:

$$\frac{\partial C}{\partial w^n} = \frac{\partial C}{\partial a^n} \cdot \frac{\partial a^n}{\partial z^n} \cdot \frac{\partial z^n}{\partial w^n}$$

$$\frac{\partial C}{\partial b^n} = \frac{\partial C}{\partial a^n} \cdot \frac{\partial a^n}{\partial z^n} \cdot \frac{\partial z^n}{\partial b^n}$$

Como función de coste usamos el error cuadrático medio, por la primera derivada parcial quedaría de la siguiente forma:

$$C(a_i{}^n) = \frac{\sum_i (y_i - a_i{}^n)^2}{2}$$

$$\frac{\partial C}{\partial a_i{}^n} = (a_i{}^n - y_i)$$

La segunda derivada parcial la obtenemos al derivar la función de activación que utilizaremos en la última neurona. Usaremos la función sigmoide como función de activación para todas las neuronas de la red, por lo tanto, el resultado es el siguiente:

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

$$\frac{\partial a^n}{\partial z^n} = a^n(z^n)\big(1 - a^n(z^n)\big)$$

La derivada parcial de la función de error respecto a la suma ponderada nos indica que tanta responsabilidad posee una neurona en el resultado final obtenido. Reduciremos esto a siguiente expresión.

$$\frac{\partial C}{\partial z^n} = \frac{\partial C}{\partial a^n} \cdot \frac{\partial a^n}{\partial z^n} = \delta^n$$

Las terceras derivadas de nuestra composición de funciones las obtenemos de la siguiente forma:

$$z^n = {w_i}^n {a_i}^{n-1} + b^n$$

$$\frac{\partial z^n}{\partial w^n} = {a_i}^{n-1} \qquad \frac{\partial z^n}{\partial b^n} = 1$$

Por lo tanto, las derivadas parciales de la función de error en la última capa quedarían expresadas de la siguiente forma:

$$\frac{\partial C}{\partial w^n} = \delta^n \cdot \frac{\partial z^n}{\partial w^n}$$

$$\frac{\partial C}{\partial b^n} = \delta^n \cdot \frac{\partial z^n}{\partial b^n}$$

Ahora analizaremos las derivadas parciales de la función de error respecto a los pesos y los parámetros de ballas en la capa anterior, es decir, en la capa n-1. La función de error de salida al analizar la capa n y n-1 quedaría definida de la siguiente forma:

$$C\big(a^n(w^n \cdot a^n(w^{n-1} \cdot a^{n-2} + b^{n-1}) + b^n)\big)$$

$$\frac{\partial C}{\partial w^n} = \frac{\partial C}{\partial a^n} \cdot \frac{\partial a^n}{\partial z^n} \cdot \frac{\partial z^n}{\partial a^{n-1}} \cdot \frac{\partial a^{n-1}}{\partial z^{n-1}} \cdot \frac{\partial z^{n-1}}{\partial w^{n-1}}$$

$$\frac{\partial C}{\partial b^n} = \frac{\partial C}{\partial a^n} \cdot \frac{\partial a^n}{\partial z^n} \cdot \frac{\partial z^n}{\partial a^{n-1}} \cdot \frac{\partial a^{n-1}}{\partial z^{n-1}} \cdot \frac{\partial z^{n-1}}{\partial b^{n-1}}$$

Podemos observar que al analizar una capa anterior aparecen las derivadas parciales que ya analizamos anteriormente. Por lo tanto, reduciremos la derivada parcial de la función de error respecto a los pesos y los parámetros de ballas a una expresión general.

$\frac{\partial C}{\partial a^n} \cdot \frac{\partial a^n}{\partial z^n} = \delta^n$ Esta expresión anterior indica la variación de la función de error que se da única y exclusivamente en la última capa de nuestra red neuronal de una sola neurona de salida.

$\frac{\partial z^n}{\partial a^{n-1}}$ Esta derivada nos indica que tanto varía la suma ponderada en la neurona de la capa que estamos analizando respecto a las neuronas de la capa anterior. Expresaremos esta derivada de la siguiente forma:

$$\frac{\partial z^n}{\partial a^{n-1}} = P^n$$

$\frac{\partial a^{n-1}}{\partial z^{n-1}}$ Derivada parcial que representa que tanto influyen los pesos y los parámetros de ballas de las neuronas anteriores respecto a la salida que esa misma neurona.

Las últimas derivas representan la influencia de los pesos y los parámetros de ballas sobre la suma ponderado, y por tanto sobre la salida de ella misma.

$$\frac{\partial z^n}{\partial w^n} = {a_i}^{n-2} \qquad \frac{\partial z^n}{\partial b^n} = 1$$

Podemos expresar la influencia de los parámetros de ballas y pesos penúltima capa sobre la función de coste o error de la siguiente forma:

$$\frac{\partial C}{\partial w^n} = \delta^n \cdot P^n \cdot \frac{\partial a^{n-1}}{\partial z^{n-1}} \cdot {a_i}^{n-2}$$

$$\frac{\partial C}{\partial b^n} = \delta^n \cdot P^n \cdot \frac{\partial a^{n-1}}{\partial z^{n-1}} \cdot 1$$

Las tres primeras expresiones determinan la responsabilidad de la neurona anterior sobre la siguiente, por tanto, la expresión que le correspondería seria δ^(n-1), por tanto, podemos reducir aún más la expresión general hallada.

$$\delta^{n-1} = \delta^n \cdot P^n \cdot \frac{\partial a^{n-1}}{\partial z^{n-1}}$$

$$\frac{\partial C}{\partial w^n} = \delta^{n-1} {a_i}^{n-2}$$

$$\frac{\partial C}{\partial b^n} = \delta^{n-1}$$

Estas expresiones nos indican cuanto influye una neurona, ubicada en cualquier capa de nuestra red, en el cálculo valor de salida de la red, por tanto, podemos “penalizar” esta neurona en mayor o menor medida. Entonces, a partir de estas expresiones podemos conocer como propaga el error hacia atrás una red neuronal perceptrón multicapa.

La propagación del error se realiza en tres procesos:

I. Se debe obtener el error que se produce en la última capa. Esto lo hacemos al aplicar las derivadas parciales para analizar que tanto influyen solo los parámetros de la última capa.

$$\delta^n = \frac{\partial C}{\partial a^n} \cdot \frac{\partial C}{\partial z^n}$$

II. Se retro propaga el error obtenido en cada parámetro hacia las neuronas de la capa anterior. Esto lo hacemos con las derivadas parciales que describen la influencia de la capa anterior sobre la siguiente.

$$\delta^{n-1} = \delta^n \cdot P^n \cdot \frac{\partial a^{n-1}}{\partial z^{n-1}}$$

$$P^n = \frac{\partial z^n}{\partial a^{n-1}}$$

III. Finalmente, se calcula las derivadas de la de la capa que queremos analizar. Se le asignará un nuevo valor de acuerdo con la influencia que esta tiene sobre las demás por medio de sus derivadas parciales.

$$\frac{\partial C}{\partial b^n} = \delta^{n-1}$$

$$\frac{\partial C}{\partial w^n} = \delta^{n-1} {a_i}^{n-2}$$

Método del gradiente de máximo descenso

"Este algoritmo iterativo que permite encontrar un mínimo local de una función multivariada no lineal sin restricciones, mediante aproximaciones sucesivas. La búsqueda de la solución sigue la dirección del gradiente descendente más pronunciado hasta llegar a su menor valor." (Ojeda, 2016)

El algoritmo de *backpropagation* usa el método de gradiente de máximo descenso para hallar el punto de menor valor de la función de la red neuronal artificial, el cual es donde el algoritmo converge. Lo hace de forma iterativa modificando los pesos de cada neurona.

Matemáticamente podemos se puede definir como:

$$\theta_n = \theta_{n-1} - \alpha * \nabla f$$

θ_{n-1}: Parámetros que definen la función de error antes de ser corregido.

θ_n: Parámetros que definen la función de error corregido.

∇f: Gradiente de la función de error.

α: Ratio de aprendizaje

2.1.8.2.3. ***Ratio de aprendizaje.*** Valor numérico que multiplica al gradiente, este indica que tan rápido queremos que el algoritmo se acerque hasta el valor que proporcione el mínimo error o coste**.**

Usaremos como ejemplo la siguiente función de error:

$$f(x, y) = (x^2 + y - 11)^2 + (x + y^2 - 7)^2$$

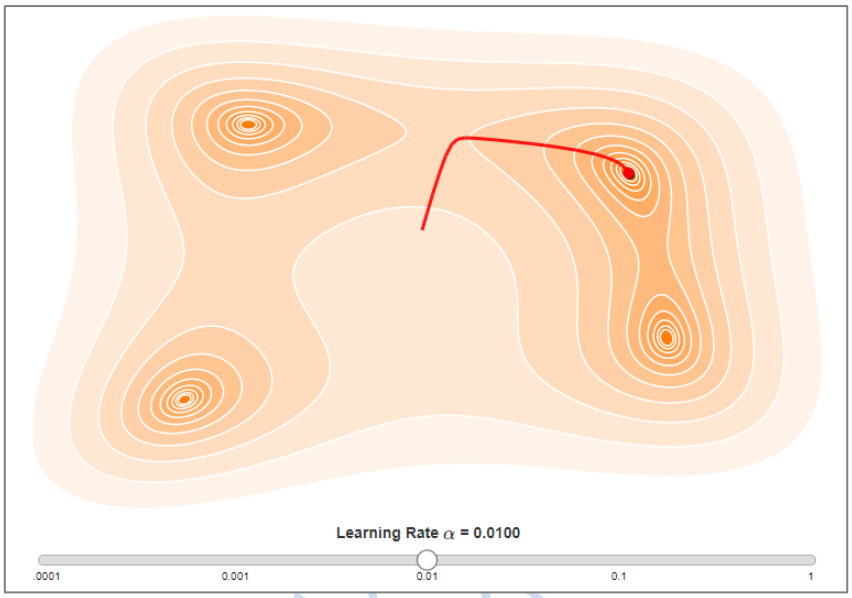

Figura 18. Función de error con un ratio de aprendizaje de 0.01.

Fuente: Tomado de (Ben Frederickson, 2016).

Para un ratio de aprendizaje de 0.01, la función converge usando el método de gradiente máximo luego de 31 iteraciones.

Al aplicarle un ratio de aprendizaje mayor, por ejemplo, un ratio de 0.03, observamos que ya no converge y entra en un bucle buscando el mínimo local.

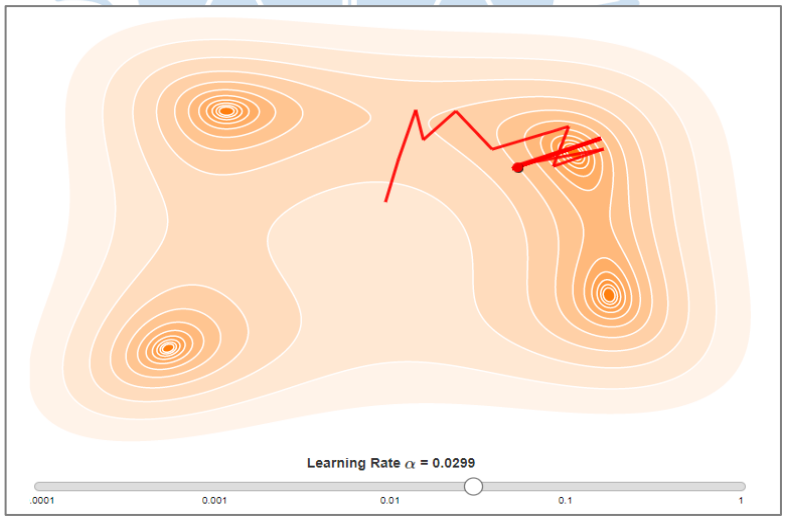

Figura 19. Función de error con un ratio de aprendizaje de 0.03.

Fuente: Tomado de (Ben Frederickson, 2016).

Al aumentar el ratio de aprendizaje, el descenso de gradiente no logra entrar en la zona de mínimo local, lo que ocasiona un mal funcionamiento del algoritmo de *backpropagation*. Por ello, es muy importante elegir un ratio de aprendizaje adecuado al momento de entrenar la red neuronal.

Algoritmos que optimizan el *Learning Rate* (ratio de aprendizaje)

Existen algoritmos que solucionan el problema del ratio de aprendizaje, asignándole diferentes valores al ratio de aprendizaje durante el proceso de *backpropagation*. Estos algoritmos de optimización matemática son los siguientes:

- SDG
- Momentum
- NAG
- Adagrad
- Adadelta
- Rmsprop

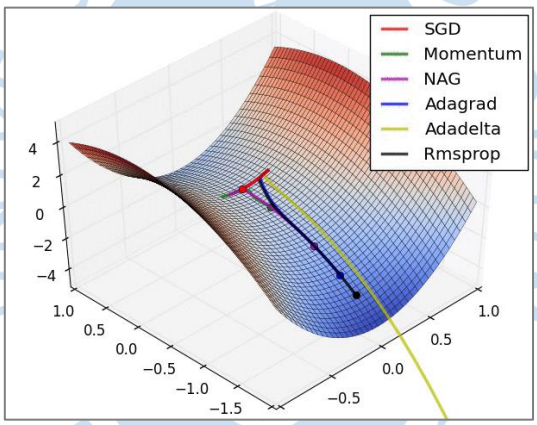

Figura 20. Comportamiento de los algoritmos de optimización en un punto de silla.

Fuente: Tomado de (Sebastian Ruder, 2016).

2.1.8.3. Modelos de Red Neuronal Artificial (RNA). A continuación, se explicarán las principales características de algunos modelos de redes neuronales.

2.1.8.3.1. *Red Neuronal Monocapa o Perceptrón simple.* El Perceptrón es un modelo lineal (unidimensional) constituido por dos capas de neuronas, la primera capa está

constituida por las neuronas de entrada y la segunda por neuronas de salida. No presenta capas ocultas y su función de activación es de tipo escalón**.**

Considerando N neuronas de entrada y M neuronas de salida (Flores, Fernández, 2008), La capa de salida es calculada de la siguiente forma:

$$y_i = H\left(\sum_{i=0}^{N} w_{ij}x_j\right), \quad \forall j, 1 \leq j \leq M$$

Este tipo de red neuronal permite clasificar entre dos grupos linealmente separables (mediante compuertas OR), pero no entre grupos linealmente no separables (compuerta OR exclusivo o XOR) (Flores, Fernández, 2008) lo que limita su aplicación a sistemas más complejos.

2.1.8.3.2. ***Red neuronal multicapa o Multiperceptrón.*** La red neuronal multicapa aparte de tener capas de entrada y salida, cuenta con capas intermedias llamadas capas ocultas. (Mejías, Carrasco, Ochoa, Moreno, s.f) Las conexiones entre neuronas dentro de la red solo pueden ir desde una neurona de una capa hacia las neuronas de la capa siguiente, no existe retroalimentación, es decir, la salida de una capa no puede actuar como entrada de una capa anterior. (Villamil, Delgado, 2007)**.**

A continuación, se muestra el esquema de una red neuronal multicapa. Las x_i y y_i hacen referencia a las neuronas de la capa de entrada y salida, respectivamente. Los $w_{i,j}$ representan los pesos y las S las capas de la red neuronal multicapa.

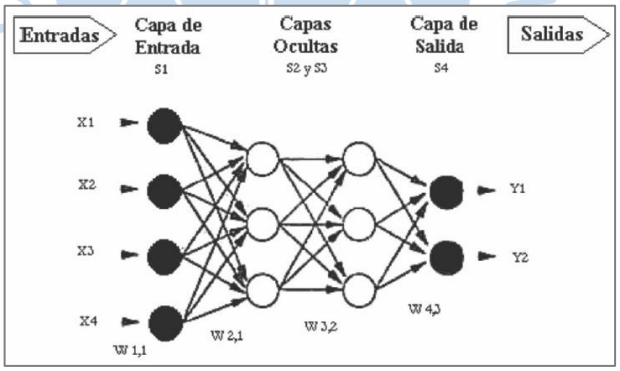

Figura 21. Red neuronal Perceptrón multicapa; Entrenamiento de una red neuronal multicapa para la tasa de cambio euro – dólar.

Fuente: Tomado de (Villamil, J., Delgado, J. 2007).

El entrenamiento de una red neuronal multicapa puede ser de dos formas: incremental o batch, el primero consiste en desarrollar el aprendizaje a medida que se incorpora secuencialmente cada entrada con su respectiva salida y el segundo, en ejecutar el

aprendizaje tras haber incorporado a la red todas las entradas con sus salidas correspondientes. En el proceso de entrenamiento de la red es importante, además de ajustar los pesos y umbrales, optimizar el número de neuronas que la van a conformar ya que de esto depende la velocidad de aprendizaje que adquiera. (Villamil, J., Delgado, J., 2007).

2.1.8.3.3. ***Red neuronal convolucional (CNN).*** La red neuronal convolucional se utiliza para el procesamiento de datos tipo malla, como es el caso de las imágenes que pueden ser consideradas como una malla bidimensional de píxeles (Sánchez, 2018). Este tipo de redes neuronales utiliza la operación matemática llamada convolución**.**

La aplicación de redes neuronales convolucionales implica siempre que las entradas sean en forma de imágenes debido a que detectan patrones espaciales locales en los datos (Távara, 2019). Consiste en ir reduciendo las imágenes, pero aumentando los canales. A medida que se avanza en las capas de la red disminuyen dimensiones incorporando características cada vez más complejas. (Quintero, Merchán, Cornejo, Sánchez-Galán, 2018).

2.2. *Tensorflow*

Tensorflow es una librería de código abierto diseñada por Google Brain especialmente para su implementación en *deep learning*; inicialmente creado con la finalidad de poder diseñar redes neuronales de gran precisión capaces de aprender y distinguir patrones o signos; Tensorflow está basado en el lenguaje C++ y Python, lo cual le permite tener una gran versatilidad en comparación con su antigua versión DistBelief que fue creado en el 2011, esta librería de código cerrado fue de gran uso por Alphabet, la cual es una empresa aliada de Google y especializada en el desarrollo de productos ligados al uso de internet, telecomunicaciones, etc.

Tensorflow hace uso de grafos de flujo de datos, estos son la representación de operaciones matemáticas, así mismo consta con conexiones que representan los tensores que en realidad son conjuntos de datos multidimensionales.

Asimismo, existe Keras es una librería escrita en Python usada en el desarrollo de redes neuronales intentando “imitar” ciertas características, esta librería permite diseñar algoritmos en pocas líneas de código.

Actualmente las actualizaciones realizadas a Tensorflow han permitido obtener una herramienta potente para el desarrollo de redes neuronales, ya que mediante la implementación de las mejores características de Keras se logran obtener algoritmos de mayor precisión y con mejores respuestas en las etapas de entrenamiento, el uso de otras librerías complementarias como Sklearn permiten minimizar tiempo y al mismo tiempo permite tener una mejor comprensión de los distintos comandos usados.

Figura 22. Librerías relacionadas a Tensorfow.

Fuente: Tomado de (Claire, 2020).

2.3. Partial Least Square (PLS)

El análisis de componentes principales es una técnica ligada a los métodos PLS; el análisis de componentes principales nos da a conocer información de un conjunto de datos a partir de una muestra, esta es una técnica muy usada en la estadística descriptiva, ya que permite acercarse mucho a la verdadera información del conjunto de datos considerando la existencia de un sesgo que tiende a ser mínimo; las principales ventajas son:

- Reduce la cantidad de datos presentes para el análisis, permitiendo que el sistema se desarrolle con mayor facilidad y no presente saturación.
- Permite tener una mejor predicción de datos no usados en la clasificación, siempre y cuando estos compartan las principales características del sistema.

Dentro del procesamiento de la data se tiene que existen algoritmos como el PCR o PCA, los cuales estiman la varianza de los componentes principales para poder tener un mejor modelo predictivo, sin embargo, una desventaja para lograr la implementación de estos algoritmos es que solo se puede predecir una variable por modelo.

El algoritmo PLS está basado en el cálculo de la "covarianza de los componentes principales", estas son variables latentes las cuales no son observables directamente, sino más bien que son producto de la inferencia, esto es basándose en características de variables existentes, esto nos da como ventaja la reducción de la cantidad de datos a analizar

A continuación, se describirán las variables usadas en la explicación de PLS:

X: Variables independientes (en forma de matriz).

Y: Variables dependientes (en forma de matriz).

x: Variable independiente (en forma de vector, propio de PLS1).

y: Variable dependiente (en forma de vector, propio de PLS1).

f: El vector error.

b: Coef. de regresión, multiplica a T. (en forma de vector)

W: vector de pesos (*weights*)

w: Matriz de pesos.

E: Matriz residual de variables independientes.

T: representa los resultados como matriz (*scores*), variables latentes.

t: representa los resultados como vector (variables latentes).

$\mathbf{P}^{\mathrm{T}}$: Representa las cargas (*loadings*).

h: número de iteración.

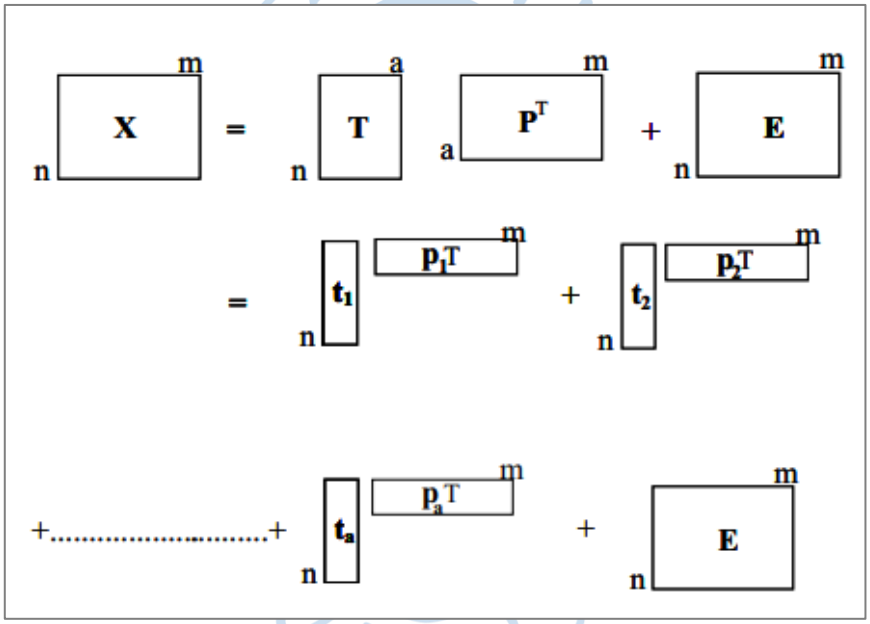

Figura 23. Representación de la matriz X y su descomposición en componentes basadas en variables latentes.

Fuente: PLS regression and its application to coal analysis, 2003.

Teniendo así que, para un modelo, la ecuación característica para una matriz X podrá ser representada por una matriz T con un número menor de columnas y con un cierto error, en donde si se consideran todas las variables aleatorias el error será nulo:

$$X = \mathrm{TP}^{\mathrm{T}} + E$$

Existen dos modelos de regresión de mínimos cuadrados parciales, estos son denominados modelo PLS1 y modelo PLS2, los cuales tienen como principal diferencia el

número de variables dependientes, teniendo así que el modelo PLS1 trabaja con vectores y el modelo PLS2 trabaja con matrices.

El modelo PLS1 tiene como base que las variables latentes de mayor importancia "t" usadas en el desarrollo de la variable "y" son las primeras; además también existen otros vectores usados en la etapa de calibración que vuelven sensible el modelo, estos son W^T y b (sensibilidades). El algoritmo usado en la calibración viene determinado por la relación existente entre "y" y "t", obteniendo así una ecuación de carácter lineal en donde mediante la aplicación de un vector con coeficientes de regresión se busca reducir el vector error:

$$y = \mathrm{T}b + f$$

El algoritmo de calibración viene determinado por una serie de pasos a seguir:

$f_0 = \mathrm{y}$; $E_0 = X$, deben estar centrados y escalados con varianza 1.

1. ${W_h}^{\mathrm{T}} = ({f_{h-1}}^{\mathrm{T}} * {E_{h-1}}^{1})/({f_{h-1}}^{\mathrm{T}} * {f_{h-1}}^{1})$

Normalización

2. ${W_h}^{\mathrm{T}} = \frac{({W_h}^{\mathrm{T}})}{\|{W_h}^{\mathrm{T}}\|}$
3. ${t_h}^{1} = ({E_{h-1}}^{1} * {W_h}^{1})/({W_h}^{\mathrm{T}} * {W_h}^{1})$
4. ${P_h}^{\mathrm{T}} = ({t_h}^{\mathrm{T}} * {E_{h-1}}^{1})/({t_h}^{\mathrm{T}} * {t_h}^{1})$
5. ${t_h}^{1} = ({t_h}^{1}) * \|{P_h}^{\mathrm{T}}\|$
6. ${w_h}^{\mathrm{T}} = ({w_h}^{\mathrm{T}}) * \|{P_h}^{\mathrm{T}}\|$

Normalización:

7. ${P_h}^{\mathrm{T}} = \frac{({P_h}^{\mathrm{T}})}{\|{P_h}^{\mathrm{T}}\|}$
8. ${b_h}^{1} = ({f_h}^{\mathrm{T}} * {t_h}^{1})/({t_h}^{\mathrm{T}} * {t_h}^{1})$

Cálculo de residuales de X:

9. ${E_h}^{1} = ({E_{h-1}}^{1} - {t_h}^{1} * {P_h}^{\mathrm{T}})$

Cálculo de residuales de y:

10. ${f_h}^{1} = ({f_{h-1}}^{1} - {b_h}^{1} * {t_h}^{1})$

Entrada para siguiente iteración:

11. $h = h + 1$

Se logra inferir que "y" es una variable dependiente de los vectores "${w_h}^{\mathrm{T}}$, ${P_h}^{\mathrm{T}}$y b".

Es necesario lograr definir "X" de manera empirica haciendo haciendo uso de los vectores "w´ y p´" y ademas calcular la matriz "T", esto se da mediante un algoritmo de predicción implementado en el PLS1, el cual tiene pasos a seguir:

$E_0 = X$ … haciendo uso del promedio y la desviacion estandar estimados en la etapa de calibración.

1. ${t_h}^1 = ({E_{h-1}}^1 * {W_h}^1)/({W_h}' * {W_h}^1)$
2. ${E_h}^1 = ({E_{h-1}}^1 - {t_h}^1 * {P_h}')$

Donde la entrada para la siguiente iteración es:

3. $h = h + 1$
4. $y = \sum b_h * t_h$

Ventajas y desventajas

- El metodo permite obtener una matriz T conformada con vectores lineales e independientes.
- Permite definir un sistema de tipo ortogonal.
- Permite reducir los datos basandose en sus principales carateristicas
- La posibilidad de obtener un error en las mediciones es baja ya que la muestra de datos usada es muy similar a la general.
- La principal desventaja es que se puede ignorar informacion que puede ser relevante y su descarte no es lo mas indicado.

Capítulo 3
Desarrollo, experimentación y resultados

3.1. Ajuste de hiperparámetros de una red neuronal

Al trabajar con redes neuronales, la fase de entrenamiento es de vital importancia ya que se comprueban el correcto funcionamiento de los parámetros e hiperparámetros usados en el diseño de esta misma; los parámetros son las variables que son usadas para entrenar el algoritmo (80% de los datos totales) mientras que los hiperparámetros son aquellas partes en el diseño de la red neuronal que son seleccionadas para el desarrollo y procesamiento de los datos de entrada, estos son:

3.1.1. *Selección de función de activación*

La función de activación es de vital importancia para el correcto ajuste de nuestros datos, como ya se demostró en el informe 2 se dispone de una amplia variedad de función de activación, sin embargo, para el entrenamiento del algoritmo se tomó en consideración las siguientes observaciones:

Sigmoide: es usada para los modelos de regresión logística, usada para clasificar los datos en dos categorías, entre 1 y 0, su principal inconveniente es que si se tuviera valores de entrada negativos la función se mantiene constante en 0, mientras que si se tiene valores de entrada positivos la función se mantiene constante en 1, de todo esto se puede decir que la función se satura, en el entrenamiento la función de error compara la salida de la neurona con la categoría a la que pertenecen los datos; esto tiene como efecto que no se observen mucha variación en el proceso de clasificación.

Tangente hiperbólica: esta función es similar a la función sigmoidal, pero la diferencia es el rango de valores a la salida va entre -1 y +1, sin embargo, no se soluciona la saturación de datos positivos o negativos ya que ahora ocurrirá en -1 o +1

Relu: la función de activación Relu (*rectified linear unit*) es una función no lineal; considerando 0 los valores negativos de entrada y los valores positivos de entrada los considera iguales a los valores obtenidos a la salida, no existiendo un valor de saturación, ya que no hay un rango que limite la función, la función Relu es la más usada hoy en día gracias a su facilidad de manejo, el principal problema con la función Relu es que en el algoritmo

produce una “muerte neuronal” y es que cuando la neurona alcance el valor 0, deja de aprender.

Leaky Relu: Esta función anula el problema antes mencionado haciendo uso de una pendiente muy pequeña en lugar de un valor constante en 0; gracias a esta variación se considera que para las capas ocultas es recomendable usar la función Relu modificada.

3.1.2. *Número de capas ocultas*

Normalmente una red neuronal contiene entre 1 y 2 capas, variando según el nivel de complejidad de los datos, mientras más capas se añadan se necesitara una mayor capacidad de trabajo por parte del software; además al tener una mayor cantidad de capas, se tendrá un mayor número de neuronas lo cual producirá que se obtenga una mayor cantidad de mínimos locales, si se llegara a aumentar el número de capas se tendrá que seguir la regla piramidal en donde se obtiene que el número de neuronas de la siguiente capa tendrá que ser menor.

Debido al número de datos usados para el entrenamiento la red neuronal y considerando que es un algoritmo que no tiene gran complejidad, se asumirá una sola capa oculta en donde el número de neuronas en esta vendrá definido por la siguiente ecuación:

$$h = (i * o)^{0.5}$$

Donde,

h = n° de neuronas en la capa oculta
i = n° de neuronas en la capa de entrada
o = n° de neuronas en la capa de salida

Cuando se elige usar una capa oculta el valor del número de neuronas (h) obtenido en la ecuación se toma como referencia, a partir de aquí el número de neuronas puede aumentar o disminuir según los resultados obtenidos en el entrenamiento; mientras que, si se tuviera dos capas ocultas, el número de neuronas en cada capa vendrá definido por la siguiente ecuación:

$$h1 = o * (r)^2$$

$$h2 = o * r$$

$$r = \left(\frac{i}{o}\right)^{0.333}$$

Donde,

h 1 = n° de neuronas en la capa oculta 1

h 2 = n° de neuronas en la capa oculta 2

i = n° de neuronas en la capa de entrada

o = n° de neuronas en la capa de salida

Cuando se tiene un bajo contenido de neuronas la red neural es incapaz de aprender ya que en algún punto las neuronas se saturaran lo cual impide que guarden información, es decir pierden la capacidad de adaptarse frente a nuevos datos usados para el entrenamiento de la red, este problema es conocido como *underfitting*; mientras que si se tiene un alto número de neuronas las red neuronal aprende con mucha precisión, la información es dividida en muchas partes lo cual permite el aprendizaje "especializado" de estos datos, perdiendo así capacidad de interpolación frente a un valor que no haya sido usado para el aprendizaje, este problema es conocido como *overfitting*.

3.1.3. *Velocidad de aprendizaje*

La velocidad de aprendizaje viene definida por un valor típico igual a 0.1; sin embargo, lo recomendable es usar valor que oscilan entre 0.01 y 0.99, al tener un valor muy próximo a 0.1, se obtiene una oscilación en os resultados produciendo que no haya convergencia.

3.1.4. *Valor inicial de los pesos*

De manera común los pesos son valores asignados de manera aleatoria que oscilan entre -1 y 1 o entre -0.5 y 0.5; existe una regla opcional usada para determinar estos pesos y es que se encontraran entre los valores de (-1)/n y 1/n , en donde "n" es el número de pesos que serán usados.

3.2. Programación de la Red Neuronal Multicapa

Importar librerías

Para la creación de esta Red Neuronal se usó las librerías de pandas para manipulación y análisis de datos de las longitudes de onda de cada cubo hiperespectral. Usamos la librería de numpy para convertir los datos con lo que contamos en *arrays*, y de esa forma poder trabajar con vectores y matrices. La librería os solo la usamos para exportar el dataset a nuestro programa. Mediante la librería matplotlib graficaremos que tanto disminuye el error cuadrático medio, y de esa poder visualizar de forma gráfica si nuestro modelo llega a converger. La librería más importante es la de tensorflow, con ella vamos a modelar toda nuestra red neuronal multicapa.

```
1. import pandas as pd
2. import tensorflow as tf
3. import numpy as np
4. import os.path
5. import matplotlib.pyplot as plt
```

Importar el *dataset*

Importamos todos los datos con los que contamos. Los parámetros de salida los tenemos en un Excel, estos parámetros son la proteína, grasa, humedad y ceniza del lote de

pescado, los cuales fueron medidos y proporcionados por la certificadora. Nos proporcionaron los datos de entrada (longitudes de onda de cada cubo hiperesperctral) en documentos de texto, por lo que procedemos a importar todos los documentos mediante un bucle.

```
list = []
resultados = pd.read_csv("Resultados.csv")
data1 = pd.read_csv("Dataset/C011180098.txt", sep="\t", header=None)
data2 = pd.read_csv("Dataset/C011180099.txt", sep="\t", header=None)
list.append(data1[1])
list.append(data2[1])
for x in range(100, 454):
    posicion = str(x)
    ruma = "C011180"+posicion
       ruta = "Dataset/"+ruma+".txt"
       if os.path.isfile(ruta) == True:
           data = pd.read_csv(ruta, sep="\t", header=None)
           list.append(data[1])
```

Convertir la data en array

Las redes neuronales no son más que operaciones matriciales y vectoriales, por lo que para poder diseñar y trabajar con los datos proporcionados necesitamos convertir toda la data con la que vamos a trabajar en matrices y vectores.

```
array_fishmeal = np.asarray(list)
array_fishmeal
```

De esta manera, se obtiene los datos con estructura de arreglo:

```
array([[0.116572 , 0.10317  , 0.0943778, ..., 0.577043 , 0.582872 ,
        0.588531 ],
       [0.106541 , 0.0961461, 0.0871986, ..., 0.563301 , 0.568735 ,
        0.575253 ],
       [0.112578 , 0.100005 , 0.0914678, ..., 0.573188 , 0.578893 ,
        0.584569 ],
       ...,
       [0.10706  , 0.0945424, 0.0852444, ..., 0.590925 , 0.596473 ,
        0.602781 ],
       [0.105737 , 0.094146 , 0.0850428, ..., 0.573756 , 0.579098 ,
        0.585327 ],
       [0.118916 , 0.106301 , 0.0965878, ..., 0.608553 , 0.61411  ,
        0.620498 ]])
```

Figura 24. Resultado del *dataset* convertido en arreglo.

Fuente: Elaboración propia.

Separar los arrays según el parámetro de salida

Nuestra red neuronal multicapa posee una neurona de salida, por lo tanto, va a proporcionar un modelo por cada parámetro. Por ese motivo, se deben separar los cuatro parámetros en distintos arrays para las operaciones matriciales.

```
resultados_proteina = resultados["PROTEINA (DUMAS)"]
resultados_grasa = resultados["GRASA"]
resultados_humedad = resultados["HUMEDAD"]
resultados_ceniza = resultados["CENIZAS"]
array_proteina = np.asarray(resultados_proteina)
array_grasa = np.asarray(resultados_grasa)
array_humedad = np.asarray(resultados_humedad)
array_ceniza = np.asarray(resultados_ceniza)
```

Crear una semilla

Como las redes neuronales artificiales inicializan sus pesos y ballas de forma aleatoria, se debe crear una semilla para la reproductibilidad de los experimentos. Lo que hace esta línea de código es generar los mismos valores aleatorios para cada valor de semilla.

```
seed = 584945
tf.set_random_seed(seed)
np.random.seed(seed)
```

Valores de entrada y salida

Definimos cuales van a ser los arrays de salida y los de entrada, en este caso, el array de salida será el que contenga los valores de proteína. Los valores de entrada son las longitudes de onda obtenida de cada cubo hiperespectral. La entrada será la misma para todos los modelos.

```
x_vals = array_fishmeal
y_vals = array_proteina
```

Separar datos de entrenamiento y test

Para poder comprobar que el modelo realmente es el adecuado, debemos separa nuestros datos en dos grupos. El primer grupo será usado para la creación del modelo y el segundo será utilizado para comprar la eficacia del modelo.

Se separaron los grupos de la siguiente forma: 80% para entrenar el modelo, 20% para validar el modelo. No existe un criterio específico para la separación entrenamiento/validación.

```
def convert(set):
    return sorted(set)
train_idx = np.random.choice(len(x_vals), round(len(x_vals)*0.8),
replace=False)
test_idx = np.asarray(convert(set(range(len(x_vals)))-
set(train_idx)))
x_vals_train = x_vals[train_idx]
x_vals_test = x_vals[test_idx]
y_vals_train = y_vals[train_idx]
y_vals_test = y_vals[test_idx]
```

Entrenamiento por lotes o "batch size"

El tamaño del lote define el número de muestras que se propagarán a través de la red. Por ejemplo, supongamos que tiene 1000 muestras de entrenamiento y desea configurar un tamaño de lote igual a 50. El algoritmo toma las primeras 50 muestras del conjunto de datos de entrenamiento y entrena la red. Luego, toma las segundas 50 muestras (de la 51 a la 100) y entrena la red nuevamente. Las ventajas de hacer esto es que se requiere menos memoria y que por lo general, las redes entrenan más rápido en lotes.

```
1. batch_size = 50
2. x_data = tf.placeholder(shape = [None, 240], dtype = tf.float32)
3. y_target = tf.placeholder(shape=[None, 1], dtype = tf.float32)
```

Creación de las capas ocultas

Para la creación de capaz ocultas utilizaremos la librería de tensorflow. Primero debemos definir cuantas neuronas habrá en cada capa oculta, luego se procederá a hacer operaciones matriciales con los valores de ballas y pesos. Como ya se vio anteriormente, los parámetros de ballas y los pesos de toda la red inicializarán sus valores de forma aleatoria.

Es importante definir correctamente las dimensiones de las matrices. Los errores más comunes al programar una red neuronal se deben a la dimensionalidad de las matrices. Las dimensiones de las matrices deben cumplir con las reglas de operación del algebra matricial.

La red que se observa en el código es que contiene 5 capas ocultas, posteriormente probamos una menor cantidad de capaz ocultas y obtuvimos mejores resultados. Un mayor número de capas no siempre es mejor, una red neuronal muy compleja podría originar un sobreajuste u *overfitting*.

```
1. hidden_layer_nodes= [5, 5, 5, 1, 1]
2. A1 = tf.Variable(tf.random_normal(shape=[240, hidden_layer_nodes[0
   ]]))
3. b1 = tf.Variable(tf.random_normal(shape=[hidden_layer_nodes[0]]))
4. A2 = tf.Variable(tf.random_normal(shape=[hidden_layer_nodes[0],
   hidden_layer_nodes[1]]))
5. b2 = tf.Variable(tf.random_normal(shape=[hidden_layer_nodes[1]]))
6. A3 = tf.Variable(tf.random_normal(shape=[hidden_layer_nodes[1],
   hidden_layer_nodes[2]]))
7. b3 = tf.Variable(tf.random_normal(shape=[hidden_layer_nodes[2]]))
8. A4 = tf.Variable(tf.random_normal(shape=[hidden_layer_nodes[2],
   hidden_layer_nodes[3]]))
9. b4 = tf.Variable(tf.random_normal(shape=[hidden_layer_nodes[3]]))
10.   A5 = tf.Variable(tf.random_normal(shape=[hidden_layer_nodes[3],

   hidden_layer_nodes[4]]))
11.   b5 = tf.Variable(tf.random_normal(shape=[hidden_layer_nodes[4]]
   ))
12.   A6 = tf.Variable(tf.random_normal(shape=[hidden_layer_nodes[4],
    1]))
13.   b6 = tf.Variable(tf.random_normal(shape=[1]))
```

Operación de las capaz ocultas

Una capa se puede expresar como la operación para cada neurona de la capa, de la suma del parámetro de ballas con el producto de los pesos que le corresponde a cada neurona de la capa anterior. La salida de cada neurona pasa por una función de activación. En este informe se utilizó la función de activación *Rectified Linear Unit* (ReLu).

```
1. hidden_output1 = tf.nn.relu(tf.add(tf.matmul(x_data, A1), b1))
2. hidden_output2 = tf.nn.relu(tf.add(tf.matmul(hidden_output1, A2),
   b2))
3. hidden_output3 = tf.nn.relu(tf.add(tf.matmul(hidden_output2, A3),
   b3))
4. hidden_output4 = tf.nn.relu(tf.add(tf.matmul(hidden_output3, A4),
   b4))
5. hidden_output5 = tf.nn.relu(tf.add(tf.matmul(hidden_output4, A5),
   b5))
6. final_output = tf.nn.relu(tf.add(tf.matmul(hidden_output5, A6), b6
   ))
```

Función de error

La función de error nos dirá que tanto difieren los valores de salida obtenidos con el modelo actual en la etapa de entramiento con el valor real. La función que utilizaremos para medir el error del modelo es la del error cuadrático medio (RMSE).

```
1. loss = tf.reduce_mean(tf.square(y_target-final_output))
```

Algoritmo de backpropagation

Este algoritmo usa las derivadas parciales de la función de error para “propagar hacia atrás” el error de salida del modelo en la etapa de entrenamiento, modificando los pesos y ballas, y de esa forma obtener un modelo más preciso. La explicación de la matemática está explicada en la sección de red neuronal multicapa del informe.

```
1. my_optim = tf.train.GradientDescentOptimizer(learning_rate=0.01)
2. train_step = my_optim.minimize(loss)
```

Inicializar variables

Tensorflow solo define las variables, para poder operar con ellas e imprimir los resultados en pantalla, necesitamos inicializar todas las variables.

```
1. init = tf.global_variables_initializer()
2. session.run(init)
```

Entrenar el modelo y validarlo

Luego de definir toda la red neuronal multicapa procedemos a entrenar el modelo y a validarlo. Crearemos arrays que contengan el error de los datos de entrenamiento y otro con el error de los datos de validación para el modelo generado. Luego se mostrarán de forma gráfica con la librería matplotlib.

Mediante un bucle definimos la cantidad de veces que entrenará el modelo (pasos o épocas).

Dentro del bucle se creará el modelo con los datos de entrenamiento. Con los datos de validación se validará ese modelo generado. En cada paso el modelo cambiará e idealmente, el error debería disminuir.

```
1. loss_vect = []
2. test_loss = []
3. for i in range(10000):
4.     rand_idx = np.random.choice(len(x_vals_train), size=batch_size
   )
5.     rand_x = x_vals_train[rand_idx]
6.     rand_y = np.transpose([y_vals_train[rand_idx]])
7.     session.run(train_step, feed_dict={x_data:rand_x, y_target:ran
   d_y})
8.
9.     temp_loss = session.run(loss, feed_dict={x_data:rand_x,
       y_target:rand_y})
10.       loss_vect.append(np.sqrt(temp_loss))
11.
12.       temp_loss_test = session.run(loss, feed_dict={x_data:
       x_vals_test, y_target: np.transpose([y_vals_test])})
13.       test_loss.append(np.sqrt(temp_loss_test))
14.
15.       if(i+1)%50==0:
16.           print("Paso #"+str(i+1)+", Loss = "+str(temp_loss))
```

En la iteración (paso) número 50 vemos un error RMSE de 630 y conforme el modelo se sigue entrenando, el error disminuye cada vez más hasta oscilar en un modelo con décimas de error medio cuadrático.

```
Paso #50, Loss = 630.73126
Paso #100, Loss = 82.1665
Paso #150, Loss = 11.438301
Paso #200, Loss = 2.2167559
Paso #250, Loss = 1.0107378
Paso #300, Loss = 1.0315943
Paso #350, Loss = 0.837083
Paso #400, Loss = 0.7974546
Paso #450, Loss = 0.7242638
Paso #500, Loss = 0.64702386
Paso #550, Loss = 0.8629131
Paso #600, Loss = 0.8364887
Paso #650, Loss = 0.99050784
Paso #700, Loss = 0.6917727
Paso #750, Loss = 0.79234093
Paso #800, Loss = 0.77267
Paso #850, Loss = 0.73333275
Paso #900, Loss = 1.2549709
Paso #950, Loss = 0.89822066
Paso #1000, Loss = 0.7567629
```

Figura 25. "Pérdidas" obtenidas en el entrenamiento.

Fuente: Elaboración propia.

Graficar el error resultante del modelo

Con la librería matplotlib podemos graficar los resultados obtenidos.

```
1. plt.plot(loss_vect, "r-", label="Pérdida Entrenamiento")
2. plt.plot(test_loss, "b--", label ="Pérdida Test")
3. plt.title("Pérdida (RMSE) por iteración en la Proteina")
4. plt.xlabel("Iteración")
```

```
5. plt.ylabel("RMSE")
6. plt.legend(loc ="upper right")
7. plt.show()
```

En la gráfica se observa como el error cuadrático medio disminuye hasta un valor de error aceptable. La línea roja representa el valor RMSE de los datos de entrenamiento y la línea azul el valor RMSE de los datos de validación. Vemos como el modelo se ajusta a ambos datos, por lo que el modelo obtenido sí es un modelo aceptable, sin *overfitting* ni *underfitting*.

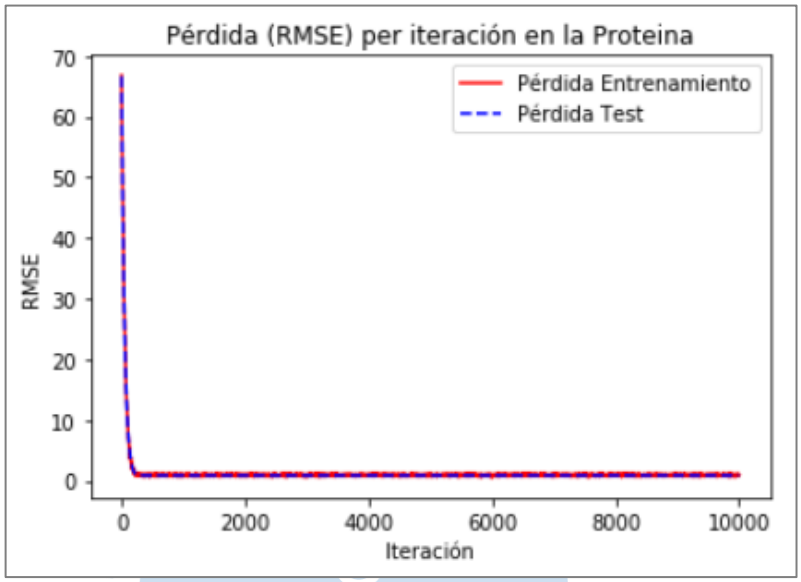

Figura 26. Grafica del error al entrenar la Red Neuronal.

Fuente: Elaboración propia.

Generar el modelo obtenido

Podemos imprimir los valores de los pesos y los parámetros de ballas del modelo generado.

Estos parámetros definen todo el modelo generado por la red para la proteína.

- Pesos de la primera capa oculta con 5 neuronas:

```
1. print(session.run(A1))
```

```
[[-0.20842202  0.52706003  0.55224234  0.933294    0.51383436]
 [-0.82788473  0.5757222   0.9018055   0.4501978  -1.6402131 ]
 [ 0.23727751  0.33049813  1.0097969   0.9000987   0.5398288 ]
 ...
 [-0.63832486 -0.77257806  1.1721529  -1.4532584   0.6787666 ]
 [ 0.24410436  0.7914458  -1.9392885  -0.45447442 -1.6386455 ]
 [ 0.19522072  1.5239136  -0.26334605  0.14014281  0.48620608]]
```

- Ballas de la primera capa oculta con 5 neuronas:

```
1. print(session.run(b1))
```

```
[-0.1897393   0.64894664 -0.02969149 -0.73929226 -0.35140768]
```

- Pesos de la segunda capa oculta con 5 neuronas:

```
2. print(session.run(A2))
```

```
[[ 1.5252358e+00 -2.5134505e-04 -2.7262291e-01 -5.5287492e-01
  -3.8004327e-01]
 [ 8.6270040e-01  7.1139693e-01  8.0502194e-01  1.2437102e+00
  -4.3106336e-02]
 [-6.2444669e-01  1.3511616e-01 -6.7100823e-01 -8.9397407e-01
   4.9259070e-01]
 [-4.8860374e-01  2.6001815e-02  1.7315366e+00  5.8873999e-01
   7.8829902e-01]
 [-7.2406399e-01 -6.9422239e-01  3.7111187e-01 -1.5258522e-01
  -2.9632634e-01]]
```

- Ballas de la segunda capa oculta con 5 neuronas:

```
2. print(session.run(b2))
```

```
[-0.66172343  1.9106797  -0.421649   -0.08250617  0.35461462]
```

- Pesos de la tercera capa oculta con 5 neuronas:

```
3. print(session.run(A3))
```

```
[[-0.07982305 -1.4046396   0.40846676  0.01308282 -0.54549295]
 [-1.116348   -1.9150941   0.1467966  -0.12951899 -0.34035832]
 [-1.7240883  -1.0580809   1.2602246   1.1505102   1.1786187 ]
 [ 1.025627   -2.1313064  -1.284858   -1.0366708  -0.5095603 ]
 [ 2.7401767   0.31945354 -0.32146296 -1.026127    1.5260487 ]]
```

- Ballas de la tercera capa oculta con 5 neuronas:

```
3. print(session.run(b2))
```

```
[-0.28617075 -0.80095273 -1.4090811  -1.917504   -1.711487  ]
```

- Pesos de la cuarta capa oculta con 1 neurona:

```
4. print(session.run(A4))
```

```
[[0.7259707 ]
 [0.52349204]
 [1.1826754 ]
 [2.1797047 ]
 [1.5613068 ]]
```

- Ballas de la cuarta capa oculta con 1 neurona:

```
4. print(session.run(b4))
```

```
[0.56606734]
```

- Pesos de la quinta capa oculta con 1 neurona:

```
5. print(session.run(A5))
```

```
[[-1.1557827]]
```

- Ballas de la quinta capa oculta con 1 neurona:

```
5. print(session.run(b5))
```

```
[-0.10458248]
```

3.3. Modelos para estimar parámetros de calidad de la harina de pescado

En base a la teoría descrita anteriormente, se ha decidido trabajar con una red neuronal del tipo perceptrón multicapa para la predicción de los parámetros de calidad de la harina de pescado, específicamente: proteína, humedad, cenizas y grasa. En todos los modelos la entrada a la red son las firmas espectrales obtenidas de 175 cubos de data, a través del software Spectronon Pro. Para primeros análisis se hará uso de las 240 bandas espectrales medidas correspondientes al rango del espectro electromagnético captado por la cámara hiperespectral modelo PIKA II.

La salida de cada red neuronal multicapa corresponde a una neurona que tomará los 175 valores de la data correspondiente a la empresa COPEINCA según sus metodologías tradicionales para la medición de parámetros de calidad de la harina, una vez terminado el proceso de producción.

Como se mencionó previamente se trabajará con 175 firmas espectrales, obtenidas tras el filtrado de data, con sus respectivos valores correspondientes a cada parámetro medido según la metodología tradicional de la industria. Se utilizará el 80% de la data para el entrenamiento de la red y el 20% para su respectiva validación.

A continuación, se mostrarán los resultados de los modelos de red neuronal correspondientes a cada parámetro en estudio.

3.3.1. *Humedad*

La humedad presenta un papel importa al momento de definir la calidad de la harina de pescado ya que de ella depende la optimización de los otros parámetros en estudio. Por ello, es conveniente detectar la calidad de la harina de pescado a la salida del secador

mediante la predicción de sus parámetros y poder optimizar su calidad según la corrección del porcentaje de humedad de la harina a la salida tras un reingreso en el secador (únicamente rotatubos en este caso). La pronta corrección se realiza a fin de obtener el mejor estándar de calidad para la comercialización del producto. Para ello, se han diseñado modelos según la descripción previa.

3.3.1.1. Modelo 1. El primer modelo para la predicción de humedad se realizó con 3 capas ocultas con 6, 4 y 2 neuronas en cada una, respectivamente. Además, contó con un ratio de aprendizaje igual a 0.01 y 20 000 iteraciones. Se utilizó un *batch size* igual a 500.

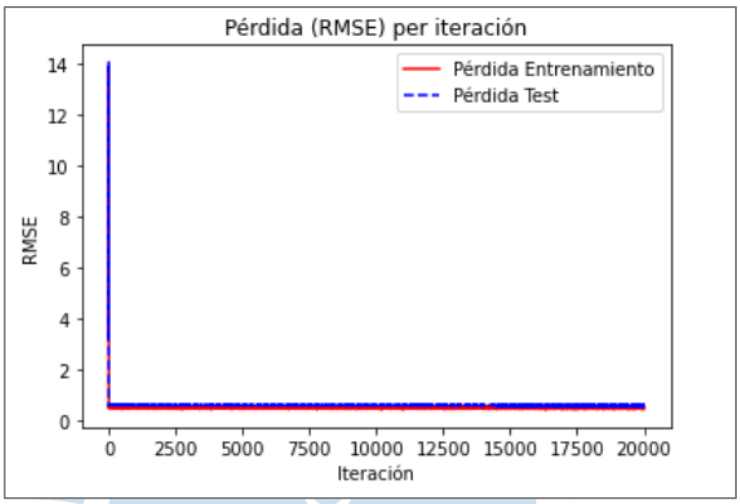

Figura 27. RMSE de la data de entrenamiento (rojo) y validación (azul) del Modelo 1, Humedad.

Fuente: Elaboración propia.

Se observa que la raíz cuadrada del error cuadrático medio (RMSE) que presenta la red es muy bajo, alrededor de 0.22 en las últimas iteraciones con un valor mínimo de 0.19.

3.3.1.2. Modelo 2. El segundo modelo se realizará con 1 capa oculta y 15 neuronas en la misma. Además, contará con un ratio de aprendizaje de 0.01 y 20 000 iteraciones. Además, se utilizó un *batch size* igual a 100 con la finalidad de no perder precisión en la respuesta tras el entrenamiento.

De los resultados obtenidos, se tiene que el punto mínimo del RMSE es 0.12590621 y se mantiene en valores próximos a este en las últimas iteraciones.

Con este modelo se obtienen RMSE menores respecto al modelo anterior. En este caso se utilizó los hiperparámetros definidos previamente respecto al número de capas ocultas, en este caso igual a uno, y el número de neuronas que le corresponden, obtenidas de la siguiente expresión:

$$h = (i * o)^{0.5}$$

Donde, referenciando a la teoría explicada previamente, cada variable corresponde a:

h: número de neuronas de la capa oculta

i: número de neuronas de la capa de entrada

o: número de neuronas de la capa de salida.

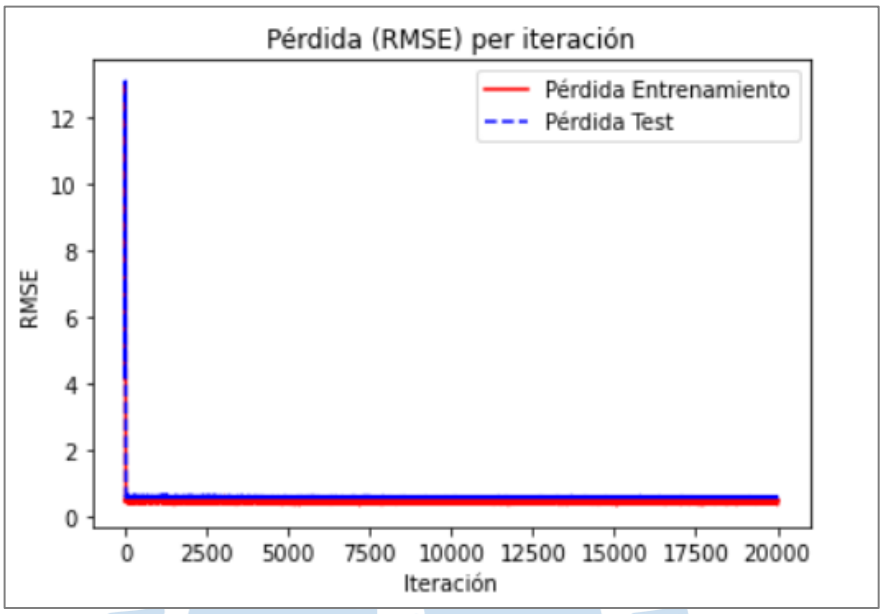

Figura 28. RMSE de la data de entrenamiento (rojo) y validación (azul) del Modelo 2, Humedad.

Fuente: Elaboración propia.

De esta manera, se procede a reemplazar dichos valores definidos inicialmente.

$$h = (240 * 1)^{0.5}$$

$$h = 15.4919$$

Para que sea posible establecer el resultado al número de neuronas, se aproxima al valor entero menor.

$$h \approx 15$$

Si se observa el RMSE se puede afirmar que los hiperparámetros calculados brindan un mejor resultado y, por ende, mejor predicción que el primer modelo.

3.3.1.3. Comparaciones. Ambos modelos presentan un error cuadrático medio bajo indicando un buen funcionamiento de la red neuronal con los hiperparámetros establecidos. Además, ninguno de los dos modelos indica *overfitting* porque el RMSE de la data de validación (elegida aleatoriamente por la red neuronal), con un margen de error aceptable,

disminuye conforme a la data de entrenamiento, es decir, no llega un punto donde los RMSE de la data de validación aumentan y se separan de los RMSE de la data de entrenamiento.

En caso uno de los modelos presentara *overfitting* estaría memorizando los datos de entrenamiento y ajustándose mucho a ellos crearía un modelo que prediga muy bien dichos datos, pero con la data de validación ocurriría lo contrario, es decir, el RMSE empezaría a aumentar a medida que el modelo se ajusta más a los datos de entrenamiento, generando una mala predicción para nuevos datos.

- Conclusiones

En caso el número de capas ocultas hubiese sido mayor, se daría el caso donde la predicción de la red es menos precisa, incluso puede presentar *overfitting* debido a un sobreajuste del modelo para predecir los parámetros de entrenamiento. Si, por ejemplo, se aplican 15 capas ocultas a la red neuronal, el RMSE no converge y se queda oscilando en valores elevados.

- Sugerencias

Según las conclusiones previas sería conveniente aplicar el segundo modelo para la predicción del parámetro de calidad correspondiente a la humedad. Además, se puede tener confianza en la predicción para nuevos datos debido a que, en la industria de la harina de pescado, se trabaja con medidas de calidad preestablecidas que corresponden a un determinado tipo de calidad. De esta manera, la industria tendrá indicadores similares entre sus lotes buscando que siempre sean los óptimos, por lo que, los datos nuevos para la red, si bien no serán los mismos, serán valores no muy distantes de los aplicados para entrenar y validar la red.

3.3.2. *Proteína*

3.3.2.1. **Modelo 1.** El primer modelo para la predicción de la proteína se realizará con dos capas ocultas, la primera conformada por 2 neuronas y la segunda por 1. Además, se aplicará un ratio de aprendizaje igual a 0.01 y 15 000 iteraciones. Se utilizó un *batch size* igual a 50.

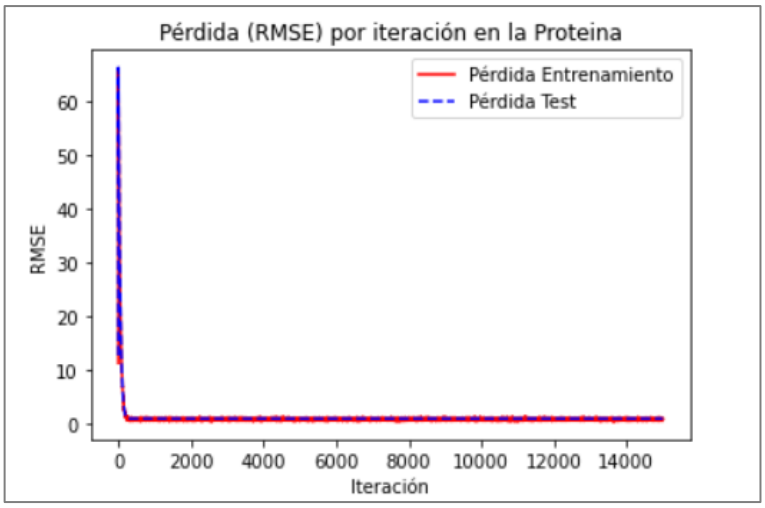

Figura 29. RMSE de la data de entrenamiento (rojo) y validación (azul) del Modelo 1, Proteína.

Fuente: Elaboración propia.

RMSE mínimo 0.40148172 y en las últimas iteraciones es aproximadamente 0.8.

3.3.2.2. Modelo 2. Este modelo es trabajado con una capa oculta y una sola neurona en la misma. Además, presenta un *batch size* de 50 y realiza 15000 iteraciones. El *learning rate* (ratio de aprendizaje) es igual a 0.01.

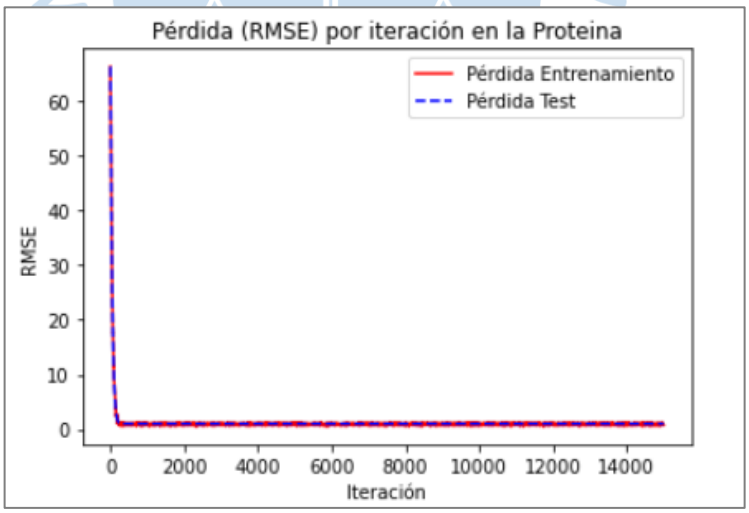

Figura 30. RMSE de la data de entrenamiento (rojo) y validación (azul) del Modelo 2, Proteína.

Fuente: Elaboración propia.

El RMSE mínimo es igual a 0.38303834, en las últimas iteraciones presenta un RMSE de aproximadamente 0.5.

3.3.2.3. Comparaciones. En el segundo caso se realizaron igual número de iteraciones, mismo *batch size* y se trabajó con un *learning rate* igual a 0.1, por lo tanto, el cambio entre ambos modelos estuvo en el número de capas ocultas y el número de neuronas en las mismas. En el segundo modelo se obtuvo mejores resultados porque se presenta un MRSE menor, si bien tienen un RMSE mínimo similar, el RMSE que se obtuvo en las últimas iteraciones fue menor en el segundo caso. Esto puede ser consecuencia de la relación entre los datos a la salida de la red, basta con una capa oculta y una neurona para que el sistema converja de la mejor manera, ya que si se aumente el número de capas se podría presentar *overfitting* haciendo que el RMSE de los valores de validación no vaya conforme al de la data de entrenamiento y diverja.

Si se agregan más neuronas a una capa oculta, el sistema podría presentar la misma situación a medida que se aumente el número de neuronas en dicha capa, en este caso se probó con 15 neuronas y el sistema no convergió, sin embargo, para 3 neuronas en dicha capa el sistema convergió, pero con un RMSE mayor a cuando se trabajó con una sola neurona.

Si se trabaja con 2 capas ocultas, como en el segundo modelo, depende del número de neuronas en dichas capas para que el error del modelo converja o no. Si el número de neuronas es bajo hay más probabilidad de obtener un buen modelo que presente un bajo error, sin embargo, si se aumentan las neuronas en las capas ocultas ocurrirá lo contrario.

- Conclusiones

  En base a las conclusiones realizadas sería conveniente utilizar el segundo modelo debido al bajo RMSE que presenta que reflejado únicamente como error cuadrático medio sería un valor menor. De esta manera la predicción de la proteína sería más preciso. Además, por presentar un bajo número de capas y al ser evaluado en 15000 iteraciones no se presentaron indicadores de *overfitting*, por lo que, sería un modelo confiable para ser utilizado en futuras predicciones de este parámetro de calidad.

- Sugerencias

  En base a las conclusiones realizadas sería conveniente utilizar el segundo modelo debido al bajo RMSE que presenta que reflejado únicamente como error cuadrático medio sería un valor menor. De esta manera la predicción de la proteína sería más preciso. Además, por presentar un bajo número de capas y al ser evaluado en 15000 iteraciones no se presentaron indicadores de *overfitting*, por lo que, sería un modelo confiable para ser utilizado en futuras predicciones de este parámetro de calidad.

3.3.3. *Grasa*

Es uno de los parámetros más importantes al definir la calidad de la harina de pescado debido a la gran energía que proporciona hacia los consumidores generalmente animales. Es

indispensable su medición debido a que las calidades internacionales piden como máximo el 10% de grasa contenida en la harina.

3.3.3.1. Modelo 1. El primer modelo para la predicción de la Grasa se realizará con 1 capa oculta y en ella una neurona. Además, se aplicará un ratio de aprendizaje igual a 0.01 y 40000 iteraciones. Finalmente se utilizó un *batch size* igual a 30.

Se observa que el error cuadrático medio que presenta la red es bajo, alrededor de 0.29 en las últimas iteraciones con un valor mínimo de 0.18.

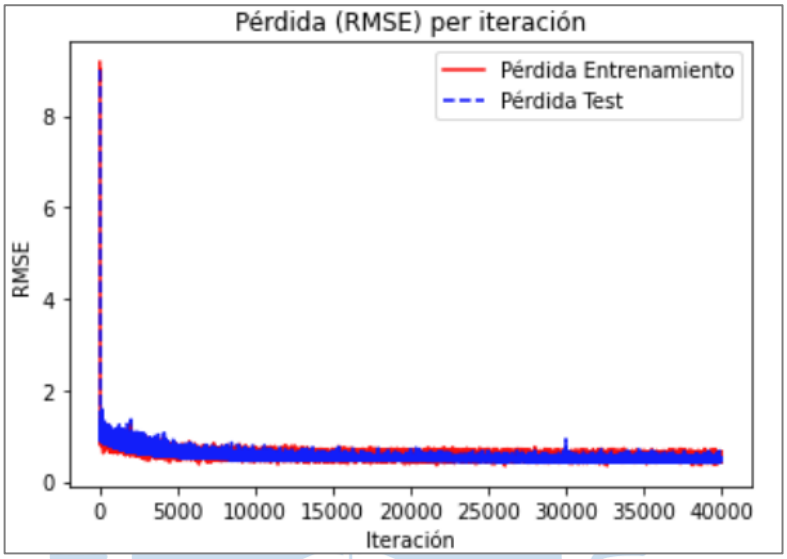

Figura 31. RMSE de la data de entrenamiento (rojo) y validación (azul) del Modelo 1, Grasa.

Fuente: Elaboración propia.

3.3.3.2. Modelo 2. El segundo modelo para la predicción de la Grasa se realizará con 1 capa oculta y en ella 15 neuronas. Además, se aplicará un ratio de aprendizaje igual a 0.01 y 40000 iteraciones. Finalmente se utilizó un *batch size* igual a 480.

En la Figura 32 se observa que el error cuadrático medio que presenta la red es bajo, alrededor de 0.29 en las últimas iteraciones con un valor mínimo de 0.24.

3.3.3.3. Comparaciones. Ambos modelos presentan un error cuadrático medio bajo indicando un buen funcionamiento de la red neuronal con los hiperparámetros establecidos.

Para estos modelos no se puede presentar *overfitting* debido a que se ataca escogiendo la data de forma aleatoria, se utilizaron modelos de una capa oculta que disminuye el riesgo de *overfitting*. Además, la data de entrenamiento y validación convergen.

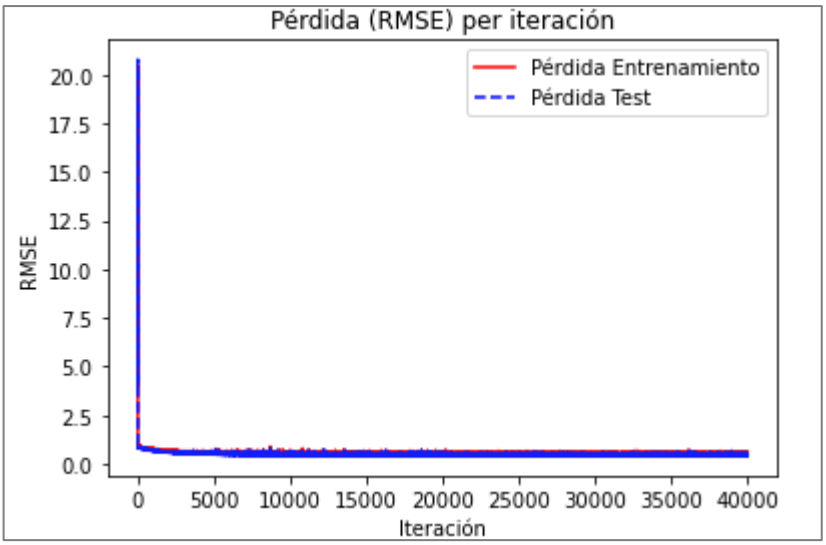

Figura 32. RMSE de la data de entrenamiento (rojo) y validación (azul) del Modelo 2, Grasa.

Fuente: Elaboración propia.

Se observa que en el primer modelo hay una baja precisión del gradiente, esto es debido a que el *batch size* es más bajo que la Data, por otro lado, en el modelo 2 la estimación del gradiente es más precisa, aunque el modelo presenta un error mínimo mayor.

- Conclusiones

Podemos concluir que el modelo 1 es mejor debido a que es más rápido y tiene menor riesgo de *overfitting* a presenta menor error a comparación del modelo 2, que tiene mayor error, pero corre el riesgo de que su aprendizaje sea muy poco confiable.

También se observa una mejora al implementar las 15 neuronas en una capa como se demostró en la Humedad.

- Sugerencias

Para tener mejores estimaciones es recomendable que el *batch size* se tome en múltiplos de la Data, por ejemplo, ambos modelos fueron múltiplos de 240, uno mayor y el otro menor, Esto se debe a que en el momento de entrenamiento es más probable que la Data se entrene de forma más equitativa.

3.3.4. *Ceniza*

Este parámetro generalmente se somete a altas temperaturas entre 2 a 3 veces y se calcula con relaciones matemáticas, con el modelo a desarrollar esto se simplifica a la toma de imágenes hiperespectrales y aplicación del modelo de predicción.

3.3.4.1. Modelo 1. El primer modelo para la predicción de la Grasa se realizará con 3 capas oculta y en ella una neurona. Además, se aplicará un ratio de aprendizaje igual a 0.01 y 40000 iteraciones. Finalmente se utilizó un *batch size* igual a 30.

Se observa que el error cuadrático medio que presenta la red es bajo, alrededor de 0.14 en las últimas iteraciones con un valor mínimo de 0.11.

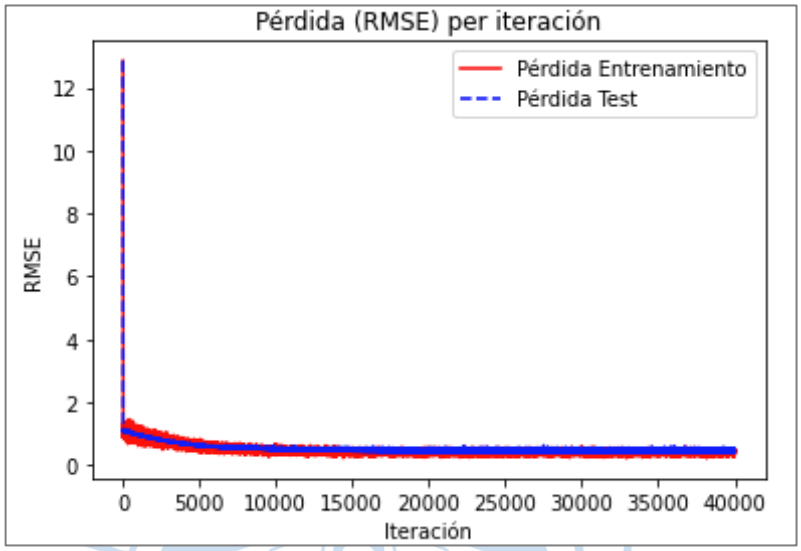

Figura 33. RMSE de la data de entrenamiento (rojo) y validación (azul) del Modelo 1, Ceniza.

Fuente: Elaboración propia.

3.3.4.2. **Modelo 2.** En este segundo modelo se implementó el comando de *dropout* del 90%, la predicción de la Grasa se realizará con 1 capas oculta y en ella 15 neuronas. Además, se aplicará un ratio de aprendizaje igual a 0.01 y 30000 iteraciones. Finalmente se utilizó un *batch size* igual a 480.

Se observa que el error cuadrático medio que presenta la red es bajo, alrededor de 0.17 en las últimas iteraciones con un valor mínimo de 0.16.

```
hidden_layer_nodes= [15]
A1 = tf.Variable(tf.random_normal(shape=[240, hidden_layer_nodes[0]]))
b1 = tf.Variable(tf.random_normal(shape=[hidden_layer_nodes[0]]))
b1_drop=tf.nn.dropout(b1, keep_prob)
```

Figura 34. Capa oculta de la red neuronal con implementación de *dropout*.

Fuente: Elaboración propia.

3.3.4.1. **Comparaciones.** Ambos modelos poseen una capa, la diferencia está en el número de neuronas, *batch size* e iteraciones. Los dos modelos pueden predecir muy bien por tener bajo

error, además, como se comentó anteriormente la probabilidad de que aparezca *overfitting* en los modelos de una capa, data aleatoria son bajos.

El modelo 2 presenta una mejor y más rápida convergencia debido al *batch size* mayor, por otro lado, el primer modelo tiene un error muy bajo. Además, en el modelo 2 está implementado el *Dropout* que sirve para apagar neuronas aleatorias y evitar que sus coeficientes se modifiquen, con esto se mejor la capacidad de generalización del modelo.

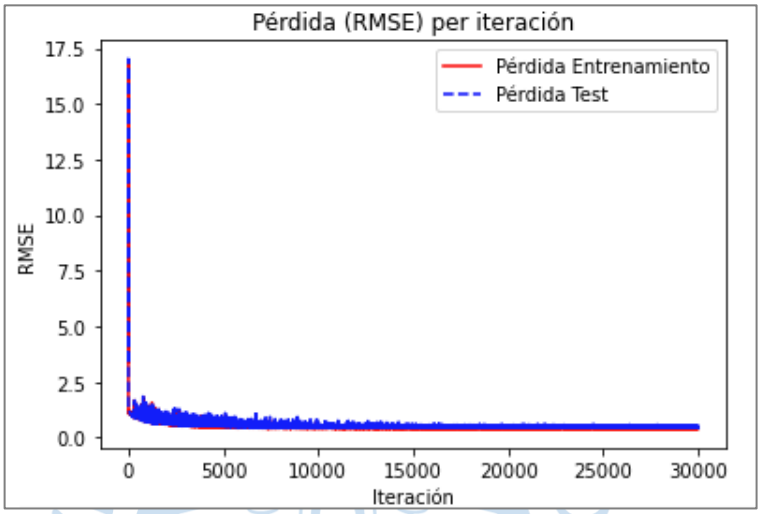

Figura 35. RMSE de la data de entrenamiento (rojo) y validación (azul) con *dropout* en Modelo 2, Ceniza.

Fuente: Elaboración propia.

- Conclusiones

Ambos modelos son buenos, pero el modelo 2 presenta el *Dropout* que garantiza que el riesgo de *overfitting* disminuya mucho más.

- Sugerencias

Es recomendable atacar el *overfitting* y *underfitting* debido a que estos son los culpables de que el modelo no pueda reconocer nueva data de entrada. Esta tarea suele ser una de las más importantes en la creación y aplicación de un modelo.

Capítulo 4
Análisis de resultados y discusión

Se presenta el análisis de los algoritmos finales, presentan una mayor predicción y menor que los algoritmos experimentales y preliminares presentados en el capítulo 3.

4.1. Humedad

La humedad es un parámetro importante usado para la determinación del nivel de calidad de la harina de pescado ya que de ella depende en gran manera la optimización de los otros parámetros usados para determinar la calidad. Debido a esto, la detección de la humedad en la harina de pescado a la salida del secador haciendo uso de las imágenes hiperespectrales sumadas al algoritmo de ANN desarrollado, permite predecir de manera muy aceptable el porcentaje de humedad presente en las muestras usadas para el desarrollo del presente trabajo de investigación, determinando así que el modelo de red neuronal que se adapta satisfactoriamente a los resultados esperados, presenta una arquitectura con 5 capas ocultas presentando un orden piramidal de 10,8,6,4,2 neuronas respectivamente, al mismo tiempo se optó por hacer uso de una tasa de aprendizaje igual a 0,0001 con 10,000 épocas. De esta manera se logró cuantificar un error cuadrático medio (RMSE) aproximado a 0.1082.

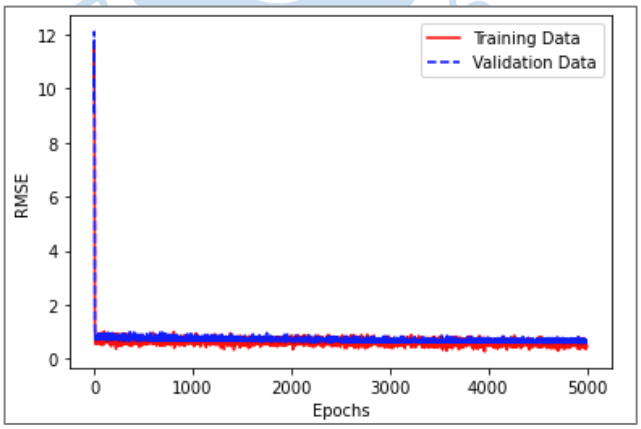

Figura 36. RMSE entrenamiento y validación para la humedad.

Fuente: Elaboración propia.

4.2. Proteína

La proteína se encuentra entre los parámetros principales para determinar el nivel de calidad de la harina de pescado, este parámetro puede variar según la frescura del pescado, pero además durante el proceso de secado puede variar según la temperatura y tiempo de exposición, es por esto que lograr cuantificar el porcentaje de proteína durante el proceso de producción, puede evitar el reprocesamiento del producto obtenido, es así que el algoritmo desarrollado y adaptado específicamente para la predicción de este parámetro, presenta una arquitectura con 5 capas ocultas y 5 neuronas en cada una respectivamente, además con un tamaño de lote igual a 20 y un error cuadrático medio (RMSE) aproximado de 0.3.

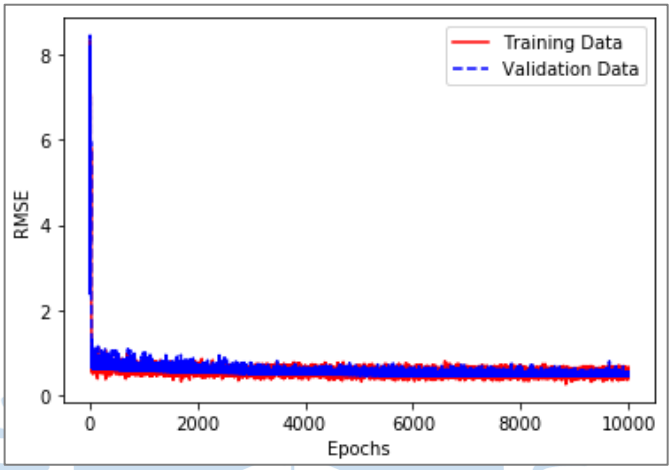

Figura 37. RMSE entrenamiento y validación para la proteína.

Fuente: Elaboración propia.

4.3. Grasa

La grasa en la harina de pescado es un factor de calidad que frecuentemente se busca controlar durante todo el proceso de producción ya que este parámetro permitirá controlar el nivel hormonal de los animales y al mismo tiempo gracias a su contenido energético permitirá obtener mayor producción en la industria alimentaria, los estándares de calidad indican que el porcentaje de grasa debe encontrarse entre 10% y 12%, la grasa normalmente es un parámetro controlable durante el proceso de producción, sin embargo si se llega a obtener que otros parámetros tales como la humedad y la proteína necesitan un re-procesamiento, el porcentaje y calidad de grasa se verá afectado lo que puede volver al producto inservible o disminuir su precio; a partir de las muestras usadas en el presente trabajo y el algoritmo desarrollado se logró diseñar un modelo con 3 capas ocultas y 5 neuronas en cada capa respectivamente, se hizo uso de un entrenamiento de lote igual a 20 y una taza de aprendizaje de 0,0001 con 20,000 épocas; logrando así obtener un error cuadrático medio (RMSE) aproximado a 0.19.

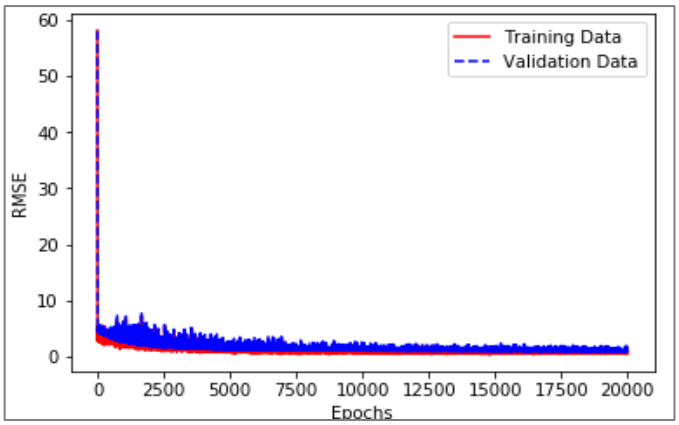

Figura 38. RMSE entrenamiento y validación para la grasa.

Fuente: Elaboración propia.

4.4. Ceniza

Este parámetro de calidad es producto del proceso de cocción y al mismo tiempo de la etapa secado, en donde producto de la temperatura y movimiento del sistema "rotatubos" se produce la calcinación de los restos óseos y parte de la materia prima obtenida en procesos anteriores, el contenido de ceniza en la harina de pescado es un parámetro no deseado durante el proceso de producción, según estándares internacionales el porcentaje de ceniza debe encontrarse entre 10% y 45%; lograr llevar el control de este parámetro implicara evitar el re-procesamiento de la materia prima, mediante la aplicación de imágenes hiperespectrales y el algoritmo desarrollado, se lograra realizar un seguimiento constante del contenido de ceniza, permitiendo regular la temperatura o velocidad del proceso; mediante ensayos se logró determinar un algoritmo con una arquitectura ideal, teniendo así 3 capas ocultas y 5 neuronas en cada capa respectivamente; además se realizó un entrenamiento de tamaño de lote igual a 20 y una tasa de aprendizaje de 0,0001 con 50,000 épocas; logrando así obtener un error cuadrático medio (RMSE) aproximado de 0,103.

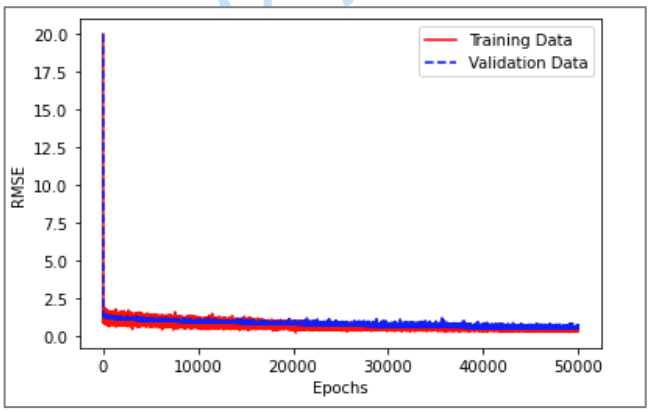

Figura 39. RMSE entrenamiento y validación para la ceniza.

Fuente: Elaboración propia.

4.5. Error porcentual entre parámetros reales y predichos

La proteína muestra una variación mínima de 0.01% y una variación máxima de 4.07% entre los valores predichos y reales. La variación promedio entre los resultados de predicción y los valores reales es 1.11%.

Para la humedad, existe una variación mínima de 0.08% entre los valores pronosticados y reales y el porcentaje más alto de la variación es de 9.8%. La variación promedio entre los resultados de predicción y los valores reales es 5.72%.

En el análisis de grasa se observa una variación mínima entre el valor predicho con respecto al valor real de 0.03% y una variación máxima de 8.44%, este último valor es considerable y puede deberse a los pocos datos utilizados, al observar el promedio de la variación de error tenemos 5.53%.

La ceniza tiene una variación promedio de 2.3% con un error mayor a 0.02% y menor a 8.2%, por lo que no varía considerablemente para su clasificación de calidad internacional, así como para la grasa. Se concluye que el método desarrollado predice de manera óptima a pesar de los datos limitados disponibles para el entrenamiento de NN.

Se alcanzó una correlación entre las redes neuronales y las imágenes hiperespectrales de harina de pescado donde es posible predecir los parámetros de calidad con imágenes hiperespectrales para optimizar el proceso de secado. La predicción de los parámetros se lleva a cabo de inmediato y durante el proceso una vez que la firma espectral se ha capturado con muy buena precisión en comparación con los valores medidos tradicionalmente con instrumentos convencionales y relaciones matemáticas después de tomar muestras en el proceso de envasado.

Tabla 4. *Error entre los valores predichos y reales (etiquetas).*

Muestra	**Proteína**			**Grasa**			**Humedad**			**Ceniza**		
	Real	***Pred.***	***Error***	***Real***	***Pred.***	***Error***	***Real***	***Pred.***	***Error***	***Real***	***Pred.***	***Error***
CO11180102	68.810	68.713	0.14	8.300	8.707	4.90	7.360	7.461	1.37	15.640	15.807	1.07
CO11180114	68.580	68.311	0.39	8.860	8.627	2.63	7.350	7.288	0.84	15.310	15.685	2.45
CO11180173	68.290	68.332	0.06	8.710	8.528	2.09	7.180	7.023	2.19	16.210	15.946	1.63
CO11180193	67.650	68.214	0.83	8.710	8.465	2.81	7.770	7.496	3.53	15.900	15.833	0.42
CO11180233	68.260	68.702	0.65	8.680	8.707	0.31	7.010	7.291	4.01	16.130	15.821	1.92
CO11180244	68.600	68.550	0.07	8.290	8.645	4.28	7.680	7.377	3.94	15.890	15.742	0.93
CO11180270	68.730	68.909	0.26	8.980	8.805	1.95	6.920	7.190	3.90	15.460	15.719	1.67
CO11180359	68.600	68.407	0.28	8.310	8.512	2.43	7.480	7.210	3.61	16.030	15.958	0.45

Fuente: Elaboración propia.

Conclusiones

De este informe se mostró que la red 2.4.1 que presenta mayor número de capas ocultas tiene un mayor aprendizaje que la red mayor 2,2,1 que solo las divide linealmente. Pero mientras más capas no es mejor debido a que se puede producir un sobre ajuste (*overfitting*), lo que ocurrirá es que el modelo sólo se ajustará a aprender los casos particulares que le enseñamos y será incapaz de reconocer nuevos datos de entrada.

Se pueden introducir la data de la harina de pescado a la red que se proporcionará para predecir los parámetros de calidad como entradas para que así aprenda y prediga nuevos datos de entrada de nuevas muestras de harina de pescado.

Para escoger una adecuada arquitectura de red neuronal se debe evaluar los parámetros o valores de entrada con los que la red va a trabajar y qué es lo que se espera obtener de esta. Para este trabajo se usó una estructura de red neuronal perceptrón multicapa porque los valores de entrada con los que se cuenta son puntos de una curva, esta arquitectura permite introducir esos puntos como neuronas de entrada y entrenar la red para buscar un modelo adecuado donde los valores de salida serán cuatro neuronas que se acerquen lo mayor posible a los valores reales obtenidos en laboratorio.

Es importante escoger un ratio de aprendizaje o *learning rate* adecuado para entrenar una red neuronal ya que una *learining rate* demasiado alto podría hacer que la red neuronal jamás converja, es decir, jamás llegaría al valor que haga que el error sea lo más pequeño posible. En cambio, un *learning rate* demasiado bajo haría que la red neuronal se demore demasiado en encontrar los valores esperados.

Las redes neuronales se pueden usar para realizar fotointerpretación mediante el reconocimiento de patrones teniendo la capacidad de analizar pixeles de manera independiente, es decir sin influencia de los pixeles vecinos.

Las redes neuronales tienen un gran campo de aplicación, y hoy en día mediante el uso de distintas extensiones como las redes neuronales de alto orden, redes neuronales difusas; se busca superar las limitaciones que pueden presentar estas mismas; un claro ejemplo es el algoritmo desarrollado en el MIT.

El uso del perceptrón multicapa y el algoritmo de retro propagación, permitirá tener un “afinamiento” de los datos obtenidos en tiempo real, esto me permitirá trabajar con variables inconstantes en el tiempo.

La selección de nuestras funciones de activación será un paso fundamental para el éxito de la red; así podremos llegar a tener una red cada vez más simple que pueda procesar gran cantidad de datos.

La función de activación Leaky ReLU tuvo un mayor desempeño a diferencia de la ReLU simple, debido a que si esta última es modificada se obtiene la función Leaky ReLU, caracterizada por una pendiente que atenuar los valores menores a cero, mas no los convierte a cero directamente como lo haría la función ReLU sin dicha modificación.

La red neuronal tras el entrenamiento converge con un error bajo para todos los parámetros de calidad evaluados, sin embargo, se pueden variar algunos hiperparámetros, como: el learning rate, el número de capas ocultas, así como su respectivo número de nodos, para obtener mejores estimaciones, ya que con las mismas características en el código general (Apéndice 1) se observó diferentes desempeños de la red para la proteína, humedad, ceniza y grasa.

Referencias bibliográficas

Alciaturi, C; Escobar, M; De La Cruz, C; Rincón, C. (2003). Partial least squares (PLS) "regression and its application to coal analysis". *Rev. Téc. Ing. Univ. Zulia.* Vol. 26, Nº 3, 197 - 204.

A, A. (2017). "Reconocimiento de Imagenes en Frames de Video utilizando Redes Neuronales". *Ecuador: Universidad de las Fuerzas Armadas, Sangolqui*.

A, S. (2018). "Empirical Evaluation of Semi-Supervised Naïve Bayes for Active Learning".

AOAC. (1990). "Metodos Oficiales de analisis".vol.1; . *AOAC International.*

Boden, M.(1990). "The Philosophy of Artificial Intelligence. Londres: Oxford University" *Press. Traducción castellana Filosofía de la Inteligencia Artificial. Fondo de Cultura Económica. México: Fondo de Cultura Económica*.

C, C. (2019). "Medicion De Parametros De Calidad De Harina De Pescado Usando Imagenes Hiperespectrales E Inteligencia Artificial". *Piura: Pirhua.*

(s.f) "Calidad Fisico-Quimica De La Harina De Pescado Venezolana". (2013). R*evista Multidisciplinaria del Consejo de Investigacion de la Universidad de Oriente,* vol. 25.

(s.f.). Obtenido de RESONON. (s.f.). : https://resonon.com/

Corporación Pesquera Inca S.A.C. (2012). "Memoria Anual 2012".*Corporación Pesquera Inca S.A.C*

E, A. (2020). "Introduccion a Machine Learning".

ElMasry, G. &. (2010). "Principles of hyperspectral imaging technology". *En D.W. Sun (Ed.), Hyperspectral imaging for food quality analysis and control*, (pp. 3-43). *Elsevier*.

FAO. (1975). "La producción de harina y aceite de pescado". *FAO – 142 Organización De Las Naciones Unidas Para La Agricultura Y La Alimentación*.

FAO. (1994). "Control De Calidad De Insumos Y Dietas Acuícolas". *Organizacion De Las Naciones Unidas Para La Agricultura Y La Alimentacion- FAO*.

FAO. (2020). "Early closure of the Peruvian fishing season pushes prices up". *FAO(Organizacion De Las Naciones Unidas Para La Alimentacion y Agricultura).*

Faroo, H. (1996)." Industria pesquera" . *Lima: Industrial Grafica.*

Florez, O. U. (18 de junio de 2018). Obtenido de Planetachatbot: https://planetachatbot.com/un-lego-a-la-vez-explicando-la-matem%C3%A1tica-de-como-las-redes-neuronales-aprenden-9f5e21239f2d

Flores, R. F. (2008). "Las Redes Neuronales Artificales".

Frederickson, B. (2016). "An Interactive Tutorial on Numerical Optimization".

Geron, A. (2017). "Hands-On Machine Learning with Scikit-Learn and TensorFlow". *Concepts, Tools, and Techniques to Build Intelligent Systems.*

Gonzales, C. (2012). "Procesamiento a bordo de imágenes hiperespectrales de la superficie terrestre mediante hardware reconfigurable". *(Tesis Doctoral). Universidad Complutense de Madrid, Madrid, España.*

Gonzalez, J. C. (05 de Diciembre de 2017). Apsl. Obtenido de https://www.apsl.net/blog/2017/12/05/tensor-flow-para-principiantes-i/#:~:text=TensorFlow%20es%20una%20librer%C3%ADa%20de,de%20datos%20multidimensionales%20(tensores)%20.

Gutierrez, D. (2015). "Machine Learning and Data Science".

Hoskuldsson, A. (1988). "PLS regression methods". *Journal of chemometric*s, vol. 2, 211-228.

INDECOPI. (1986). "Instituto Nacional de Defensa de la Competencia y de la Protección". 1986.NTP.204.039 *Almacenamiento de Harina de Pescado . Lima, Perú.*

Ingenieros, E. (22 de septiembre de 2009). Obtenido de Slideshare: https://es.slideshare.net/mentelibre?utm_campaign=profiletracking&utm_medium=sssite&utm_source=ssslideview

Larranaga, P. I. (1997). "Tema 8. redes neuronales. Redes Neuronales". *U. del P. Vasco,* 12, 17. U. del P. Vasco, 12, 17.

Lerch, D. (13 de febrero de 2018). "Neuron4". *Obtenido de: https://medium.com/neuron4/introducci%C3%B3n-al-deep-learning-con-keras-b51c47560565*

López, C. (2019). "Aplicación de imágenes hiperespectrales para la detección temprana de Roya Amarilla (Hemileya vastatrix) en café (Coffea arábica), en el distrito de Limbamba, provincia Rodríguez de Mendoza Región Amazonas". *(Tésis de Pregrado). Universidad Nacional Toribio Rodríguez de Mendoza, Amazonas, Perú.*

Marinos, I. (2017). "Materia Prima". *IFFO*

Marsland, S. ((2015).). "Machine Learning: An Algorithmic Perspective." (*2nd edition).*

Matich, D. J. (2001). "Redes Neuronales: Conceptos Básicos y Aplicaciones".

Maza, G. V. (2018). "Aplicación de procesamiento de imágenes para clasificación de granos de cacao según su color interno". *Universidad de Piura. Piura: Repositorio Institucional Pirhua.*

Mejías, Y. C. (s.f.). "Funciones de Transferencia en el Perceptrón Multicapa: Efecto de su Combinación en Entrenamiento Local y Distribuido". *Revista cubana de Informática Médica.* 13 (2).

Microsoft. (15 de abril de 2020). "News Center Latinoamerica". *Obtenido de https://news.microsoft.com/es-xl/se-desarrolla-en-colombia-modelo-de-inteligencia-artificial-para-la-deteccion-complementaria-del-covid19/*

Microsoft. (30 de marzo de 2020). "Microsoft Build". *Obtenido de Microsoft Build: https://docs.microsoft.com/es-es/azure/machine-learning/how-to-tune-hyperparameters*

Mining, E. (2020). "Python Libraries and Advance Features".

Moon S. Kim, K. C. (2015). "United States Patente nº US 9,176,110 B1".

Moreno, M. E. (2018). "Diseño e implementación de una interfaz gráfica de usuario en lenguaje Python para el entrenamiento interactivo de redes profundas de segmentación de imagen médica". *Universidad Politecnica de Valencia.*

Mundaca, G., Soto, J., & Ipanaqué, W. (2015). "Evolution of the spectral index anthocyanin RI2 in the cocoa beans drying process". *Congreso Salesiano De Ciencia y Tecnologia,(pp 49-53).*

Municio Durán, D. (s. f.). *Técnicas de oversampling aplicadas al análisis de imágenes hiperespectrales*.

Mutlu, A. H.-A. (2011). "Prediction of wheat quality parameters using near-infrared spectroscopy and artificial". *Eur Food Res Technol, pp* 267–274.

Navarro, J. C. (2007). "Segmentación de Imágenes Hiperespectrales usando Memorias Asociativas Morfológicas". *Tonantzintla, Puebla: INAOE.*

Ojeda, L. R. (2016). "Método del gradiente de máximo descenso". *Matemática, 14(1*), *pp* 60-65.

Olabe.X, B. (2005). "Redes Neuronales Artificales Y Sus Aplicaciones". *Medicina intensiva*.

Peguero, A. (2010). "PLA espectroscopía NIR en la determinación de propiedades físicas y composición química de intermedios de producción y productos acabados". *(tesis doctoral). Universidad Autónoma de Barcelona, Barcelona, España.*

Piura, U. D. (s.f.). "Secado de harina de pescado Automatización del proceso de secado en la producción industrial de harina de pescado". *Obtenido de http://udep.edu.pe/ingenieria/proyectos/secado-de-la-harina-de-pescado/?section=resultados-esperados*

Pizardi, C. (1992)." Producción de Harinas Especiales". *Lima: Lima: Colegio de Ingenieros del Perú.*

Pu, Y. Y. (2015). "Recent progress of hyperspectral imaging on quality and safety inspection of fruits and vegetable". *Comprehensive Reviews in Food Science and Food Safety*, 14(2), *pp* 176-188.

Quintero, C. M.-G. (2018). "Uso de Redes Neuronales Convolucionales para el Reconocimiento Automático de Imágenes de Macroinvertebrados para el Biomonitoreo Participativo". *6th Engineering, Science and Technology Confe*.

Ruder, S. (2016). " An overview of gradient descent optimization algorithms".*Obtenido de: https://ruder.io/optimizing-gradient-descent/*

Ruiz, J. (2016). "Estudio de la visión hiperespectral en el proceso de fermentación del cacao (Máster en Ingeniería Mecánico-Eléctrica)". *Universidad de Piura, Piura, Perú.*

S.A.C., C. P. (2012). "Memoria Anual 2012". *COPEINCA .*

SAC, A. E. (08 de Julio de 2013). "Proceso de la Harina de Pescado". *Obtenido de SlideShare: https://es.slideshare.net/victoralayo/proceso-de-la-harina-de-pescado*

Salas, R. (2004). "Redes neuronales artificiales". U*niversidad de Valparaıso. Departamento de Computación, 1.*

Samaniego, R. C. (2009). "Análisis de la implementación del algoritmo de Backpropagation aplicado al procesamiento de imágenes satelitales sobre un entorno distribuido". *(Doctoral dissertation, Universidad Técnica Particular de Loja).*

Sánchez, S. (2018). "Recuperación de imágenes por contenido usando descriptores generados por Redes Neuronales Convolucionales". *Revista Cubana de Ciencias Informáticas.*

Smith, H. (2019). "Machine Learning".

Sociedad Nacional de Pesquería. (abril de 2018). "Harina de pescado/ Fishmeal". *Obtenido de: https://www.snp.org.pe/oferta_exportable/harina-de-pescado/*

Sociedad Nacional de Pesquería. (abril de 2018). "Harina de pescado: Perú lidera su producción mundial". *Obtenido de: https://www.snp.org.pe/harina-de-pescado/*

Sun, D. (2010). "Hyperspectral Imaging for Food Quality Analysis and Control".

Tavara, J. (2019). "Modelo de reconocimiento automático de señales de tránsito vehicular mediante aprendizaje profundo de redes neuronales convolucionales".

UNIVERSIDAD DE PIURA. (s.f.). "Secado de harina de pescado Automatización del proceso de secado en la producción industrial de harina de pescado". *Obtenido de Facultad de Ingeniería: http://udep.edu.pe/ingenieria/proyectos/secado-de-la-harina-de-pescado/?section=resultados-esperados*

Vela, V. O. (13 de abril de 2015). "Blogger". *Obtenido de: http://cienciadesatada.blogspot.com/2015/04/red-neuronal-recurrente-elman-con.html*

Viera-Maza, G. (2018). "Aplicación de procesamiento de imágenes para clasificación de granos de cacao según su color interno". *(Máster en Ingeniería Mecánico-Eléctrica). Universidad de Piura, Piura, Perú.*

Villamil, D. (2007). "Entrenamiento de una red neuronal multicapa para la tasa de cambio euro – dólar". *Ingeniera e Investigacion* .

Yvette, Q. A., & Daniel, T. C. (2019). "Beneficios para las empresas pesquerar Peruanas de conocer los niveles maximos de contaminantes permitidos en la harina de pescado para exportacion a china en los ultimos 20 años". *Universidad Privada Del Norte.*

Zambrano Alejandro, C. V. (septiembre de 2012). "Scielo". *Obtenido de http://ve.scielo.org/scielo.php?script=sci_arttext&pid=S1316-48212012000300005*

Zocca, V., Spacagna, G., Slater, D., Roelants, P. (2017). “Python Deep Learning: Next generation techniques to revolutionize computer vision, AI, speech and data analysis.”

Apéndices

UNIVERSITAS STUDIORUM PIURENSIS

Apéndice 1: Código base de la red neuronal artificial para estimación de los parámetros: humedad, proteína, grasa y ceniza.

```
import numpy as np
import scipy as cp
import matplotlib.pyplot as plt
import time
from IPython.display import clear_output
from sklearn.datasets import make_circles, make_moons, make_blobs
#CREAR EL DATASET
n=500
entradas=2
      #n número de registro de nuestros datos (500 personas)
      #'entradas' neuronas de entrada
      X, Y= make_circles(n_samples=n, factor=0.5, noise=0.05)
      Y = Y[:, np.newaxis]
      plt.scatter(X[ Y[:, 0] == 0, 0], X[ Y[:, 0] == 0, 1])
      plt.scatter(X[ Y[:, 0] == 1, 0], X[ Y[:, 0] == 1, 1])
      plt.axis("equal")
      #plt.show()
      #CLASE DE LA CAPA DE LA RED
      class capa_red():
      #__init__ inicializa los atributos del objeto que creamos
          def __init__(self, n_anterior, n_actual, funcion_activacion
):
              #función de activación
              self.funcion_activacion = funcion_activacion
              #Parametro de ballas en la red neuronal inicializados
de forma aleatoria
              self.b =np.random.rand(1, n_actual)       *2 - 1
              #Pesos entre capas inicializados de forma aleatoria
              self.w =np.random.rand(n_anterior, n_actual) *2 - 1
      #FUNCIONES DE ACTIVACIÓN
      #FA sigmoide y su derivada para el backpropagation
      sigm = (lambda x: 1/ (1 + np.e ** (-x)),
              lambda x: x * (1 - x))
      relu = lambda x: np.maximum(0,x)
      def create_redneu(neuronas_capas, funcion_activacion):
          redneu = []
          #recorre todo el bucle for hasta el último elemento
          for l, layer in enumerate(neuronas_capas[:-1]):
              redneu.append(capa_red(neuronas_capas[l], neuronas_capa
s[l+1], funcion_activacion))
          return redneu
      neuronas_capas = [entradas,4,8,4,4]
      red_neuronal = create_redneu(neuronas_capas, sigm)
      #Funcion de error o coste, y su derivada para el
backpropagation
      funcion_coste = (lambda Yp, Yr: np.mean((Yp- Yr)**2),
                  lambda Yp, Yr: (Yp - Yr))
      #Funcion de operaciones de la red neuronal
      def operar(red_neuronal, X, Y, funcion_coste, lr=0.5, entrenar=
True):
          #guardamos los valores de salida para la correción de
parametros.
          salida_neu = [(None, X)] #X: matriz de entrada
          ############### forward pass ###############
          #suma ponderada (pesos y ballas) --> función de activación
          for l, layer in enumerate(red_neuronal):
```

```
51.                 z_suma = salida_neu[-1][1]@red_neuronal[l].w +
    red_neuronal[l].b
52.                 # [-1] output anterior
53.                 # [1] escogemos el parámetro "a" de "salida_neu"
54.                 a = red_neuronal[l].funcion_activacion[0](z_suma)
55.                 salida_neu.append((z_suma,a))
56.             if entrenar:
57.                 #### Metodo de backpropagation ####
58.                 # Derivadas parciales descritas en el informe #
59.                 #### Backward pass ####
60.                 deltas = [ ]
61.                 for l in reversed (range (0, len(red_neuronal))):
62.                     z_suma = salida_neu[l+1] [0]
63.                     a = salida_neu[l+1] [1]
64.                     if l == len(red_neuronal)-1:
65.                         #Calcular el delta de la última capa
66.                         deltas.insert(0, funcion_coste[1](a, Y) *
    red_neuronal[1].funcion_activacion[1](a))
67.                     else:
68.                         deltas.insert(0,deltas[0]@ _w.T *
    red_neuronal[1].funcion_activacion[1](a))
69.                         #Calcular el delta respecto a la capa previa
70.                     _w = red_neuronal[l].w
71.                     ############### Gradient descent ###############
72.                     red_neuronal[l].b= red_neuronal[l].b -
    np.mean(deltas[0], axis=0, keepdims=True) * lr
73.                     red_neuronal[l].w= red_neuronal[l].w -
    salida_neu[l][1].T@deltas[0] * lr
74.             return salida_neu[-1][1]
75.         nueva_red = create_redneu(neuronas_capas, sigm)
76.         perdidas = []
77.         for i in range(2500):
78.           # Entrenemos a la red!
79.           pY = operar(nueva_red, X, Y, funcion_coste, lr=0.1)
80.           if i % 25 == 0:
81.             print(pY)
82.             perdidas.append(funcion_coste[0](pY, Y))
83.             res = 50
84.             #si se cambia el dataset ajustar estos valores
85.             _x0 = np.linspace(-1.5, 1.5, res)
86.             _x1 = np.linspace(-1.2, 1.2, res)
87.             _Y = np.zeros((res, res))
88.             for i0, x0 in enumerate(_x0):
89.               for i1, x1 in enumerate(_x1):
90.                 _Y[i0, i1] = operar(nueva_red, np.array([[x0, x1]]), Y,
     funcion_coste, entrenar=False)[0][0]
91.             plt.pcolormesh(_x0, _x1, _Y, cmap="coolwarm")
92.             plt.axis("equal")
93.             plt.scatter(X[Y[:,0] == 0, 0], X[Y[:,0] == 0, 1], c="yellow
    ")
94.             plt.scatter(X[Y[:,0] == 1, 0], X[Y[:,0] == 1, 1], c="lightg
    reen")
95.             clear_output(wait=True)
96.             plt.show()
97.             plt.plot(range(len(perdidas)), perdidas)
98.             plt.show()
99.             time.sleep(0.5)
```