UNIVERSIDAD AUTÓNOMA DEL ESTADO DE MÉXICO

CENTRO UNIVERSITARIO UAEM ECATEPEC

Red neuronal artificial para estimar el índice UV.

TESIS

PARA OBTENER EL GRADO DE

MAESTRO EN CIENCIAS DE LA COMPUTACIÓN

P R E S E N T A:

Ing. Jonathan Romero Rodriguez

Director De Tesis: Dr. En C. Rodolfo Zolá García Lozano

Co-Director De Tesis: Dr. En C. De La Comp. Juan De Jesús Amador Reyes

Tutor De Tesis: Dra. En Sc. Alejandra Morales Ramírez

Beca SECIHTI CVU: 1304879

Ecatepec de Morelos, Estado de México, enero 2026

Índice

Introducción 1
Planteamiento del problema 1
Objetivo 2
Capítulo 1: Marco teórico 3
Neurona biológica 3
Historia del Perceptrón 4
Perceptrón simple 5
Suma ponderada y funciones de activación 6
Perceptrón multicapa 8
Índice UV 9
Capítulo 2: Metodología 12
Metodología de prototipos 12
Multilayer perceptrón (MLP) 15
Desarrollo del proyecto. 18
Prototipo 1 19
Análisis de requerimientos 19
Diseño 19
Diagrama a bloques. 19
Diagrama eléctrico. 20
Diagrama de flujo. 21
Implementación 23
Resultados 26
Prototipo 2 27
Análisis de requerimientos 27
Diseño 28
Diagrama a bloques. 28
Diagrama eléctrico. 29
Diagrama de flujo. 30
Implementación 32
Resultados 34
Modelado del índice UV en función de la medición de radiación visible 35
Prototipo 3 39
Análisis de requerimientos 39

Diseño del prototipo 39
Construcción rápida 41
Preparación de los datos 41
Normalización de los datos 42
División del conjunto de datos. 44
Creación de la red neuronal 45
Entrenamiento de la red neuronal 45
Evaluación del modelo 46
Resultados e implementación 47
Prototipo 4 50
Análisis de requerimientos 50
Diseño 50
Diagrama a bloques. 50
Diagrama eléctrico. 51
Diagrama de flujo. 52
Implementación 54
Resultados 60
Conclusiones 64
Trabajos futuros 64
Referencias. 66
Anexos. 67
Anexo A: Códigos Fuentes 67
A.1 Código en Arduino para la medición de la intensidad luminosa. 67
A.2 Código en Arduino para la Estación de Monitoreo de Índice UV 70
A.3 Procedimiento de Extracción de Datos 76
A.4 Código en Python de la red neuronal. 79
A.5 Despeje de la variable X en la formulade Z-score. 81
A.6 Código en Arduino para la implementación de la RNA 81
A.7 Mapas de calor de cada mes. 86
Anexo B: Instalación de Librerías en Arduino IDE (Opcional) 88
B.1 Instalación de la librería DHT11 y DHT22 88
B.2 Instalación de la librería RTClib 89

Introducción

Planteamiento del problema

La luz solar es fundamental para la vida en la Tierra, ya que proporciona beneficios como la síntesis de vitamina D. Sin embargo, también conlleva riesgos importantes debido a la exposición a ciertas radiaciones presentes en su espectro, específicamente la radiación ultravioleta (UV) y la radiación infrarroja (IR), que no son visibles para el ojo humano.

De estas, la radiación ultravioleta (UV) ha sido identificada como una de las principales causas de neoplasias cutáneas, con estimaciones que sugieren que entre el 50% y el 90% de estos tipos de cáncer son inducidos por la exposición a la radiación UV. En México, se ha observado un preocupante incremento en los casos de cáncer de piel, lo que subraya la urgencia de abordar este problema de salud pública [1].

Una de las principales causas radica en el desconocimiento generalizado de los riesgos asociados con la exposición prolongada a la radiación UV. Aunque existen herramientas que informan sobre el índice UV en diversas regiones del país, estas son poco consultadas. Esta falta de acceso a información oportuna y accesible sobre los niveles de radiación UV en los espacios públicos contribuye a que las personas subestimen la importancia de utilizar productos o artículos de protección solar [2].

En respuesta a esta necesidad, el presente estudio propone el desarrollo de un dispositivo portátil de medición de radiación UV. Este dispositivo será diseñado para ser utilizado por cualquier persona en su vida cotidiana, permitiéndole tomar decisiones informadas sobre su exposición al sol. Al proporcionar información inmediata y accesible sobre los niveles de radiación UV en diferentes entornos, este dispositivo podría desempeñar un papel crucial en la prevención de cánceres de piel y en la promoción de hábitos saludables de protección solar entre la población.

Con esta investigación, se busca desarrollar una herramienta práctica, para también sensibilizar a la población sobre los riesgos de la radiación UV para reducir la incidencia de neoplasias cutáneas en México. En tal sentido el objetivo de la investigación es:

Objetivo

Diseñar e implementar un sistema innovador capaz de predecir en tiempo real los niveles de radiación UV, a partir de información obtenida por un sensor de luz visible, empleando el poder de las redes neuronales artificiales.

Objetivos específicos.

I. Construir un dataset estructurado que almacene y organice los datos obtenidos por los sensores de luz visible, destinado al entrenamiento y validación de la red neuronal.
II. Diseñar y entrenar una red neuronal artificial capaz de estimar de manera precisa los niveles de radiación UV a partir de la información captada por un sensor de luz visible.

Capítulo 1: Marco teórico

Neurona biológica

El cerebro es el órgano responsable de enviar señales y coordinar acciones en todo el cuerpo para realizar funciones específicas. Está conformado por aproximadamente 100 mil millones de neuronas, las cuales constituyen las unidades básicas del sistema nervioso.

Una neurona es una célula especializada que recibe, procesa y transmite información hacia otras neuronas, con el objetivo de generar una respuesta dentro del sistema nervioso. Morfológicamente, presenta una forma esférica y puede medir entre 5 y 10 micras de diámetro. Cuando las neuronas se agrupan e interactúan entre sí, forman un sistema de comunicación capaz de aprender de acuerdo con la interacción con el entorno, estimulando nuestros sentidos mediante el intercambio de señales. A este conjunto interconectado se le denomina red neuronal [3].

Desde un punto de vista funcional, las neuronas pueden dividirse en tres partes principales, cada una con un papel específico dentro del procesamiento de la información:

- Dendritas: Constituyen el canal de entrada (input), encargadas de recibir información proveniente del exterior o de otras neuronas. Su función principal es conducir esos impulsos hacia el soma para su procesamiento. Morfológicamente, se asemejan a un árbol receptor de señales.
- Soma: También conocido como cuerpo celular o centro de cómputo, es la región donde se procesa la información recibida. En esta etapa, se realizan operaciones de suma con funciones no lineales. Una vez procesada, la información es transmitida al axón en forma de estímulos nerviosos.
- Axón: Representa el canal de salida (output). Es una prolongación tubular que transporta los impulsos nerviosos desde el soma hacia otras neuronas o directamente hacia un músculo, mediante sus ramificaciones [4].

La información se transmite de una neurona a otra a través de la conexión entre el axón de una neurona y las dendritas de otra, conocida como sinapsis. Esta conexión determina la fuerza y tipo de relación existente entre ambas neuronas, lo cual se ajusta biológicamente mediante el aprendizaje. Las sinapsis pueden ser excitadoras o inhibidoras, dependiendo de la naturaleza de la señal.

La transmisión sináptica puede producirse de dos maneras:

- Eléctrica, que ocurre en el interior de la neurona y consiste en la propagación de impulsos eléctricos a lo largo del axón.
- Química, que se produce en el espacio sináptico mediante neurotransmisores, sustancias químicas que cruzan el vacío sináptico para llegar a las dendritas de

la neurona receptora. Este proceso se repite de manera continua hasta que la información alcanza su destino final [5].

Con el paso del tiempo, surgió el interés por imitar el funcionamiento biológico de las neuronas con el propósito de resolver problemas computacionales complejos. Este esfuerzo dio origen al perceptrón, una neurona artificial capaz de emular el proceso de aprendizaje mediante un tratamiento matemático de la información. Como se puede observar en la Fig. 1 se puede ver la comparación entre una neurona biológica y una neurona artificial.

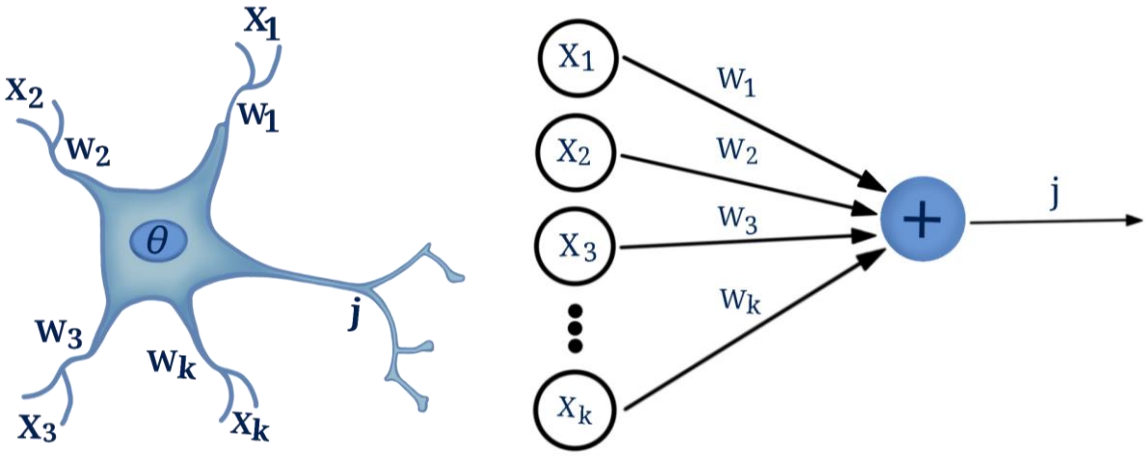


Fig. 1.- Neurona biológica y neurona artificial.

Historia del Perceptrón

En la siguiente tabla se describe las fechas más importantes de cómo fue la creación y evolución del Perceptrón.

Cronología de las redes neuronales [6], [7], [8].

Año	Evento
1936	Alan Turing es la primera persona en estudiar el cerebro como una computadora.
1943	Warren McCulloch y Walter Pitts crean una teoría acerca de cómo trabajan las neuronas y fueron los primeros en modelar una red neuronal simple con circuitos eléctricos.
1949	Donal Hebb explico los procesos de aprendizaje desde el punto psicológico, así desarrollando una regla de aprendizaje. La idea fue que el aprendizaje ocurría cuando una neurona mostraba ciertos cambios y este trabajo formaron las bases de la teoría de las redes neuronales.
1950	Karl Lashley en sus ensayos, hallo que la información no se almacenaba de forma centralizada en el cerebro si no que se distribuía en él.
1956	Da inicio la primera conferencia de la inteligencia Artificial en la iglesia de Dartmouth.

1957	Frank Rosenblatt desarrolla el Perceptrón, la red neuronal más antigua. Este modelo era capaz de reconocer patrones similares a los que aprendió. Este Perceptrón tenía limitaciones ya que no era capaz de clasificar clases no separables linealmente.
1959	Frank Rosenblatt crea el teorema de convergencia del Perceptrón, en donde, bajo ciertas condiciones, el Perceptrón convergía hacia un estado finito.
1960	Bernard Widroff y Marcian Hoff crean el modelo Adaline (ADAptative LINear Elements). Fue la primera red neuronal que se aplicó a un problema real el cual ayudo a eliminar ecos en las líneas telefónicas.
1969	Marvin Minsky y Seymour Papert probaron matemáticamente que el Perceptrón no resolvía problemas fáciles como la función no lineal. Esto afirmo que era muy débil el Perceptrón dado que en la computación y en el mundo real se usan funciones no lineales.
1974	Paul Werbos desarrollo el algoritmo de aprendizaje backpropagation.
1986	David Rumelhart y G. Hinton redescubrieron el algoritmo backpropagation dándole un significado.
1997	Se crear una computadora capaz de jugar ajedrez llamada “Deep Blue”.
2009	En este año se crea la base de datos ImageNet, con aproximadamente 14 millones de imágenes, lanzada por Fei-Fei con el fin de entrenar redes neuronales para la clasificación de fotografías e identificación de objetos.
2014	Se introducen las redes generativas adversarias, utilizadas para la generación de imágenes, videos y voz. En este mismo año, Amazon presenta a su asistente virtual Alexa.
2016	Se publican las librerías TensorFlow y PyTorch, las cuales se convierten en algunas de las más utilizadas para el desarrollo de redes neuronales.
2017	Google desarrolla AlphaGo, una inteligencia artificial capaz de jugar el juego de mesa Go.

Perceptrón simple

El Perceptrón (Fig. 2) es considerado la red neuronal más antigua y fue utilizado como un identificador de patrones. Este modelo tenía la capacidad de aprender una serie de patrones y posteriormente reconocer otros patrones que eran iguales sin la necesidad de un entrenamiento.

Este perceptrón se utilizaba para las compuertas lógicas, Sin embargo, la desventaja de este modelo es que tenía limitaciones para resolver problemas lineales como la función XOR, debido a que este no era un problema lineal [4].

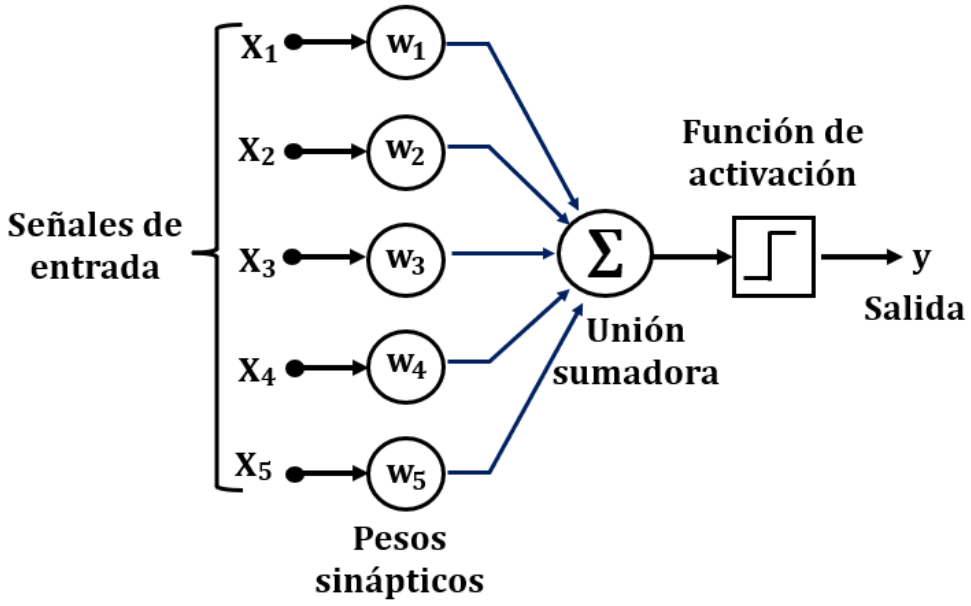


Fig. 2.- Perceptrón simple.

Suma ponderada y funciones de activación

El funcionamiento de una neurona artificial requiere de dos procesos fundamentales: la suma ponderada de las entradas y el uso de una función de activación.

La suma ponderada es el cálculo del producto de cada entrada x_i por su peso w_i más el valor adicional el cual se llama sesgo (b), Este proceso se expresa matemáticamente de la siguiente manera:

$$z = b + \sum_i w_i x_i$$

Las funciones activación trabajan sobre el valor obtenido de la suma ponderada, esto a través de una función en donde el resultado será la salida de la neurona. Dependiendo la complejidad y el problema a resolver, se puede escoger una función de activación la cual se adecue a lo que estamos buscando [9].

Una de las funciones más usadas es la función escalón (Fig. 3). Esta función trabaja de forma binaria, esto se debe a que, si el resultado de la suma es mayor o igual a 0 la activación será 1, pero en caso contrario que la suma sea menor a 0 la activación será 0.

y = h(x) donde x = {-1, 1} o {0, 1}

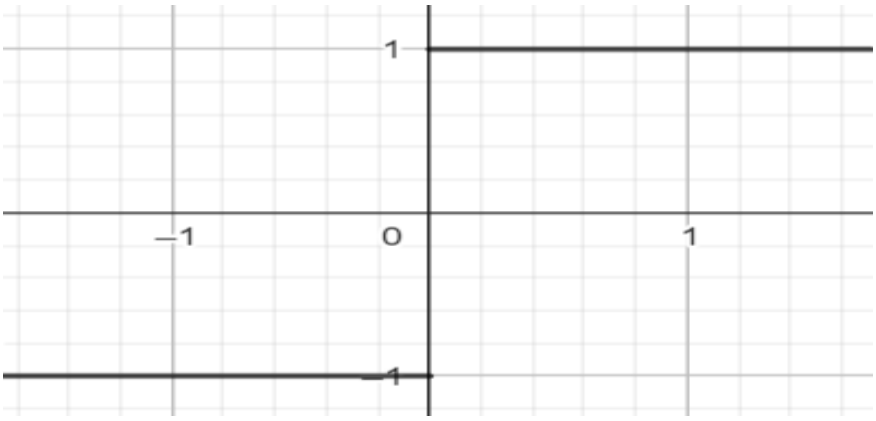


Fig. 3.- Función escalón.

En la función lineal o mixta (Fig. 4) trabaja de diferente manera que la función escalonada, esto debido a que la función lineal trabaja con límites. Si el resultado de la suma es menor al límite inferior la activación será 0, en caso contrario si la suma es igual o mayor al límite superior la activación será 1. Puede haber un caso más en que si el resultado de la suma esta entre ambos limites la activación se presenta en forma lineal.

$$y = \begin{cases} -1, si\ x < -1 \\ x, si\ 1 \leq x \leq -1 \\ 1, si > 1 \end{cases}$$ donde x = [-1, 1]

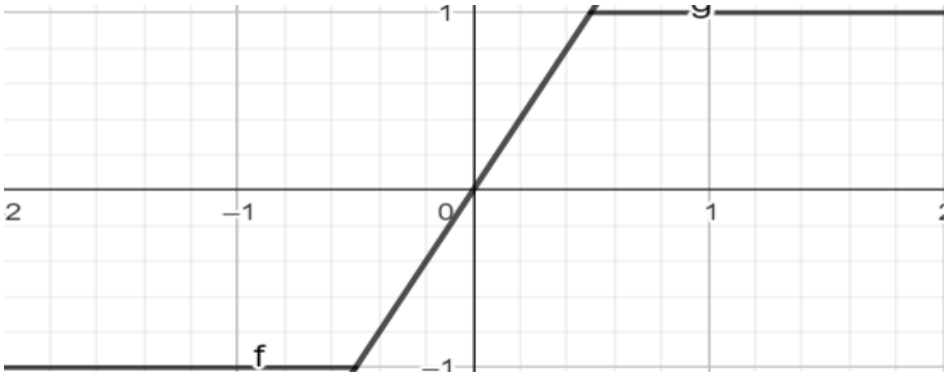


Fig. 4.- Función lineal o mixta.

La función sigmoidal (Fig. 5) está definida por posibles valores de entrada esto dentro de un intervalo y este funciona a través de un incremento monotónico y que tenga limites superiores e inferiores. Con la función sigmoidal el valor de salida puede estar en la zona alta o baja, en caso de que la pendiente de esta función es elevada tiende a la función escalón con la diferencia que la derivada siempre es positiva y cercana al 0.

$$y = \frac{1}{1 + e^{-x}}$$ donde x = [0, 1] o [-1, 1]

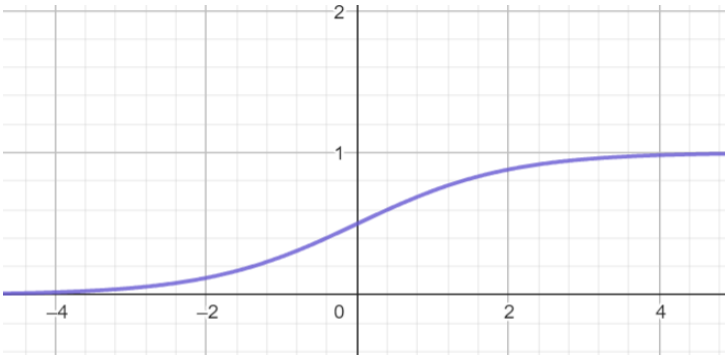


Fig. 5.- Función sigmoidal.

La función gaussiana (Fig. 6) requiere de dos capas ocultas que estén entre la capa de entrada y de salida esto a que los centros y anchuras de esta función se pueden adaptar más que la función sigmoidal. A veces esta función se puede realizar con un solo nivel en redes neuronales [10].

$$y = e^{-x^2}$$ donde x = [0,1]

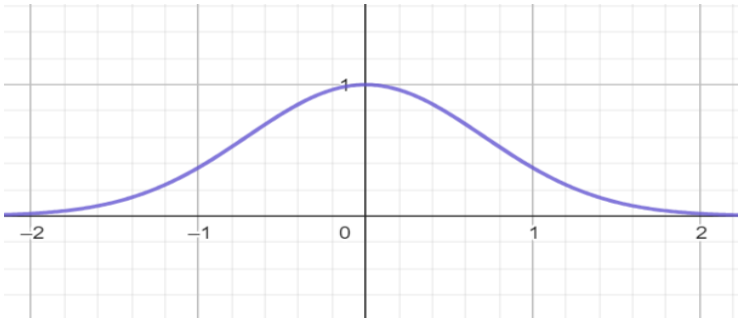


Fig. 6.- Función gaussiana.

Perceptrón multicapa

Como se observa en la Figura 7, el perceptor multicapa es una red neuronal la cual tiene una capa de entrada, una o varias capas ocultas y una capa de salida, lo que lo diferencia del Perceptrón simple ya que las capas ocultas pueden ayudar a resolver problemas no lineales. El tamaño y numero de capas dependerá de la complejidad que se desee solucionar.

La función del perceptrón multicapa es recibir los datos de la capa de entrada para su procesamiento. En las capas ocultas se realiza la suma ponderada de acuerdo con el valor introducido, tomando en cuenta los pesos y a este resultado pasas a una función de activación y el resultado pasa a la siguiente capa. Si la siguiente capa es oculta se repite el proceso, en caso contrario pasa directo a la capa de salida. Finalmente, la capa de salida recibe estos datos y hace el mismo proceso con la diferencia que el resultado es la solución del problema planteado [11].

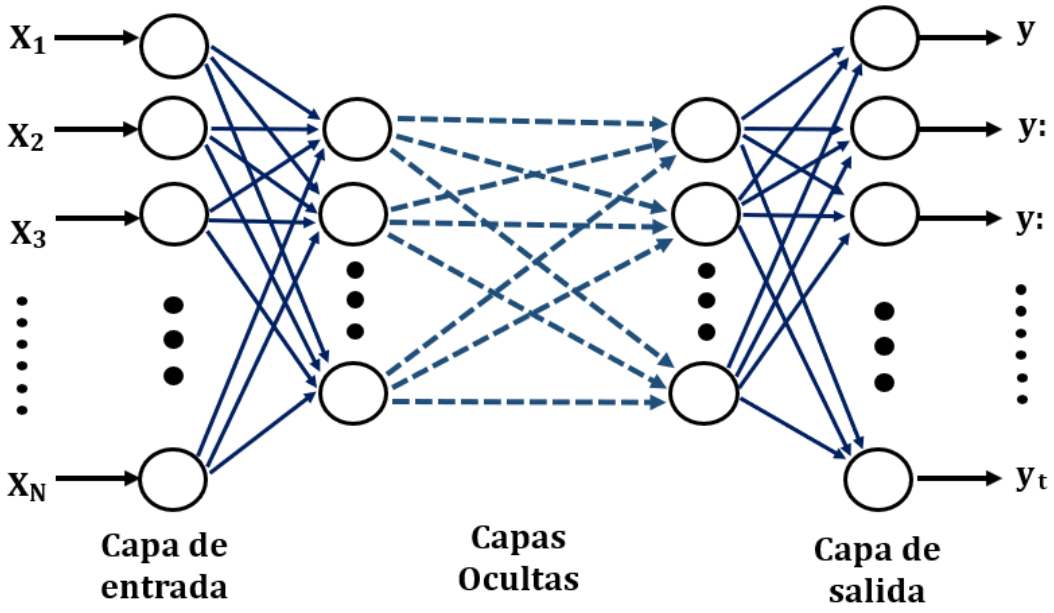


Fig. 7.- Perceptrón multicapa.

Índice UV

La radiación solar es la que modula el clima en el planeta; esta radiación tiene efectos beneficiosos, pero puede resultar peligrosa si se exceden ciertos límites. La radiación ultravioleta (UV) son los rayos provenientes del Sol que, en cantidades pequeñas, resultan beneficiosos para el ser humano debido a que estimulan la producción de vitamina D.

Estos rayos UV se encuentran entre las longitudes de onda de 100 a 400 nm y se dividen en tres bandas [12]:

- UVA (315–400 nm)
- UVB (280–315 nm)
- UVC (100–280 nm)

Las tres bandas llegan a la atmósfera terrestre, pero el ozono, el vapor de agua, el oxígeno y el dióxido de carbono absorben los rayos UVC y aproximadamente el 90 % de los rayos UVB, mientras que los rayos UVA son los que más traspasan la atmósfera, alcanzando con mayor facilidad la superficie terrestre.

La intensidad de la radiación UV depende de diversos factores, entre los cuales se encuentran:

- Altura del Sol: Cuando el sol se encuentra más alto en el cielo, la radiación UV es más intensa. Esto varía según la hora del día y la época del año. La mayor

intensidad ocurre al mediodía y durante el verano, especialmente fuera de las zonas tropicales.

- Latitud: Cuanto más cerca se esté del ecuador, mayor será la intensidad de la radiación UV.
- Nubosidad: En días nublados, la radiación disminuye; sin embargo, en ciertos casos puede mantenerse alta debido al efecto de reflexión de las nubes.
- Altitud: A mayor altitud, la atmósfera absorbe menos radiación UV, por lo que por cada kilómetro de altitud la intensidad aumenta entre un 10 % y un 12 % aproximadamente.
- Ozono: El ozono atmosférico es fundamental para la absorción de la radiación UV. Su concentración varía durante el año e incluso a lo largo del día.
- Reflexión del suelo: Dependiendo de la superficie, algunos lugares reflejan más radiación UV que otros. Por ejemplo, la arena de playa puede reflejar un 15 %, la espuma del mar un 25 %, y la nieve hasta un 80 % [13], [14].

La exposición a la radiación UV puede provocar efectos agudos y crónicos en la piel, los ojos y el sistema inmunitario.

- Efectos agudos: Incluyen las quemaduras solares y el bronceado. La exposición excesiva y prolongada puede causar envejecimiento prematuro de la piel debido a la degeneración de las células, del tejido fibroso y de los vasos sanguíneos.
- Efectos crónicos: Los más importantes son el cáncer de piel y las cataratas. A nivel mundial, se estiman entre dos y tres millones de casos de cáncer de piel, y entre 12 y 15 millones de personas padecen ceguera por cataratas. La Organización Mundial de la Salud (OMS) estima que aproximadamente el 20 % de estos casos pueden estar relacionados con la exposición solar, especialmente en países cercanos al ecuador [14].

El Índice Ultravioleta (IUV) indica el nivel de radiación solar ultravioleta que llega a la superficie terrestre. Este índice se mide en una escala que va del 1 al 11+, y utiliza un semáforo de colores desarrollado por la Organización Mundial de la Salud (OMS) [15].

El valor del IUV se expresa como un número mayor que cero; entre más alto sea, mayor será la probabilidad de sufrir lesiones cutáneas y oculares. Este índice es de gran importancia para fomentar la conciencia pública sobre los riesgos de la exposición excesiva a la radiación UV y promover medidas de prevención.

El IUV varía según factores como la ubicación geográfica, la época del año y la hora solar. En muchos países, los valores del IUV se difunden mediante medios de comunicación como periódicos, televisión y radio, principalmente durante el verano.

Los valores del IUV se presentan como números enteros, redondeados al valor más próximo, y se clasifican en las siguientes categorías de exposición:

Categoría de exposición	Intervalos de valores del IUV
Baja	< 2
Moderada	3 a 5
Alta	6 a 7
Muy alta	8 a 10
Extremadamente alta	11+

El objetivo final del IUV es que la población lo considere como una información diaria y útil. Los mensajes deben ser claros y sencillos para que las personas los adopten como hábitos saludables, teniendo en cuenta las condiciones climáticas y culturales de cada región.

Una forma efectiva de comunicar esta información es mediante un diagrama visual (Fig. 8), que muestra una escala de colores indicando el nivel de riesgo, las recomendaciones de protección y las actividades seguras según la intensidad de la radiación UV [14].

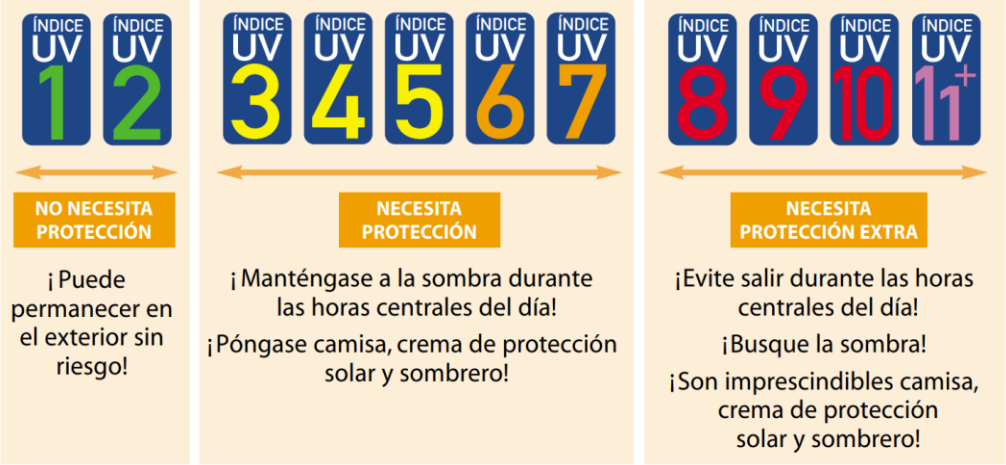


Fig. 8.- Sistema de protección solar recomendado [12].

Capítulo 2: Metodología

La metodología se define como el conjunto de métodos y estrategias que permiten planificar, ejecutar y controlar de manera efectiva el desarrollo de un proyecto. Su aplicación abarca desde el inicio hasta la finalización del proyecto, sirviendo como una guía estructurada para alcanzar los objetivos propuestos dentro de los plazos establecidos.

El propósito principal de una metodología es proporcionar un marco estratégico que incremente las probabilidades de éxito, optimizando los recursos y garantizando la coherencia en el proceso. Esto no solo implica la definición de las fases del proyecto, sino también las tareas específicas que deben realizarse en cada etapa.

La importancia de implementar una metodología radica en los siguientes beneficios:

- Eficiencia y reducción de errores: Al seguir procedimientos claramente definidos, se minimizan los errores y se ahorra tiempo y recursos.
- Estandarización de procesos: Garantiza que el proyecto se desarrolle de forma lógica y estructurada, facilitando la comunicación entre los involucrados.
- Apoyo en la toma de decisiones: Proporciona herramientas y criterios para gestionar de manera efectiva cada fase del proyecto.

Es fundamental reconocer que la metodología debe ser flexible. Esto significa que el desarrollador puede adaptar sus diferentes etapas según las necesidades y circunstancias específicas del proyecto. Esta adaptabilidad permite ajustar el plan de acción y la gestión del proyecto para abordar desafíos imprevistos y maximizar la efectividad.

En la siguiente sección se describirá la metodología de prototipos. En esta sección se abordará las condiciones para su aplicación, las etapas de la metodología, así como las ventajas y desventajas de usar este tipo de metodología.

Metodología de prototipos

La metodología de prototipos, también conocida como modelo de prototipos, consiste en la creación de una representación inicial y funcional del producto final. Su principal objetivo es permitir la prueba y validación del prototipo antes de desarrollar la versión definitiva del producto.

Este modelo es ampliamente aplicable en diferentes tipos de proyectos, como el desarrollo de sistemas o aplicaciones. Una de sus ventajas clave es que facilita la construcción de prototipos iniciales con un conjunto mínimo de requisitos, lo que permite evaluar su funcionamiento y verificar que los componentes principales operen correctamente.

Además, cada prototipo puede ser sometido a pruebas y ajustes basados en la retroalimentación proporcionada por el cliente o el equipo de desarrollo. Esto hace que la metodología sea iterativa, permitiendo realizar modificaciones para mejorar tanto el desempeño como las características del prototipo.

El prototipo final resultante no solo refleja las necesidades y expectativas identificadas durante el proceso, sino que también se convierte en la base sobre la cual se construye el producto definitivo del proyecto.

El modelo de prototipos se utiliza principalmente cuando no se tienen claros todos los detalles del desarrollo del proyecto o cuando este presenta un alto grado de complejidad. También es ideal en casos donde no existe experiencia previa en el desarrollo de proyectos similares o se está abordando un problema completamente nuevo.

Una de las principales ventajas de esta metodología es su flexibilidad. Permite iniciar el desarrollo del proyecto a partir de un prototipo preliminar con los recursos disponibles, incluso cuando los requisitos no están completamente definidos. Este enfoque facilita un proceso iterativo, en el que los requisitos iniciales se priorizan para construir un prototipo funcional, que luego puede ser mejorado en futuras iteraciones. Este ciclo permite incorporar gradualmente los demás requisitos según su importancia.

El proceso de desarrollo por prototipos no solo ayuda a entender mejor el funcionamiento del proyecto, sino que también proporciona información valiosa al documentar los resultados y los errores detectados en cada iteración. Esta retroalimentación es crucial para refinar y mejorar continuamente el prototipo, asegurando que se acerque a los objetivos esperados.

Al adoptar esta metodología, se incrementan significativamente las probabilidades de éxito del proyecto, ya que permite identificar y resolver problemas desde las etapas iniciales. Esto garantiza que el producto final sea funcional y cumpla con los estándares de calidad y las expectativas del usuario.

Las etapas de este modelo metodológico pueden adaptarse o variar según las necesidades específicas de cada proyecto. Para el desarrollo de este trabajo, se han definido cuatro etapas principales, diseñadas para guiar el proceso de manera estructurada y efectiva, como se muestra en la Fig. 9. A continuación, se describe en detalle cada una de estas etapas.

Análisis de requerimientos:

En esta etapa se estudia el producto a desarrollar. Se identifican las ideas y posibilidades del proyecto, así como los requerimientos específicos y el objetivo principal o funcionalidad del producto. Este análisis permite establecer una base para la creación de un prototipo preliminar que refleje las funciones o características básicas del sistema.

Diseño:

En esta etapa se elabora un modelo preliminar, que puede representarse de manera física o digital. Con el fin de facilitar el desarrollo, se plantea dividir el prototipo inicial en pequeños prototipos funcionales que puedan trabajarse de forma modular. Este enfoque permite abordar de manera más eficiente los desafíos del desarrollo y asegura que cada componente cumpla con su propósito.

Implementación:

Durante esta fase, se lleva a cabo la implementación de uno de los prototipos diseñados. Este prototipo, ya sea físico o digital, debe ser funcional para realizar pruebas que permitan validar las ideas propuestas. Se realiza un análisis de los requerimientos específicos del prototipo en cuestión, asegurando que el diseño cumpla con las expectativas planteadas. Además, se recopila retroalimentación para ajustar o mejorar el prototipo según sea necesario.

Resultados:

El prototipo implementado se somete a pruebas y evaluaciones para recopilar información sobre su desempeño y funcionalidad. Los resultados obtenidos permiten identificar las ideas que deben mantenerse, así como aquellas que necesitan ser ajustadas o eliminadas. Este proceso iterativo es fundamental para garantizar la calidad del producto final.

Producto terminado:

Finalmente, se crea el producto final integrando los aprendizajes y avances logrados a partir de los prototipos desarrollados en las etapas previas. Esto permite obtener un producto que cumple con los requerimientos principales del proyecto, respondiendo de manera adecuada a las necesidades identificadas durante el análisis inicial.

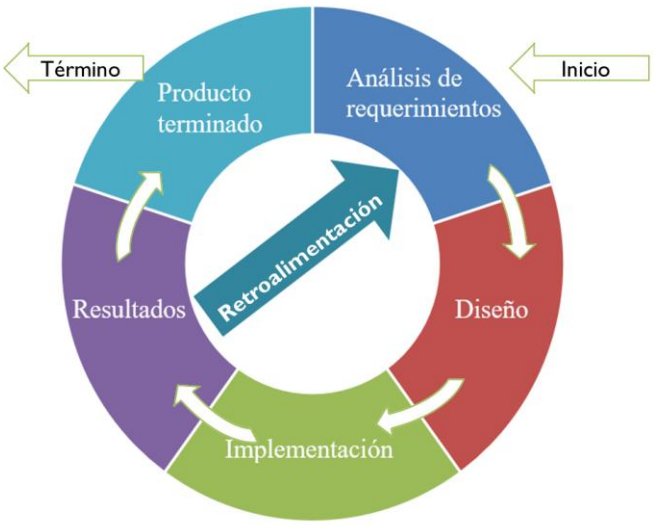


Fig. 9.- Metodología de prototipos.

Aunque este modelo es útil para abordar proyectos con requisitos inciertos o poco definidos, también presenta ciertas desventajas que deben ser consideradas antes de su implementación.

En primer lugar, la complejidad del proyecto puede afectar significativamente el tiempo necesario para su desarrollo. Debido a la naturaleza iterativa de esta metodología, no siempre es posible estimar con precisión cuánto tiempo llevará obtener un producto final satisfactorio.

Además, una de las principales limitaciones radica en la incertidumbre sobre el número de iteraciones requeridas. Dado que cada ciclo de prototipado busca ajustar y mejorar el diseño, es difícil prever cuántos ciclos serán necesarios para alcanzar un resultado aceptable, lo que puede derivar en retrasos y un incremento en los costos del proyecto.

Por último, esta falta de previsibilidad puede generar desafíos adicionales, como dificultades para planificar recursos y establecer cronogramas claros, lo que podría impactar en la eficiencia.

A pesar de estas desventajas, la metodología de prototipos sigue siendo una herramienta valiosa cuando se utiliza de manera estratégica y en los contextos adecuados.

Multilayer perceptrón (MLP)

En este proyecto se ha optado por el uso de redes neuronales debido a su capacidad para modelar relaciones complejas entre variables, lo que resulta ideal para nuestra hipótesis: a mayor radiación visible, mayor será la radiación UV.

Este enfoque permite aprovechar sensores LDR, que no están diseñados específicamente para medir radiación UV, para estimar de manera confiable este tipo de radiación. Gracias a la red neuronal, será posible procesar los datos captados por los sensores y obtener estimaciones precisas de la radiación UV. Por lo que a continuación se describe la metodología que se utilizará para el entrenamiento de la red neuronal que nos permitirá estimar la radiación UV.

Como se mencionó en el marco teórico, las redes neuronales se asemejan al cerebro humano, lo que permite replicar, en cierta medida, sus funciones utilizando computadoras. Estas redes poseen un amplio campo de aplicaciones y son capaces de resolver problemas en diversos ámbitos como la investigación, el comercio y la industria.

El alcance de las redes neuronales abarca desde soluciones simples hasta complejas, utilizando tanto software como hardware para su implementación. Una de las principales características de estas redes es su capacidad de aprendizaje, lo cual es posible gracias al empleo del algoritmo de retropropagación (backpropagation). Este algoritmo permite ajustar los pesos de cada capa dentro de la red neuronal, optimizando su desempeño.

Para el desarrollo de este trabajo, se empleará el ciclo de vida de una red neuronal que se muestra en la F, que consta de dos momentos fundamentales en su procesamiento: el momento de aprendizaje y el momento de utilización.

Este ciclo de vida está compuesto por cuatro etapas principales, descritas a continuación (Fig. 10):

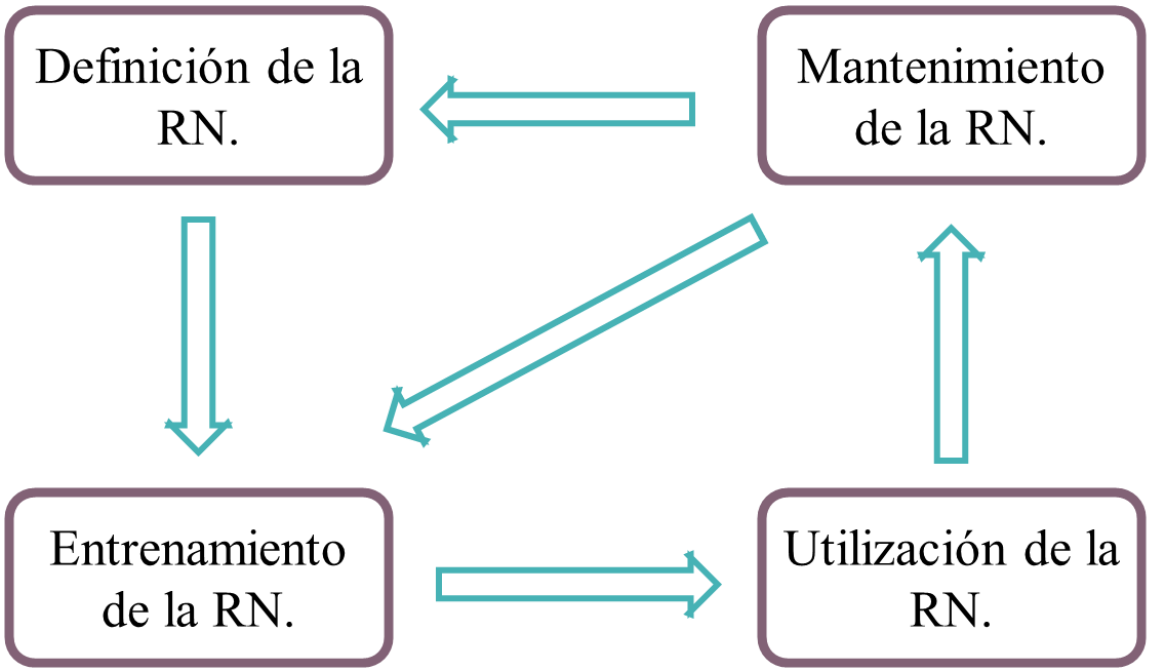


Fig. 10.- Ciclo de vida de una red neuronal artificial.

Definición de la red neuronal

En esta etapa se realiza la configuración inicial de la red neuronal. Se selecciona el tipo de red que se utilizará, así como las variables que servirán para resolver el problema planteado. Es fundamental considerar los siguientes aspectos:

- **Tamaño de la red:** cantidad de capas ocultas y número de neuronas en cada capa. No existe una regla fija para determinar la cantidad exacta de neuronas, ya que depende del problema a resolver.
- **Naturaleza del problema:** tipo de salida que se espera y el tipo de aprendizaje que se aplicará (supervisado o no supervisado).
- **Funciones de activación:** se deben definir las funciones de activación para cada capa oculta y para la capa de salida.
- **Algoritmos de entrenamiento:** elección de los algoritmos que ajustarán los pesos de las conexiones neuronales.

Esta etapa es crucial, ya que la estructura y configuración de la red determinarán su capacidad para resolver el problema planteado.

Entrenamiento de la red neuronal
Durante esta fase, se aplican los métodos de entrenamiento utilizando los algoritmos de aprendizaje previamente definidos. El entrenamiento consiste en ajustar los pesos de las conexiones entre las neuronas de las capas ocultas y de salida, con el fin de optimizar el rendimiento de la red. Cabe destacar que este ajuste no se realiza en la capa de entrada.

El proceso de ajuste de pesos incluye las siguientes operaciones:

1. Realizar la suma ponderada de los pesos de entrada.
2. Pasar el resultado obtenido a través de la función de activación asignada a cada capa.

El objetivo del entrenamiento es minimizar los errores en las predicciones y preparar la red para trabajar con nuevos datos.

Utilización de la red neuronal
En esta etapa, la red neuronal se somete a pruebas utilizando datos de entrada que no formaron parte del entrenamiento. Esto permite evaluar su capacidad para generar resultados precisos y útiles. A diferencia del entrenamiento, en esta fase los pesos no se ajustan, ya que el objetivo es observar el desempeño de la red en condiciones reales.

Mantenimiento de la red neuronal
En esta etapa final, se validan los resultados obtenidos por la red neuronal para determinar si logró resolver el problema planteado. El mantenimiento implica realizar validaciones continuas para garantizar que la red funcione correctamente a lo largo del tiempo.

Además, si se dispone de nuevos datos relevantes, el ciclo puede repetirse desde la etapa de entrenamiento para ajustar la red y mejorar su desempeño. Este proceso iterativo asegura que la red neuronal se mantenga actualizada y relevante frente a cambios en los datos o en el contexto del problema.

Desarrollo del proyecto.

Como se mencionó anteriormente, el presente proyecto tiene como objetivo desarrollar una estación de monitoreo del índice UV utilizando redes neuronales artificiales. Como primera instancia se realizado el análisis de requerimientos del prototipo funcional en el cual se identificaron los siguientes aspectos:

- Se busca implementar un sistema innovador que permita estimar la radiación UV a partir de mediciones de luz visible, sin necesidad de sensores especializados en UV.
- Se utilizarán fotorresistencias para la medición de la luz visible y sensores UV para la medición de rayos UV.
- Obtener un conjunto de datos de entrenamiento para la red neuronal.
- Los datos obtenidos a través de estos dispositivos podrán visualizarse y descargarse en una página web, donde estarán disponibles para el público en general.
- Se utilizará un modelo de red neuronal integrado en un dispositivo basado en Arduino.
- Se plantea la creación de un prototipo portátil capaz de estimar la radiación UV en tiempo real, facilitando su uso en distintas ubicaciones.

Con el fin de garantizar un desarrollo eficiente y modular del sistema, el proyecto se plantea desarrollar 4 prototipos para llegar al producto final. Los prototipos planteados son los siguientes:

- **Prototipo 1:** El objetivo de este prototipo es analizar de diferentes configuraciones de divisores de voltaje para determinar cuál representa mejor la variación de la luz visible y se adapta mejor al sistema.
- **Prototipo 2:** El objetivo de este prototipo es crear de un conjunto de datos de radiación UV y luz visible para el entrenamiento de la red neuronal, debido a la falta de bases de datos existentes con estos parámetros.
- **Prototipo 3:** El objetivo de este prototipo es el desarrollo de una red neuronal capaz de estimar la radiación UV utilizando únicamente la fotorresistencia y la hora del día, entrenada con el conjunto de datos obtenido en el prototipo anterior.
- **Prototipo 4:** El objetivo de este prototipo es la implementación de la red neuronal en un dispositivo portátil basado en Arduino para realizar estimaciones en tiempo real.

En las siguientes secciones se describe la aplicación de la metodología para el desarrollo de los prototipos anteriormente mencionados.

Prototipo 1

Análisis de requerimientos

Como se mencionó anteriormente, el objetivo del primer prototipo es realizar lecturas de la intensidad luminosa mediante el uso de fotorresistencias (LDR). A partir de este objetivo, se han definido los siguientes requerimientos:

- El sistema realizará lecturas de la intensidad de luz mediante fotorresistencias (LDR) utilizando un divisor de voltaje.

- Para determinar la mejor configuración se van a comprar 3 posibles configuraciones de divisores de voltajes (300Ω, 1kΩ y 10kΩ). Con esta información se identificará cuál de las configuraciones de divisor de voltaje es la más adecuada para utilizar las mediciones de intensidad como entrada de la red neuronal artificial.

- El sistema incluirá la medición de temperatura y humedad a través de un sensor DHT11.

- Durante los tres primeros minutos de la hora, el sistema realizará mediciones cada segundo. La medición obtenida será el promedio de las 180 mediciones realizadas durante este periodo.

- Se utilizará una memoria SD para almacenar los datos en un archivo CSV.

- Durante los 57 minutos posteriores el sistema estará inactivo y se repetirá el ciclo cuando inicie la hora.

Diseño

Diagrama a bloques.

En la Fig. 11 se muestra el diagrama a bloques del prototipo. Como se puede observar, el prototipo tiene las siguientes entradas de datos:

- Sensor DHT11: El cual medirá la humedad y temperatura y estas mediciones serán enviadas en forma de señal digital a la placa Arduino.

- Divisores de voltaje: Se implementan tres divisores de voltaje con resistencias de 330Ω, 1KΩ y 10KΩ y fotorresistencias (LDR), donde generaran un voltaje diferente de acuerdo con la intensidad luminosa y este voltaje se enviará a la placa Arduino.

Los resultados de las mediciones realizadas por los sensores descritos anteriormente serán almacenados en la tarjeta de microSD, en formato CSV. Este archivo servirá de base para el procesamiento de la información digital.

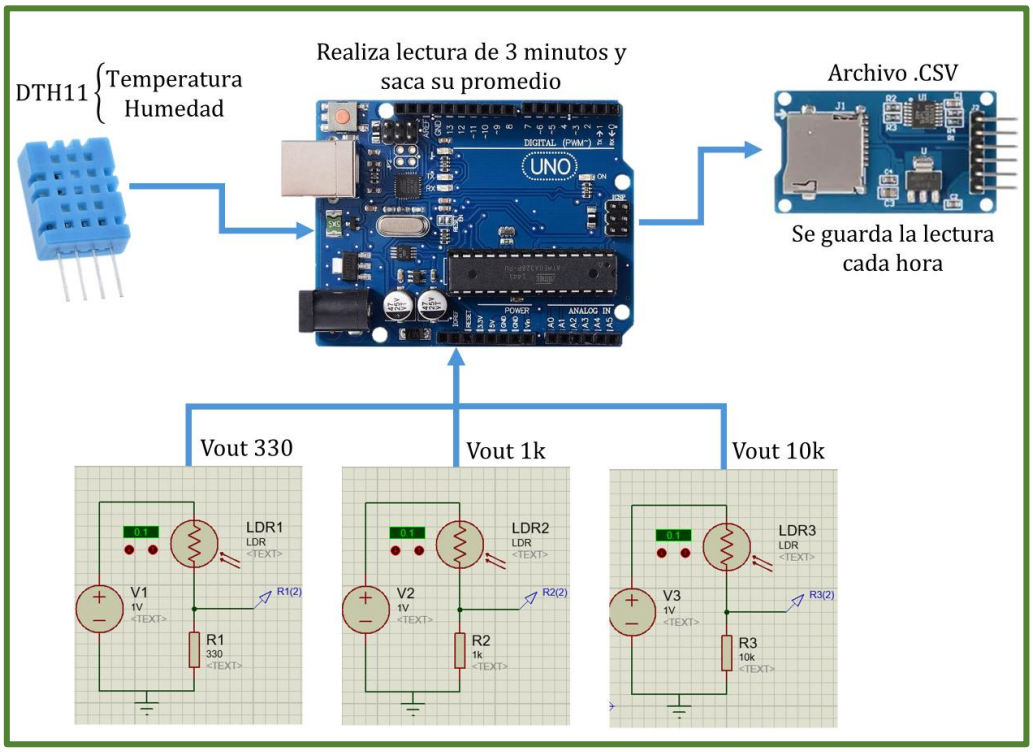


Fig. 11.- Diagrama a bloques del prototipo 1.

Diagrama eléctrico.

El diagrama eléctrico (Fig. 12) muestra las conexiones de los componentes principales del sistema: el sensor DHT11, los divisores de voltaje con sus resistencias específicas y el lector de tarjetas SD, todos conectados a la placa Arduino, operando con una fuente de 5V y compartiendo una conexión a tierra (GND).

Conexión del Sensor DHT11

- El sensor DHT11 envía los datos en una señal digital al pin D2 del Arduino.

Conexión de los Divisores de Voltaje

- La caída de voltaje en cada divisor se envía en señales analógicas a los pines A0, A1 y A2 del Arduino.

Conexión del Lector de Tarjetas SD

El lector de tarjetas SD permite la comunicación entre la microSD y el Arduino, a través de las siguientes conexiones:

- Pin D10 al SCK → Encargado de la transmisión de datos entre la microSD y el Arduino.
- Pin D11 al MISO → Transfiere los datos desde la microSD hacia el Arduino.
- Pin D12 al MOSI → Transfiere los datos desde el Arduino hacia la microSD.
- Pin D13 al CS → Habilita el lector de tarjetas SD.

Esto nos permite que el Arduino pueda leer el archivo o cree el archivo en la MicroSD donde se guardara el promedio de las lecturas cada hora durante más de un mes.

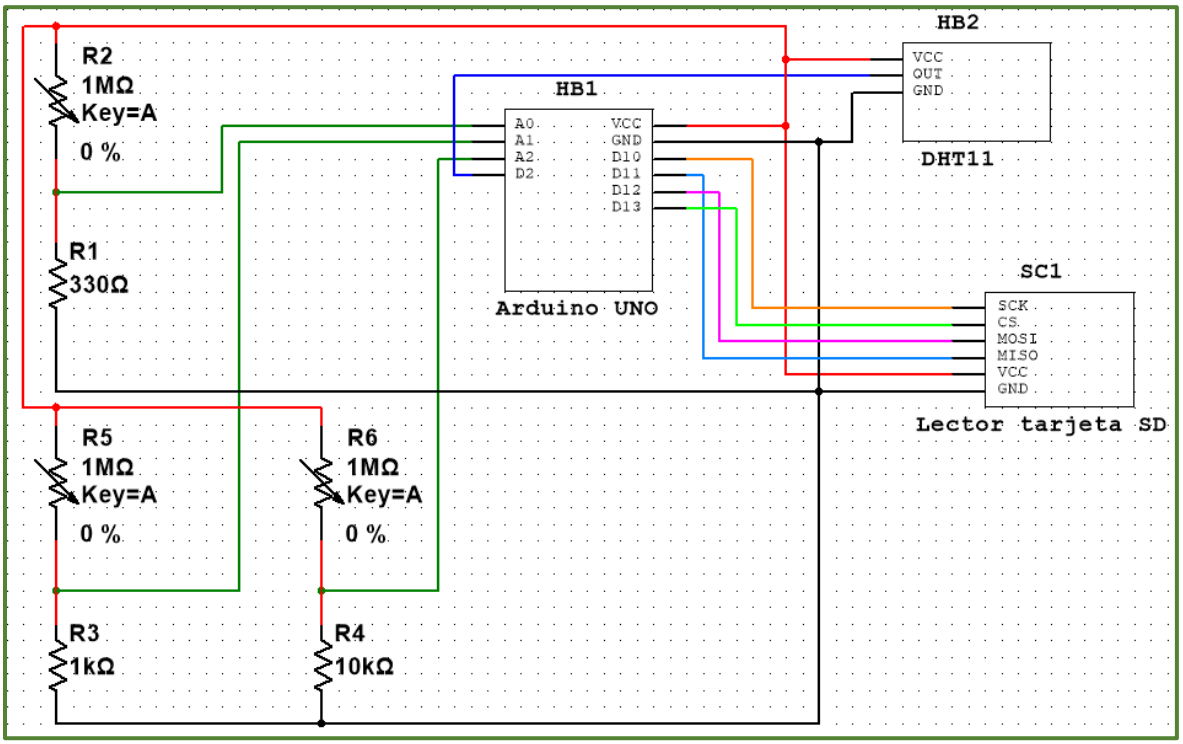


Fig. 12.- Diagrama eléctrico del prototipo 1.

Diagrama de flujo.

Como se puede observar en el diagrama de flujo (Fig. 13) el programa inicia con la declaración de variables, la configuración de los pines y la inicialización del lector de tarjetas SD. A continuación, comienza el proceso de medición, que se ejecuta cada segundo durante un período de 3 minutos (180 mediciones).

En cada iteración del bucle, se realizan los siguientes procesos:

- Medición del Voltaje de salida de los tres divisores de voltaje.
- Medición de la Temperatura y humedad.
- Se calcula una autosuma de cada una de estas mediciones.

Una vez finalizado el proceso de medición, si la tarjeta SD está correctamente inicializada, el sistema buscará el archivo correspondiente; si no existe, lo creará.

Cuando el archivo está disponible, se calculará el promedio de las lecturas dividendo el resulta de la autosuma de cada medición entre 180. Este valor se almacenará en un archivo CSV.

Finalmente, el sistema entra en un período de espera de 57 minutos antes de reiniciar el ciclo.

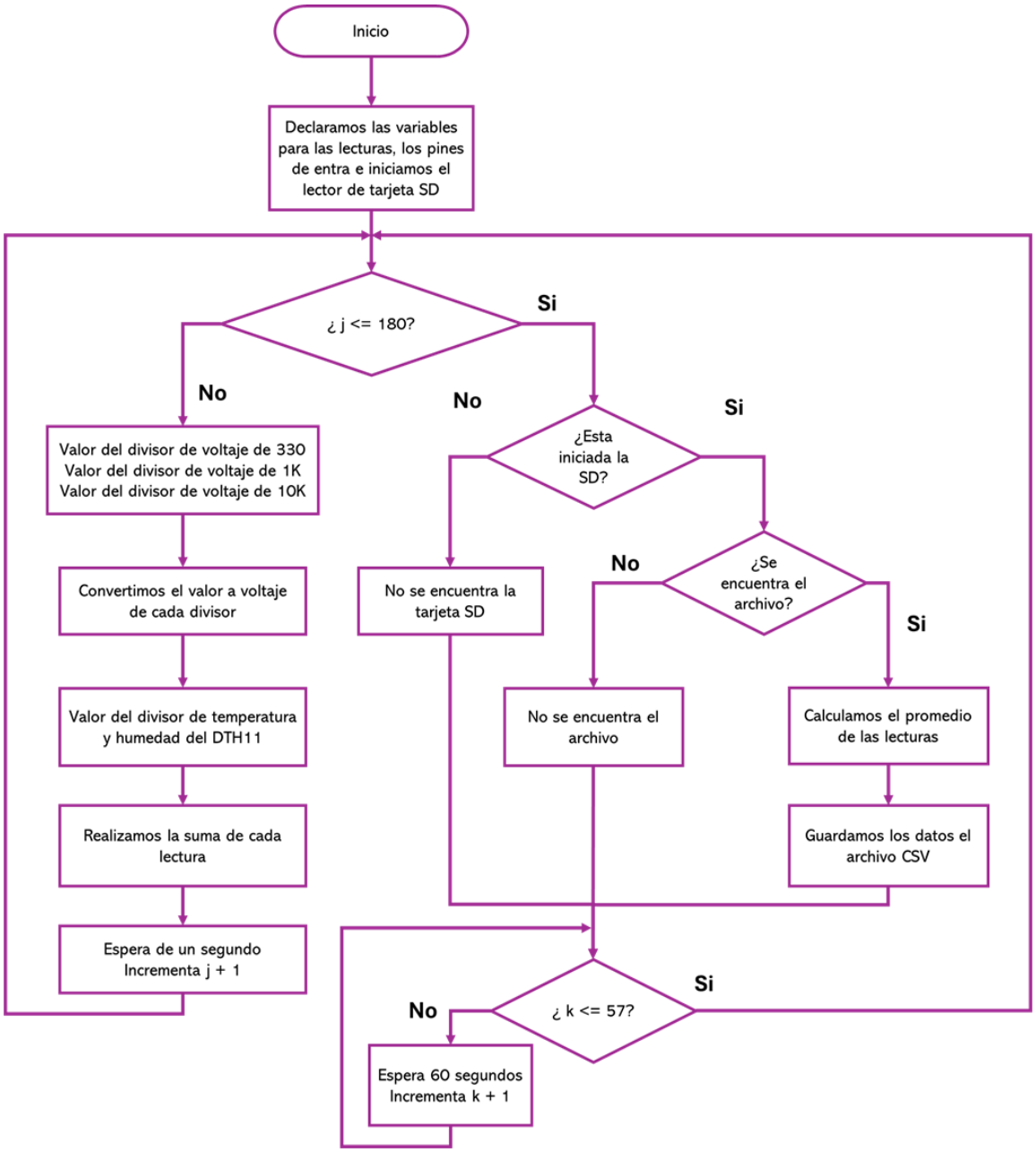


Fig. 13.- Diagrama de flujo del prototipo 1.

Implementación

El siguiente paso es la codificación del diagrama de flujo y la implementación del prototipo en físico, tomando en cuenta el diagrama eléctrico descrito anteriormente. Para esto, utilizamos Arduino IDE y, como primer paso, es necesario cargar la librería del sensor DHT11. Para ello, en el gestor de bibliotecas buscamos "DHT sensor library", la instalamos y posteriormente iniciamos la codificación.

En el **Anexo B**, se presenta un ejemplo detallado del proceso de importación de librerías en Arduino, incluyendo los pasos para instalar y utilizar correctamente la librería del sensor DHT11.

Inicialización del Código

Definimos los pines de conexión del sensor DHT11 y del lector de tarjetas SD, declaramos una variable tipo File para el almacenamiento de datos y creamos una instancia del sensor DHT (Fig. 14).

```
1   #include <SD.h>
2   #include <SPI.h>
3   #include <DHT.h>
4   #include <DHT_U.h>
6   #define sensor 7
7   #define SSpin 10
8
9   File archivo;
10  DHT dht(sensor, DHT11);
12  int i = 1;
13  float pv1 = 0, pv2 = 0, pv3 = 0;
14  float prt = 0, prh = 0;
```

Fig. 14.- Definición de variable y librerías.

Lectura y Procesamiento de Datos en loop()

Durante cada iteración del loop(), se realizan las lecturas de los valores de los divisores de voltaje. Se establece una condición para que, si la resistencia es mayor a 10MΩ, el valor se mantenga en ese límite. Luego, los valores analógicos obtenidos se convierten a voltaje. También se mide la temperatura y la humedad, y se realiza una autosuma de

los valores registrados. Este proceso se repite cada segundo, completando un ciclo de 180 iteraciones. Ver Fig. 15.

```
32    for (int j = 1; j <= 180; j++) {
33      float valor = analogRead(A0);
34      float valor2 = analogRead(A1);
35      float valor3 = analogRead(A2);
36
37      float V = valor * (5.0 / 1023);
38      float o = (V * 330) / (5.0 - V);
39      float V2 = valor2 * (5.0 / 1023);
40      float o2 = (V2 * 1000) / (5.0 - V2);
41      float V3 = valor3 * (5.0 / 1023);
42      float o3 = (V3 * 10000) / (5.0 - V3);
43
44      if (o > 10000000) {
45        o = 10000000;
46      }
47
48      if (o2 > 10000000) {
49        o2 = 10000000;
50      }
51
52      if (o3 > 10000000) {
53        o3 = 10000000;
54      }
55
56      float Vout = (5.0 * o) / (o + 330);
57      float Vout2 = (5.0 * o2) / (o2 + 1000);
58      float Vout3 = (5.0 * o3) / (o3 + 10000);
59
60      float tmp = dht.readTemperature();
61      float hum = dht.readHumidity();
62
63      pv1 = pv1 + Vout;
64      pv2 = pv2 + Vout2;
65      pv3 = pv3 + Vout3;
66      prt = prt + tmp;
67      prh = prh + hum;
68
69      delay(1000);
70    }
```

Fig. 15.- Ciclo for de la toma de lecturas.

Almacenamiento de datos en la Tarjeta SD

Se verifica la presencia del lector de tarjetas SD y, si está disponible, se abre o crea el archivo CSV. Luego, se calcula el promedio de las lecturas y los valores resultantes se almacenan en el archivo. Finalmente, el sistema entra en un período de espera de 57 minutos antes de la siguiente captura de datos. Ver Fig. 16.

```
72  if (SD.begin(SSpin)) {
73    archivo = SD.open("PRUEBA2.csv", FILE_WRITE);
74
75    if (archivo) {
76
77      float rv1 = pv1 / 180;
78      float rv2 = pv2 / 180;
79      float rv3 = pv3 / 180;
80      float rvt = prt / 180;
81      float rvh = prh / 180;
82
83      float v1 = 5.0 - rv1;
84      float v2 = 5.0 - rv2;
85      float v3 = 5.0 - rv3;
86
87      archivo.print(i);
88      archivo.print(",");
89      archivo.print(v1, 4);
90      archivo.print(",");
91      archivo.print(v2, 4);
92      archivo.print(",");
93      archivo.print(v3, 4);
94      archivo.print(",");
95      archivo.print(rvt);
96      archivo.print(",");
97      archivo.println(rvh);
98
99      archivo.close();
```

Fig. 16.- Código para guardar las lecturas en archivo CSV.

Código Completo

El código completo de este prototipo se encuentra en el **Anexo A.1**.

Resultados

Los datos obtenidos fueron almacenados en un archivo CSV que incluye la siguiente información:

- Voltajes de los tres divisores (330 Ω, 1 kΩ y 10 kΩ).
- Temperatura.
- Humedad.
- Hora de cada registro.

En la Fig. 17 se muestra un fragmento de estos datos.

#Dato	Vout 330	Vout 1K	Vout 10K	Temperatura	Humedad	Hora
1	1.7859	3.5741	4.8455	20.14	33.27	08:00
2	1.9083	3.6937	4.8577	20.62	21.03	09:00
3	1.9859	3.7201	4.8646	19	21.59	10:00
4	2.2109	3.8649	4.8869	23.04	17.33	11:00
5	2.0155	3.8043	4.8735	24.67	15.28	12:00
6	2.3895	3.9865	4.898	24.5	15.49	13:00
7	2.8726	4.2235	4.9286	26.67	14.43	14:00

Fig. 17.- Datos obtenidos.

Para evaluar las mediciones se compararon los resultados con los datos obtenidos de cada divisor de voltaje con los datos de la página de monitoreo ambiental (www.indiceuv.com). Los resultados se muestran en la Fig. 18.

Como se puede observar, el divisor de 10 kΩ genera un voltaje que satura, prácticamente durante todo el tiempo de medición. Por otro lado, los divisores de 300 Ω y 1 kΩ producen valores con comportamientos cualitativos más parecidos a los que presenta el monitoreo de radiación UV de referencia.

Como conclusión de esta etapa, se determinó que en el siguiente prototipo solo se utilizarán los divisores conectados a 300 Ω. Además, se incorporarán componentes de medición más precisos con el objetivo de obtener un dataset más confiable.

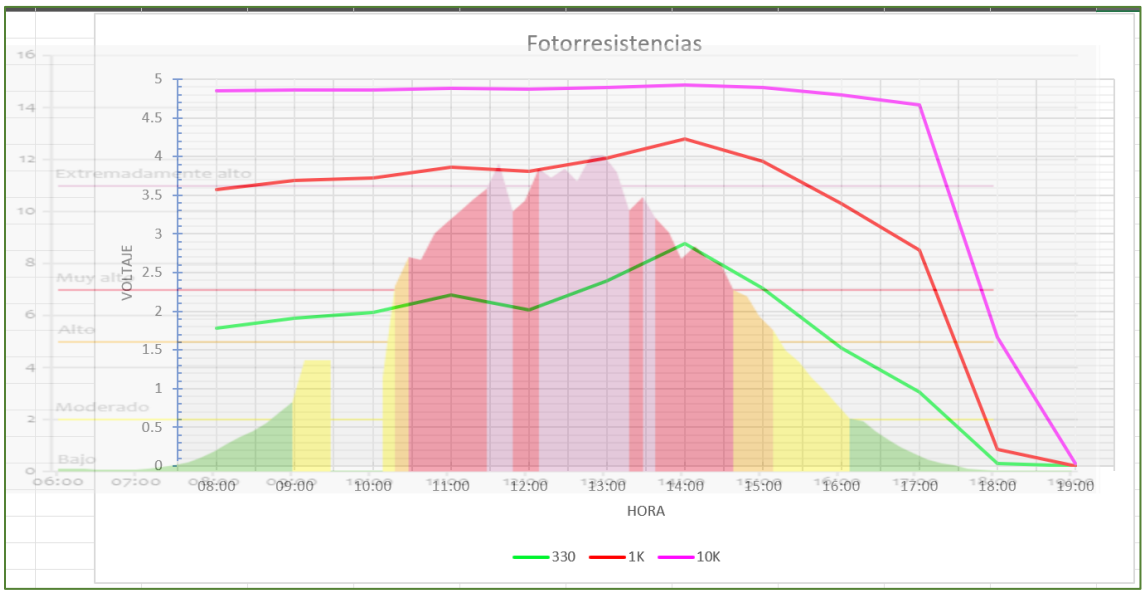


Fig. 18.- Resultados obtenidos con diferentes divisores de voltaje.

Prototipo 2

Análisis de requerimientos

Dado que no existen páginas que proporcionen datos específicos sobre la radiación ultravioleta (UV) y la radiación visible, los cuales son fundamentales para el entrenamiento de la red neuronal, el objetivo de este prototipo es construir un dataset robusto y bien estructurado que incluya mediciones de radiación UV, luz visible, humedad y temperatura, junto con marcas de tiempo, para su uso en el entrenamiento y validación de una red neuronal u otros modelos de inteligencia artificial.

Por lo cual se planteó los siguientes requerimientos:

Medición de Radiación Ultravioleta (UV)

- El prototipo debe contar con un sensor que permita medir la radiación ultravioleta en el entorno.
- La precisión de las mediciones debe ser suficiente para garantizar datos confiables para el entrenamiento de la red neuronal.

Medición de la Intensidad de Luz Visible

- Se implementará un sensor basado en un LDR (Light Dependent Resistor) para medir la intensidad de luz visible.
- El LDR estará conectado a un divisor de voltaje para mejorar la precisión de las mediciones.

- Se utilizará una resistencia de 330 ohms en el divisor de voltaje, ya que pruebas previas han demostrado que esta configuración proporciona los mejores resultados.

Medición de Temperatura y Humedad

- El prototipo incluirá un sensor de temperatura y humedad para complementar los datos ambientales recopilados.

Intervalos de Medición

- Las mediciones se realizarán en intervalos de 10 minutos.
- Este intervalo garantiza la captura de variaciones ambientales sin generar un volumen excesivo de datos.

Almacenamiento de Datos

- Los datos recolectados deben almacenarse en una tarjeta SD en formato CSV.
- Cada registro debe incluir la fecha y la hora de la medición para facilitar su procesamiento y análisis posterior.

Diseño

Diagrama a bloques.

En la Fig. 19 se muestra el diagrama a bloques del prototipo. Como se puede observar, el prototipo tiene las siguientes entradas de datos:

- Sensor ML8511: El cual medirá la radiación Ultravioleta y esta medición será enviada en forma de señal analógica a la placa Arduino.
- Divisor de voltaje: Se implementan un divisor de voltaje con una resistencia de 330Ω y una fotorresistencia (LDR), donde este voltaje se enviará a la placa Arduino.
- Sensor DHT11: El cual medirá la humedad y temperatura y estas mediciones serán enviadas en forma de señal digital a la placa Arduino.
- RTC DS3231: El cual enviara la fecha y hora en una señal a la placa Arduino.

Los resultados de las mediciones obtenidas por los sensores serán almacenados en una tarjeta microSD en formato CSV. Este archivo servirá como base para el posterior procesamiento y análisis de los datos registrados, así como para la creación del dataset utilizado en el entrenamiento de la red neuronal.

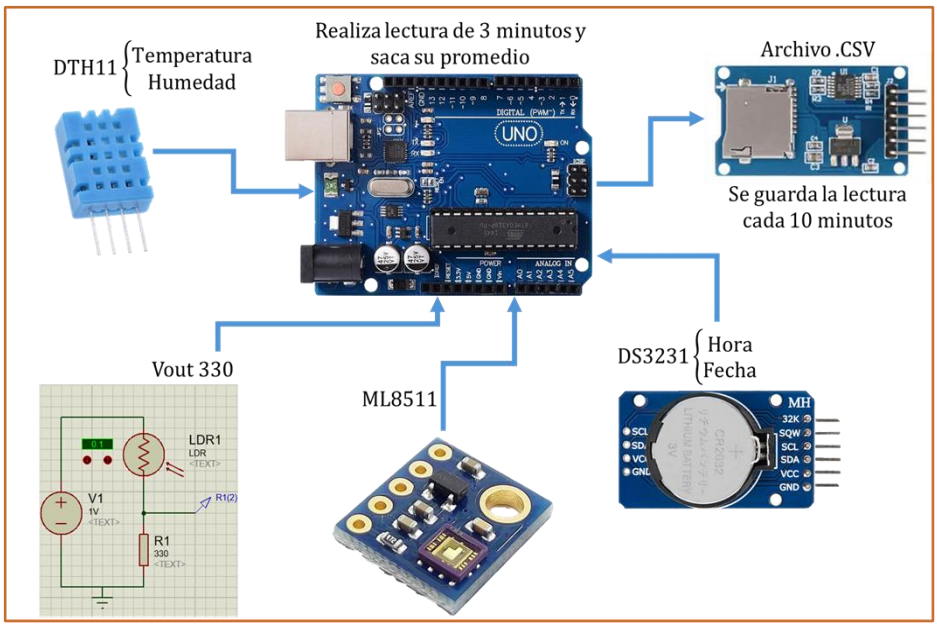


Fig. 19.- Diagrama a bloques del prototipo 2.

Diagrama eléctrico.

El diagrama eléctrico (Fig. 20) muestra las conexiones de los componentes principales del sistema: el sensor el sensor ML8511, el divisor de voltaje con su resistencia, DHT11, el RTC DS3231 y el lector de tarjetas SD, todos conectados a la placa Arduino, operando con una fuente de 5V y compartiendo una conexión a tierra (GND).

Conexión de los sensores

- El sensor DHT11 envía los datos en una señal digital al pin D2 del Arduino.
- La salida del divisor de voltaje se envía al pin analógico A0 del Arduino.
- El sensor ML8511 envía el dato en una señal analógica al pin analógico A2 del Arduino.
- El Arduino enviara voltaje de 3.3 al sensor ML8511, al pin EN y al pin analógico A3.

Conexión del DS3231

- El RTC DS3231 envía los datos en una señal a los pines SDA y SCL del Arduino.

Conexión del Lector de Tarjetas SD

- Se utilizará la misma configuración explicada en el prototipo 1.

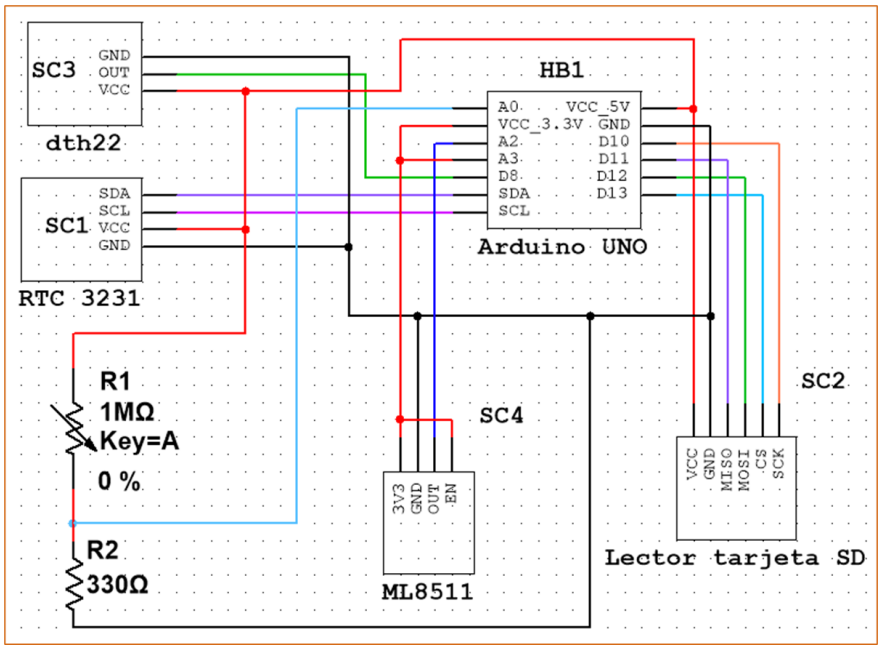


Fig. 20.- Diagrama eléctrico del prototipo 2.

Diagrama de flujo.

El programa considera realizar mediciones cada 10 minutos, pero el resultado de la medición será el promedio de 180 mediciones realizadas cada segundo durante los 3 minutos previos a los 10 minutos. Como se observa en la Fig. 21 para reportar la medición de las 12:30 la medición empezara a las 12:27 y realizara el proceso de las 180 lecturas y terminara a las 12:30 y así sucesivamente.

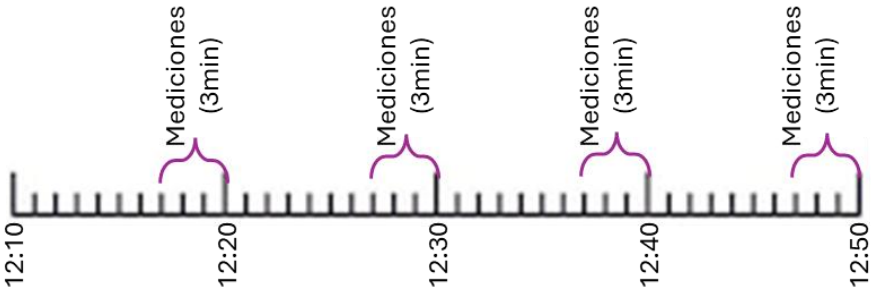


Fig. 21.- Rango de lecturas del prototipo 2.

Como se puede observar en el diagrama de flujo (Fig. 22) el programa inicia con la declaración de variables, la configuración de los pines y la inicialización del lector de tarjetas SD y los sensores. A continuación, comienza con el proceso de la verificación de la tarjeta SD, si esta esta iniciada verifica si el minuto termina con el digito 7, en caso contrario regresa verificar.

A continuación, comienza el proceso de medición, que se ejecuta cada segundo durante un período de 3 minutos (180 mediciones). En cada iteración del bucle, se realizan los siguientes procesos:

- Medición del Voltaje de salida del divisor de voltaje.
- Medición de la Temperatura y humedad.
- Medición del sensor UV.
- Se calcula una autosuma de cada una de estas mediciones.

Una vez finalizado el proceso de medición, realizará el promedio de las 180 lecturas y volverá a verificar si el digito es 7. Si el digito no es 7 pero si se realizaron mediciones buscara el archivo. Si está el archivo se guardarán los datos en formato CSV. En caso contario creará el archivo y se repetirá el ciclo desde la inicialización del lector de tarjeta SD.

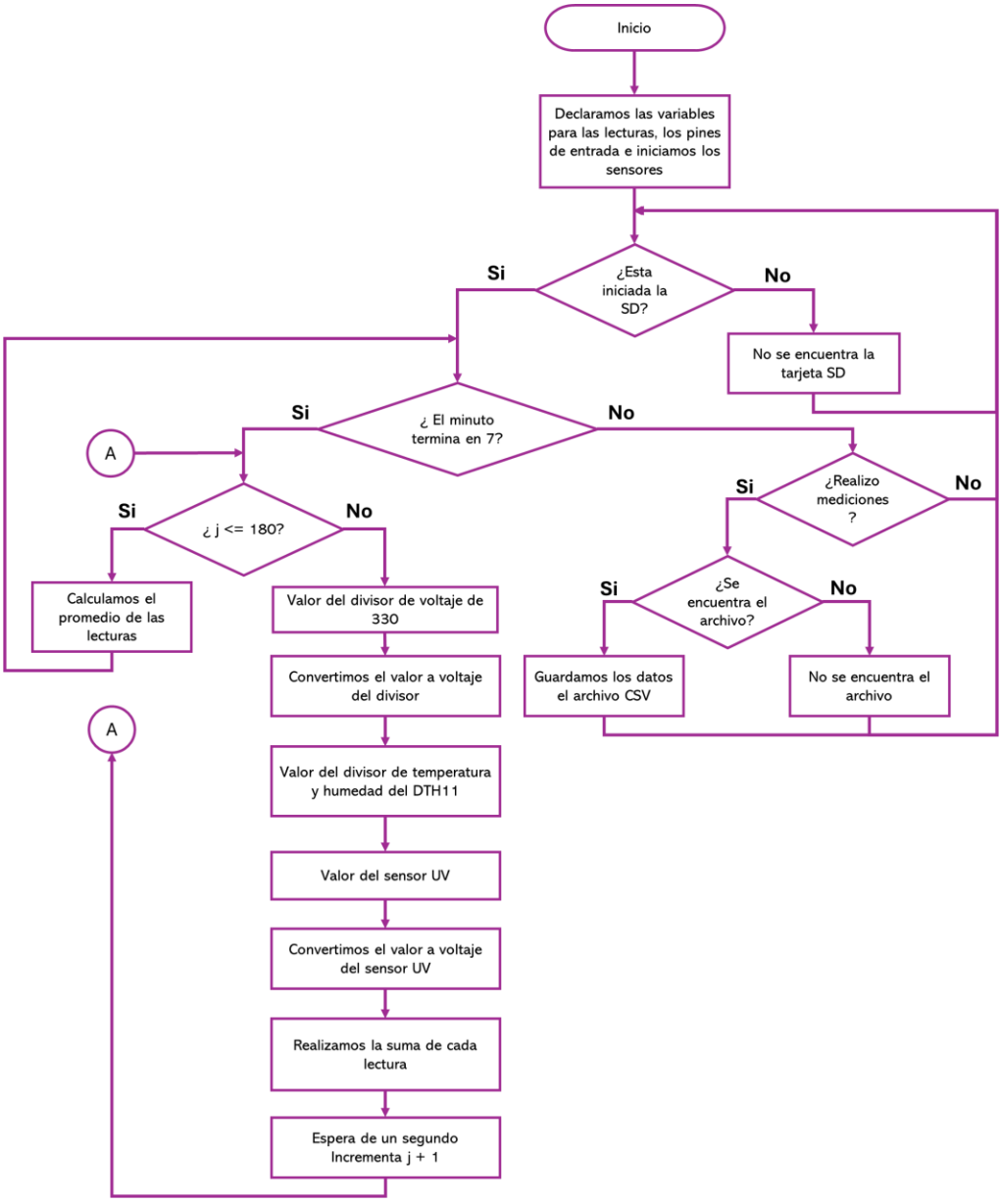


Fig. 22.- Diagrama de flujo del prototipo 2.

Implementación

El siguiente paso es la codificación del diagrama de flujo y la implementación del prototipo en físico, tomando en cuenta el diagrama eléctrico descrito anteriormente. Para esto, utilizamos Arduino IDE y, como primer paso, es necesario cargar la librería del sensor DHT22 y del RTC.

En el **Anexo B**, se presenta un ejemplo detallado del proceso de importación de librerías en Arduino, incluyendo los pasos para instalar y utilizar correctamente la librería del sensor DHT22 y del RTC.

Inicialización del Código

Definimos los pines de conexión del sensor DHT22, el lector de tarjetas SD y del sensor UV, Declaramos las variables necesarias para los sensores y donde se guardarán las lecturas. Ver Fig. 23.

```
#include <SD.h>
#include <SPI.h>
#include <DHT.h>
#include <DHT_U.h>
#include <Wire.h>
#include <RTClib.h>

#define sensor 8
#define SSpin 10

File archivo;
RTC_DS3231 rtc;
DHT dht(sensor, DHT22);

char bufferDatoFecha[19];
char bufferDatoHora[19];
int UVOUT = A2;
int REF_3V3 = A3;
int i = 0;
float pv1 = 0, pv2 = 0, pv3 = 0, puv = 0;
float prt = 0, prh = 0, pvuv = 0;
float rv1 = 0, rv2 = 0, rvt = 0, rvh = 0, ruv = 0, rvuv = 0;
```

Fig. 23.- Declaración de librerías y variables.

Lectura y Procesamiento de datos en loop()

Durante cada iteración del loop(), se realizan una verificación del minuto. Cuando este termine en 7, 17, etc. Realiza la lectura del divisor de voltaje. Se establece una condición para que, si la resistencia es mayor a 10MΩ, el valor se mantenga en ese

límite. Luego, los valores analógicos obtenidos se convierten a voltaje. También se mide la temperatura y la humedad, se realiza la lectura del sensor UV y se realizar una conversión de esa lectura dentro de un rango y al final se hace una autosuma de los valores registrados. Este proceso se repite cada segundo, completando un ciclo de 180 iteraciones. Ver Fig. 24.

```
//if para programar fecha
if (fecha.minute() == 7 || fecha.minute() == 17 || fecha.minute() == 27

  limpiar();

 for (int j = 1; j <= 180; j++) {  //180
   float valor = analogRead(A0);

   float V = valor * (5.0 / 1023);
   float o = (V * 330) / (5.0 - V);

   if (o > 10000000) {
     o = 10000000;
   }

   float Vout = (5.0 * o) / (o + 330);

   float tmp = dht.readTemperature();
   float hum = dht.readHumidity();

   int uvLevel = averageAnalogRead(UVOUT);
   int refLevel = averageAnalogRead(REF_3V3);
   float outputVoltage = 3.33 / refLevel * uvLevel;

   float uvIntensity = mapfloat(outputVoltage, 0.98, 2.9, 0.0, 15.0);

   puv = puv + uvIntensity;
   pv1 = pv1 + Vout;
   prt = prt + tmp;
   prh = prh + hum;

   delay(1000);
 }
```

Fig. 24.- Ciclo for par tomar lecturas.

Almacenamiento de Datos en la Tarjeta SD

Se abre o se crea el archivo CSV y los valores promediados se almacenan en el archivo. Finalmente, el sistema entra vuelve a verificar que el minuto sea 7, 17, etc. para la siguiente captura de datos. Ver Fig. 25.

```
if (archivo) {

  sprintf(bufferDatoFecha, "%02d/%02d/%04d", fecha.day(), fecha.month(), fecha.year());
  sprintf(bufferDatoHora, "%02d:%02d", fecha.hour(), fecha.minute());
  archivo.print(bufferDatoFecha);
  archivo.print(",");
  archivo.print(bufferDatoHora);
  archivo.print(",");
  archivo.print(round(ruv));
  archivo.print(",");
  archivo.print(ruv, 2);
  archivo.print(",");
  archivo.print(v1, 4);
  archivo.print(",");
  archivo.print(v2, 4);
  archivo.print(",");
  archivo.print(rvt);
  archivo.print(",");
  archivo.print(rvh);
  archivo.print(",");
  archivo.println(rvuv);

  archivo.close();


  limpiar();

  i = 0;

} else {
  Serial.println("no se encuentra el archivo");
}
```

Fig. 25.- Código para guardar las lecturas en archivo CSV.

Código Completo

El código completo de este prototipo se encuentra en el **Anexo A.2**.

Resultados

Los datos obtenidos fueron almacenados en un archivo CSV que incluye la siguiente información:

- Fecha y hora de la lectura.

- Índice UV entero y el Índice UV con decimales
- El voltaje del divisor de 330Ω
- La temperatura.
- La humedad.

En la Fig. 26 se muestra un fragmento de estos datos.

Fecha	Hora	Indice UV int	Indice UV flo	Vout 330	Temperatura	Humedad
13/01/2024	13:40	2	2.24	2.714	26.48	26.78
13/01/2024	13:50	2	1.97	2.6213	26.2	27.76
13/01/2024	14:00	2	1.69	2.4829	24.33	30.32
13/01/2024	14:10	2	1.53	2.376	23.53	31.49
13/01/2024	14:20	1	1.27	2.2269	23.96	31.12
13/01/2024	14:30	1	1.28	2.2298	23.81	31.24
13/01/2024	14:40	1	1.27	2.2244	23.31	31.86
13/01/2024	14:50	1	1.42	2.3428	24.85	30.26
13/01/2024	15:00	2	1.63	2.4952	25.29	29.92
13/01/2024	15:10	2	1.77	2.5611	26.47	28.69

Fig. 26.- Fragmento de las lecturas del prototipo 2.

Modelado del índice UV en función de la medición de radiación visible

En la Fig. 27 se presenta la gráfica de la medición del nivel de radiación UV respecto a la intensidad de luz visible medida el día 3 de marzo de 2024. El segmento de la curva con línea azul representa las mediciones de la radiación UV obtenidos en el movimiento aparente del sol desde el amanecer hasta que el sol alcanza su zenit (aproximadamente a las 13:00 hrs). Las mediciones realizadas desde el zenit hasta la puesta del sol se representan con el segmento violeta.

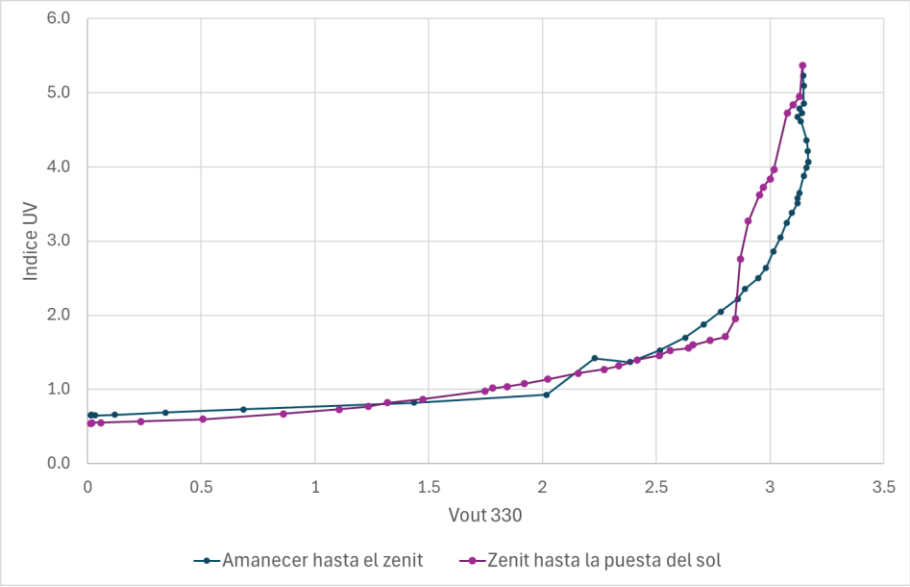


Fig. 27.- Grafica del índice UV con respecto al voltaje del divisor del día 3 de marzo.

Con la intención de entender mejor el comportamiento de las mediciones, inicialmente planteó encontrar un modelo matemático que describa el comportamiento de la radiación UV en función del voltaje medido correspondiente a la intensidad de luz visible (la salida del divisor de voltaje formado por la fotorresistencia y la resistencia de 330 ohms). Como se observa en la Fig. 28, en ambas curvas se presentan dos regiones con los siguientes comportamientos.

- Región de comportamiento lineal. En el rango de voltajes que va de $0 \leq V_{out330} \leq 2.5$ V el índice de radiación UV presenta un comportamiento aproximadamente lineal (Línea verde de la Fig. 28).

- Región de comportamiento exponencial. En el rango de voltajes que va de $V_{out330} > 2.5$V, el índice de radiación UV es una función aproximadamente exponencial (Línea roja de la Fig. 28).

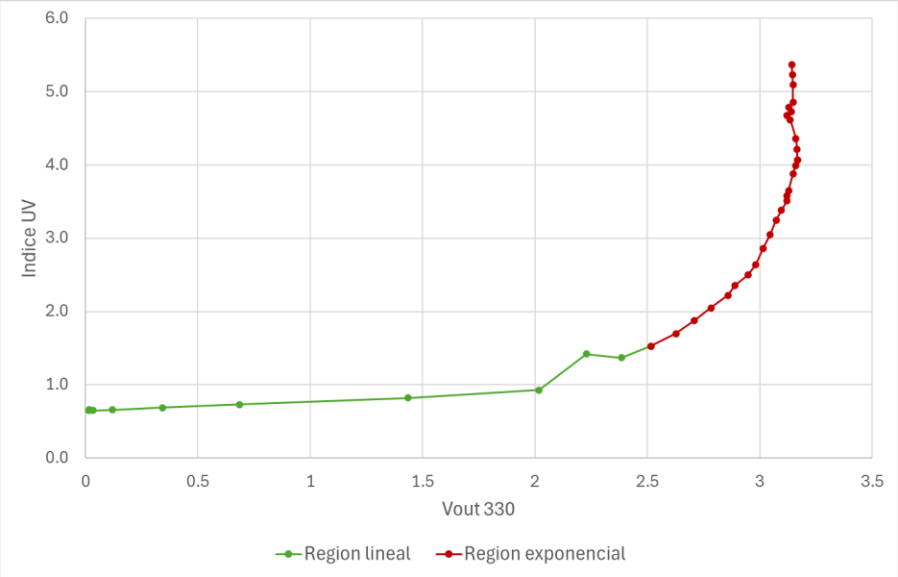


Fig. 28.- Grafica del comportamiento de las mediciones.

Tomando en cuenta ambas regiones se definió la función que corresponde a la radiación UV respecto al voltaje de salida del divisor de voltaje, correspondiente a la intensidad de luz visible (Ecu. 1). En esta ecuación la región lineal está representada por los términos $ax + b$, mientras que la componente exponencial de la función se incluye mediante el término ce^{dx}.

(1)

$$y = ax + b + ce^{dx}$$

En la tabla 1 se presentan los parámetros de la función extraídos a partir de las mediciones experimentales realizadas en los periodos comprendidos del amanecer al

Zenit y del Zenit a la puesta del sol. En el **Anexo A.3** se describe el procedimiento de extracción de estos parámetros.

Tabla 1.- Parámetros de la función de radiación UV extraídos de los datos experimentales.

Antes del zenit				
Punto	x	y		
1	0.0329	0.65		
2	2.0185	0.93		
3	3.1478	5.1		
4	2.7826	2.05		
	a	b	c	d
Valor	0.14	0.65	2.81E-05	3.77

Despues del zenit				
Punto	x	y		
1	0.0586	0.55		
2	1.474	0.87		
3	3.1425	5.37		
4	2.8696	2.76		
	a	b	c	d
Valor	0.23	0.54	6.33E-05	3.53

La comparación entre las curvas experimentales y las curvas modeladas utilizando la Ecu. 1 se presenta en la Fig. 29.

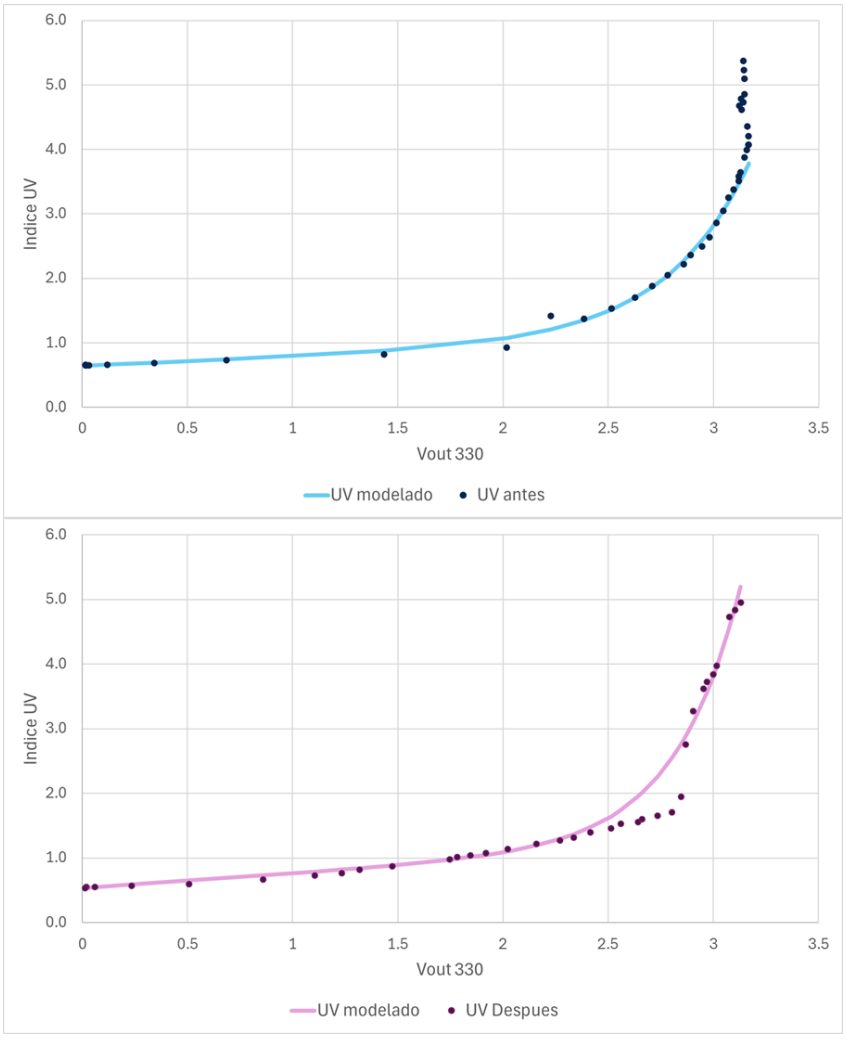


Fig. 29.- (a) Comparación antes del zenit. (b) Comparación después del zenit.

En la Fig. 29 se muestra la comparación entre la curva generada por el modelo matemático y la curva obtenida a partir de los datos del sensor ML8511.

- En la Fig. 29a se observa un buen ajuste entre ambas curvas, excepto en los últimos datos, donde se forma una línea vertical debido a que los valores se repiten y permanecen constantes. El error promedio en esta curva es de 6.5%.

- En la Fig. 29b, nuevamente se aprecia un buen ajuste general, salvo en la región comprendida entre los 2.5 V y 2.8 V, donde se presenta una mayor discrepancia. El error promedio de esta curva es de 6.8%.

Conclusiones

El modelo matemático utilizado presenta un error promedio del 6.65%, lo cual indica un desempeño aceptable para estimar el índice UV. En la Fig. 30 se muestran los resultados obtenidos al aplicar el modelo matemático completo. Se puede observar que el error es mínimo en la mayoría de los casos, excepto en algunas regiones específicas.

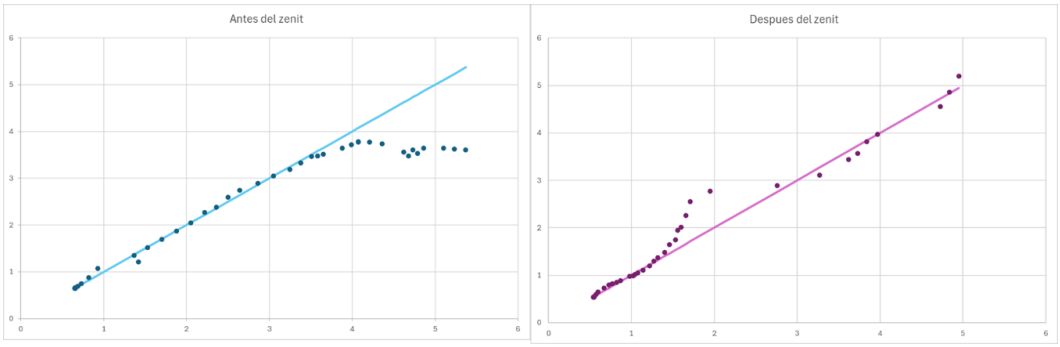


Fig. 30.- (a) Resultados del modelo antes del zenit. (b)Resultados del modelo después del zenit.

- En la Fig. 30a, los errores observados pueden atribuirse a una región de saturación de datos. Este comportamiento también se detectó durante las pruebas del Prototipo 1, donde el uso de resistencias de 1kΩ provocó una saturación (mayor a la que se observa en la resistencia 330) lineal cerca del pico máximo de radiación solar durante el día.
- En la Fig. 30b, el error podría deberse a un fenómeno meteorológico ocurrido en ese momento, lo que afectó la precisión de los datos capturados. A la linealidad de los datos.

Como siguiente paso, se desarrollará una red neuronal artificial para estimar el índice UV a partir de la salida del divisor de voltaje utilizando una resistencia de 330 ohms. Posteriormente, se realizará una comparación entre los resultados obtenidos con el modelo matemático y la red neuronal para determinar cuál ofrece una mejor precisión en la estimación del índice UV.

Prototipo 3

Análisis de requerimientos

En este prototipo se tiene como finalidad comprar los resultados del índice de radiación ultravioleta (UV) proporcionados mediante una RNA con los resultados obtenidos a través de un modelo matemático. Esta comparación permitirá evaluar el desempeño del modelo basado en una RNA, determinado su precisión para estimar el índice UV.

El objetivo principal del sistema es desarrollar una red neuronal capaz de estimar el índice UV utilizando como parámetros de entrada el valor del divisor de voltaje, generado por una fotorresistencia, y la hora del día. A partir de este objetivo, se establecieron los siguientes requerimientos funcionales y técnicos:

- La red neuronal deberá recibir como entradas:
    - El valor del divisor de voltaje generado por la fotorresistencia.
    - La hora del día en formato adecuado para su procesamiento.
- La salida del modelo será el índice UV estimado.
- Se implementará una red neuronal del tipo Perceptrón Multicapa (Multilayer Perceptron, MLP).
- El desempeño de la red neuronal se evaluará utilizando el Error Cuadrático Medio (MSE, por sus siglas en inglés) como métrica principal.
- El entrenamiento inicial se llevará a cabo con un subconjunto de datos recolectado el día 3 de marzo de 2024. Posteriormente, se extenderá el entrenamiento al conjunto completo de datos disponibles.
- La red neuronal deberá tener la capacidad de realizar predicciones en tiempo real con datos que no hayan sido incluidos en el conjunto de entrenamiento, con el fin de validar su capacidad de generalización y adaptación a nuevas condiciones.

Diseño del prototipo

En este prototipo la RNA será entrenada utilizando el dataset obtenido en el prototipo 2. Este dataset fue desarrollado aplicando a metodología RAD (Rapid Application Development), tal como se describe en la sección previa. La base de datos esta almacenada en un archivo CSV y consta de un conjunto de datos formado por la hora de medición, la intensidad de radiación UV medida con un sensor ML8511 y el voltaje de salida del divisor de voltaje que es una función de la intensidad de luz visible. Este

conjunto de datos fue obtenido mediante mediciones realizadas cada 10 minutos durante el día 03 de marzo del 2024.

La RNA implementada corresponde a un Perceptrón Multicapa (MLP), cuya arquitectura se detalla a continuación (ver Fig. 31):

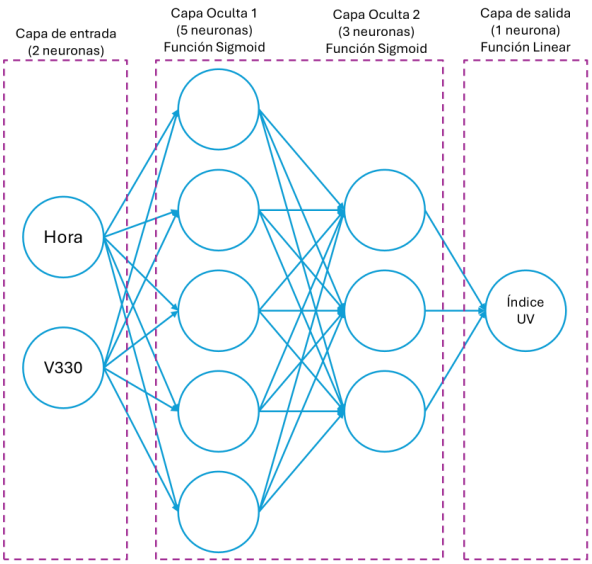
- La cada de entrada con 2 neuronas (Hora y Vout330 del divisor de voltaje).
- La primera capa oculta con 5 neuronas.
- La segunda capa oculta con 3 neuronas.
- La capa de salida con una neurona el cual indicara el índice UV.

Funciones de activación

- Las capas ocultas utilizan la función de activación sigmoide.
- La capa de salida emplea una función de activación lineal, adecuada para problemas de regresión.

Parámetros de entrenamiento

- Optimizador: Se utilizó el algoritmo Adam, con una tasa de aprendizaje de 0.1.
- Épocas de entrenamiento: La red fue entrenada durante 300 épocas.
- Métrica de evaluación: Se utilizó el Error Cuadrático Medio (MSE) para medir el rendimiento del modelo.



Hiperparámetro	Descripción	Valor en el modelo
Estructura de la red	Número de neuronas en cada capa de la red (capa de entrada, ocultas, salida).	2 (entrada) 5 - 3 (Oculta) 1 (salida)
Función de activación	La función utilizada en cada capa para introducir no linealidad.	Sigmoid (capas ocultas), Lineal (capa de salida)
Optimización	Algoritmo utilizado para ajustar los pesos de la red.	Adam
Tasa de aprendizaje	La velocidad a la que el modelo ajusta sus pesos durante el entrenamiento.	0.1
Función de pérdida	Métrica utilizada para calcular el error del modelo.	Error cuadrático medio (MSE)
Número de épocas	El número de veces que todo el conjunto de datos de entrenamiento pasa por la red.	300

Fig. 31.- Diseño de la red neuronal y los hiperparámetros utilizados

Construcción rápida

Preparación de los datos

Como se mencionó en secciones previas, el proceso de entrenamiento de la RNA requiere de un conjunto de datos de entrada y un conjunto de datos de salida, llamado función objetivo. En este prototipo, el conjunto de datos se encuentra almacenados en un archivo CSV el cual contiene lo siguiente:

- La hora en que se realizó la medición,
- El voltaje del divisor de voltaje, el cual es una función de la intensidad luminosa. Para esta medición se utilizó en divisor de voltaje formado por la conexión del LDR y la resistencia de 330 ohms (Vout330).
- El Índice UV registrado por el sensor ML8511 con decimales (Índice UV float).

Para este prototipo se utilizaron los datos recolectados el día 3 de marzo de 2024, el cual está compuesto por 81 mediciones.

Los datos del archivo CVS son cargados al entorno de programación de Google colab y fueron almacenados en una la variable denominada dataset, leemos el archivo y guardaremos todo su contenido para realizar, de forma automática, la estandarización de estos. Ver Fig. 32.

```
dataset = pd.read_csv('datos.csv')
```

Fig. 32.- Lectura y almacenamiento del archivo CSV.

En la Fig. 33 se presentan los datos de entrenamiento en el formato original:

- Hora de realización de la medición. En formato horas y minutos (HH:mm).
- Salida del divisor de voltaje (Vout330). Este voltaje está relacionado con el nivel de radiación de luz visible y es medido a partir de la fotorresistencia. El Vout330 se encuentra expresado en 4 decimales.
- Índice UV. Este valor es medido a través del sensor ML8511 y se expresa en formato de 2 decimales.

La RNA será entrenada con estos datos donde las entradas serán la hora de la medición y el Vout330; mientras que el índice UV será la función objetivo. Con el objetivo de optimizar el proceso de entrenamiento de la RNA es necesario realizar una normalización de los datos de entrenamiento. En la siguiente sección se explicará a más detalle el proceso.

	Hora	Vout330	Indice UV float
0	05:50	0.0147	0.56
1	06:00	0.0152	0.65
2	06:10	0.0158	0.66
3	06:20	0.0180	0.65
4	06:30	0.0329	0.65

Fig. 33.- Datos de entrenamiento originales.

Normalización de los datos

La normalización es un preprocesamiento que ajusta los datos de entrada y salida a una escala para que evitar que los datos tengan una mayor magnitud sobre otros. Con esta normalización nos aseguramos de que los datos estén dentro de un rango y evitar que un mal entramiento de la RNA.

Los principales beneficios de la normalización son:

- Evitar que los valores grandes dominen el entrenamiento: Al estar los datos en diferentes escalas, las variables con valores más altos podrían influir más en el cálculo de los pesos, generando un desbalance.
- Mejorar la convergencia: Facilita que el entrenamiento de la RNA sea más estable y rápido, reduciendo el número de épocas necesarias.
- Mejorar la precisión del modelo: Al trabajar en la misma escala, se reduce el sobreajuste y mejora la generalización.

Para la normalización estos datos se utilizó Z-score, esto a que los datos se encuentran en escalas diferentes se tiene que buscar su media y desviación entandar para poder tener los datos dentro de una misma escala donde sus valores comprendidos estén entre los -3 y 3.

Como primer paso en el proceso de normalización es procesar los datos correspondientes a la hora de medición. En esta se convertirá el dato expresado en horas y minutos a su equivalente expresado en minutos (ver ecu. 2).

$$Hora\ en\ minutos = (horas * 60) + minutos \tag{2}$$

En la Fig. 34 se observa el código utilizado para expresar la hora de medición en minutos.

```
# Estandarización de las características con calculo de la media

# Pasamos de formato la hora para que pueda ser procesada
dataset['Hora'] = pd.to_datetime(dataset['Hora'], format='%H:%M').dt.hour * 60 + pd.to_datetime(dataset['Hora'], format='%H:%M').dt.minute
```

Fig. 34.- Código para convertir la hora de medición en minutos.

Los datos de la hora de la medición están almacenados en el arreglo dataset. Para cada medición, se extrae la hora con la función dt.hour, y posteriormente se multiplica por 60. A este valor se le suma el resultado de dt.minute, obteniendo así la hora expresada en minutos. Este nuevo valor se guarda en la misma variable dataset.

Los valores correspondientes a la hora de medición (en minutos), Vout330 y índice UV son almacenados en el arreglo caracteristicas_estandarizar, el cual se utiliza para aplicar la normalización (ver Fig. 35).

```
# declaramos las columnas que vamos a estandarizar
caracteristicas_estandarizar = ['Hora', 'Vout330','Indice UV float']
```

Fig. 35.- Código para estandarizar los datos.

Para aplicar Z-score, es necesario calcular la media (μ) y la desviación estándar (σ) de cada variable, y luego aplicar la fórmula:

$$Z = \frac{x - \mu}{\sigma} \tag{2}$$

- x: Es el valor original
- μ: es la media (Promedio)
- σ: desviación estándar
- z: valor estandarizado

El cálculo de la media y la desviación estándar se realiza dentro de un ciclo for, y los resultados se almacenan en las variables mean y std, como se muestra en la Fig. 36. Luego, para cada medición, se calcula el valor estandarizado correspondiente y se guarda en la variable dataset.

```
medias = []
desviaciones = []
for caracteristica in caracteristicas_estandarizar:#el bucle paraq que calcule la media
    mean = dataset[caracteristica].mean() #promedio
    std = dataset[caracteristica].std()#desviacion estandar
    dataset[caracteristica] = (dataset[caracteristica] - mean)/std #se aplica la formula
    medias.append(mean)
    desviaciones.append(std)
```

Fig. 36.- Código para estandarizar los datos

Como resultado del procesamiento, se obtuvieron los valores de media y desviación estándar de cada característica, los cuales se muestran en la Tabla 2.

Tabla 2.- Media y desviación estándar de cada característica.

Característica	Media (μ)	Desviación estándar (σ)
Hora (en minutos)	755.00	238.15
Vout330	2.25	1.07
Índice UV (float)	2.43	1.59

En la Fig. 37a se observan los datos originales en su formato de origen, mientras que en la Fig. 37b se presentan los datos ya normalizados, todos dentro de una misma escala.

	Hora	Vout330	Indice UV float
0	05:50	0.0147	0.56
1	06:00	0.0152	0.65
2	06:10	0.0158	0.66
3	06:20	0.0180	0.65
4	06:30	0.0329	0.65

	Hora	Vout330	Indice UV float
0	-1.700590	-2.086033	-1.172157
1	-1.658600	-2.085567	-1.115669
2	-1.616610	-2.085008	-1.109393
3	-1.574621	-2.082959	-1.115669
4	-1.532631	-2.069079	-1.115669

Fig. 37.- Datos de medición. a) Originales, b) Normalizados.

División del conjunto de datos.

Los datos ya normalizados se separan en dos conjuntos para el entrenamiento de la RNA:

- El conjunto de datos de entrenamiento el cual corresponde al 80% de los datos totales (65 datos).
- El conjunto de datos de prueba el cual corresponde al 20% de los datos totales (17 datos).

El código para separar los conjuntos se muestra en la Fig. 38. En las primeras líneas, se separan las características (hora y Vout330) y la variable objetivo (Índice UV float). En la última línea, se almacenan los datos en las variables x_train y y_train para el entrenamiento, y x_test y y_test para la prueba. Para esta división se utilizó la función train_test_split, con el parámetro test_size=0.2, lo que indica que el 20% de los datos será de prueba.

```
# Separación de las características y las variables objetivo
caracteristicas = dataset[['Hora','Vout330']] #los datos de entrada
targets = dataset[['Indice UV float']] #el dato de salida
# Construcción de los conjuntos de entrenamiento y prueba
x_train, x_test, y_train, y_test = train_test_split(caracteristicas, targets, test_size=0.20)
```

Fig. 38.- Código para separar datos de entrenamiento y prueba.

Una vez realizada esta separación, el siguiente paso es la creación del modelo de la RNA, de acuerdo con el diseño de la Fig. 31.

Creación de la red neuronal

Con los conjuntos de datos preparados, se procedió a codificar la red neuronal artificial (ver Fig. 39). Lo primero es definir el modelo el cual será secuencial. Las siguientes líneas de código definimos la estructura de nuestra RNA:

- La línea 2 definimos la primera capa oculta donde asignamos 5 neuronas, la función de activación es sigmoide, definimos que usara bias y como ultimo parámetro en input_shape agregamos las neuronas de entrada en este caso dos.
- La línea 3 definimos la segunda capa oculta con 3 neuronas y la función de activación es sigmoid.
- La línea 4 definimos nuestra neurona de salida y la función de activación linear

```
# Definición del modelo
modelo = Sequential()
modelo.add(Dense(5, activation='sigmoid', use_bias = True, input_shape = (2,)))
modelo.add(Dense(3, activation='sigmoid'))
modelo.add(Dense(1, activation='linear'))
```

Fig. 39.- Creación del modelo de la red neuronal.

A continuación, se definió el optimizador que permitirá entrenar la red neuronal (ver Fig. 40). En la primera línea se configuró el optimizador Adam, estableciendo una tasa de aprendizaje de 0.1. Finalmente, se compiló el modelo utilizando como función de pérdida el Error Cuadrático Medio (MSE).

```
# Definición del optimizador
adam = optimizers.Adam(learning_rate = 0.1)#hipermarametro
# Compilación del modelo
modelo.compile(loss='mean_squared_error', optimizer=adam, metrics=['mse'])
# como calculara el error con el error cuadratico medio y las metricas mse es solo para la observacion
```

Fig. 40.- Código del optimizador de la red neuronal.

Entrenamiento de la red neuronal

En la Fig. 41 se muestra el código con el que se inició el entrenamiento de la red neuronal. Se definió el uso de x_train y y_train como datos de entrada y salida, respectivamente, y se estableció un número total de 300 épocas. Para validar el entrenamiento, se empleó el conjunto de prueba.

```
# Entrenamiento del modelo
M = modelo.fit(x_train, y_train, epochs=300, verbose=1, validation_data=(x_test, y_test), batch_size=100,shuffle=True)
#verbose es para ver como va avanzado el modelo por epocas
#batch_size es el numero de registro que agrra por epoca y no tarde mucho
#shuffle en true no tiene razon si no tiene un batch es para que no tenga un sesgo en el modelo
```

Fig. 41.- Código del optimizador de la red neuronal.

Evaluación del modelo

En la Fig. 42 se presenta la gráfica del entrenamiento y evaluación de la RNA. En ella se puede observar que el MSE tiende a acercarse a 0 antes de la época 50. A partir de la época 100, el comportamiento del error se estabiliza, lo que indica que la red neuronal ha alcanzado un punto de convergencia estable.

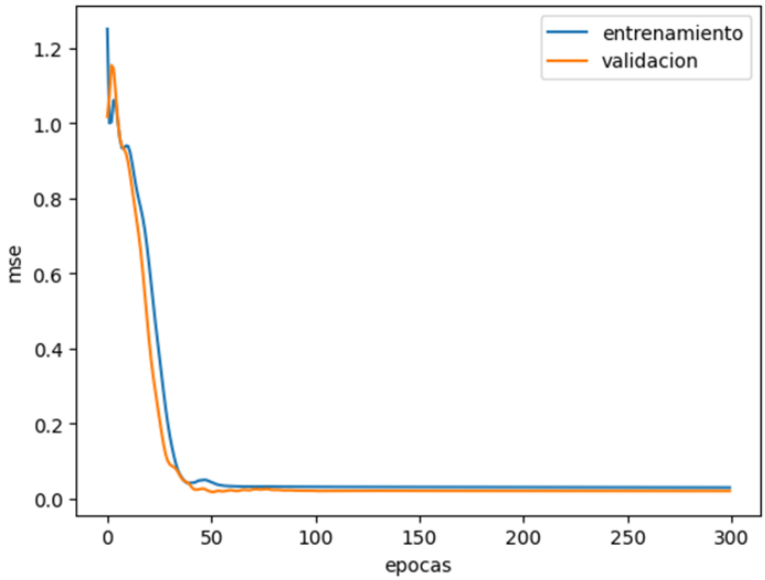


Fig. 42.- Evaluación y resultados de la red neuronal entrenada

Para evaluar el ajuste de las predicciones generadas por la RNA, se utilizó el coeficiente de determinación R^2 (r2_score), el cual permite ver el ajusta de los datos.

Para ello, se importaron las librerías correspondientes y se realizaron dos evaluaciones:

- **r2_train**: Valor obtenido al aplicar modelo.predict sobre el conjunto de entrenamiento.
- **r2_test**: Valor obtenido al aplicar modelo.predict sobre el conjunto de prueba.

Ambos valores reflejan la precisión del modelo en sus predicciones. Como se observa en la Fig. 43, el resultado del coeficiente R^2 para ambas pruebas fue de 0.99, lo que indica que la red neuronal presenta un excelente nivel de ajuste tanto con los datos de entrenamiento como con los de prueba.

```
# r2_score calcula el coeficiente de determinación (R^2), que mide qué tan bien se ajustan las predicciones..
from sklearn.metrics import r2_score
# Calcula R^2 en los datos de entrenamiento, evaluando el rendimiento del modelo en datos conocidos.
r2_train = r2_score(y_train, modelo.predict(x_train))
# Calcula R^2 en los datos de prueba, evaluando la capacidad de generalización del modelo.
r2_test = r2_score(y_test, modelo.predict(x_test))

print('r2_train = ', r2_train)  # Imprime el valor de R^2 para el conjunto de entrenamiento.
print('r2_test = ', r2_test)  # Imprime el valor de R^2 para el conjunto de prueba.
```

```
3/3 ━━━━━━━━━━━━━━━━━━━━ 1s 185ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 118ms/step
r2_train =  0.9904758334159851
r2_test =  0.9923619627952576
```

Fig. 43.- Valor de la predicción de los datos usando la red neuronal.

El código completo se encuentra en el **Anexo A.4**.

Resultados e implementación

Una vez evaluado el funcionamiento de la RNA, se procedió a extraer los pesos y sesgos (bias) de cada capa del modelo, los cuales se utilizarán en la construcción del Prototipo 4. En la Fig. 44 se muestran los valores de los pesos y sesgos correspondientes a cada neurona en cada capa.

Capa oculta 1			
# Neurona	Peso 1	Peso 2	Sesgo
1	2.81	2.35	-1.49
2	-1.17	2.26	-1.38
3	4.12	-3.28	-0.48
4	2.35	2.14	-1.09
5	0.82	-2.68	2.43

Capa oculta 2						
# Neurona	Peso 1	Peso 2	Peso 3	Peso 4	Peso 5	Sesgo
1	2.44	1.17	-2.96	1.41	-2.27	-0.93
2	-1.84	-0.69	1.20	-1.10	2.75	0.62
3	-1.68	-1.57	1.30	-1.29	-0.66	-0.21

Capa de salida				
# Neurona	Peso 1	Peso 2	Peso 3	Sesgo
1	2.41	-1.67	-0.36	0.62

Fig. 44.- Pesos y sesgos de cada neurona por capa.

Se realizó una comparación entre los datos obtenidos con el sensor ML8511, los resultados del modelo matemático y los generados por la red neuronal artificial.

En la Fig. 45 se presenta dicha comparación, en la cual se muestran tres curvas:

- La medición de la radiación UV realizada con el sensor ML8511 (línea punteada),
- La estimación obtenida mediante el modelo matemático (línea amarilla),
- Y la estimación generada por la RNA (línea verde).

Como se puede observar, la estimación de la radiación UV obtenida con la RNA muestra un mejor ajuste que la realizada mediante el modelo matemático. Es importante resaltar que, aunque ambos modelos utilizaron los mismos datos de entrada, el entrenamiento de la RNA le permite lograr un mejor ajuste en el rango horario de 10:30 a 13:30 horas, como se aprecia en la Fig. 45.

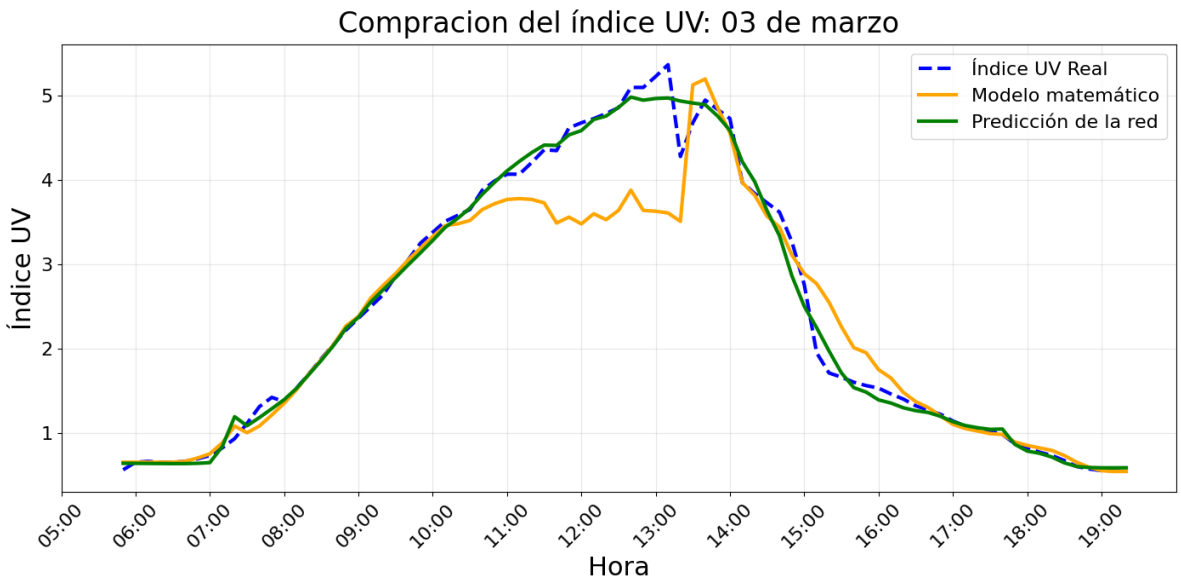


Fig. 45.- Comparación entre el modelo matemático y la RNA del día 03 de marzo del 2024.

Para validar este comportamiento, se compararon ambos modelos utilizando datos de días diferentes: el día anterior (02 de marzo) y el día posterior (04 de marzo).

En la Fig. 46, correspondiente al 02 de marzo, se observa que la red neuronal mantiene un mejor ajuste respecto a los datos del sensor, mientras que el modelo matemático presenta una diferencia significativa, especialmente en el mismo rango de horas.

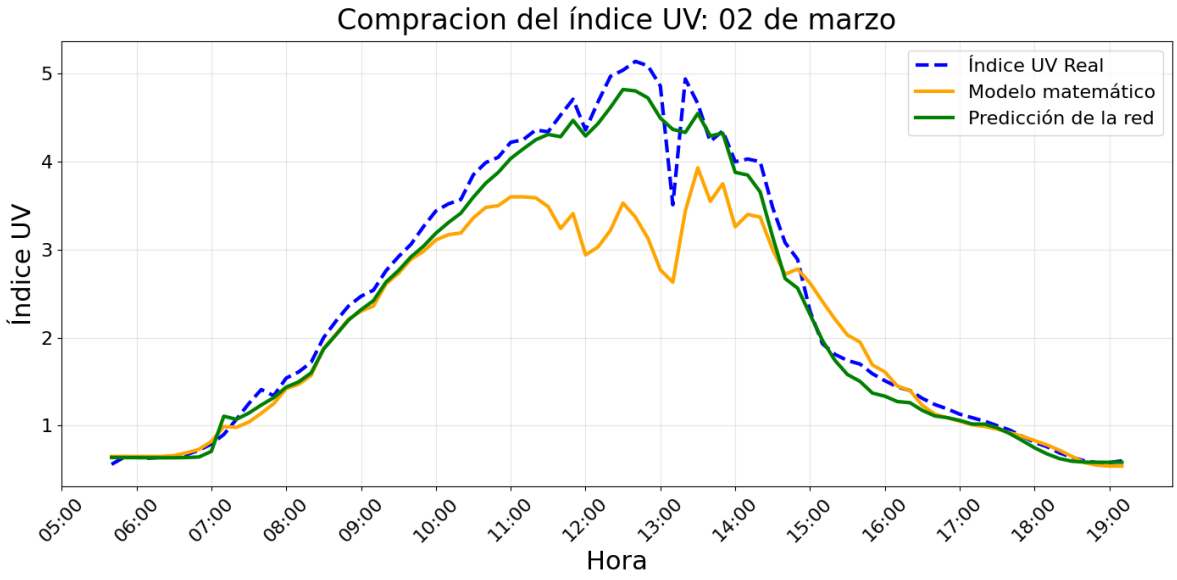


Fig. 46.- Comparación entre el modelo matemático y la RNA del día 02 de marzo del 2024.

El día 04 de marzo se observó una tendencia similar, como se muestra en la Fig. 47, la RNA sigue mostrando un ajuste más preciso que el modelo matemático, confirmando su mayor capacidad de generalización.

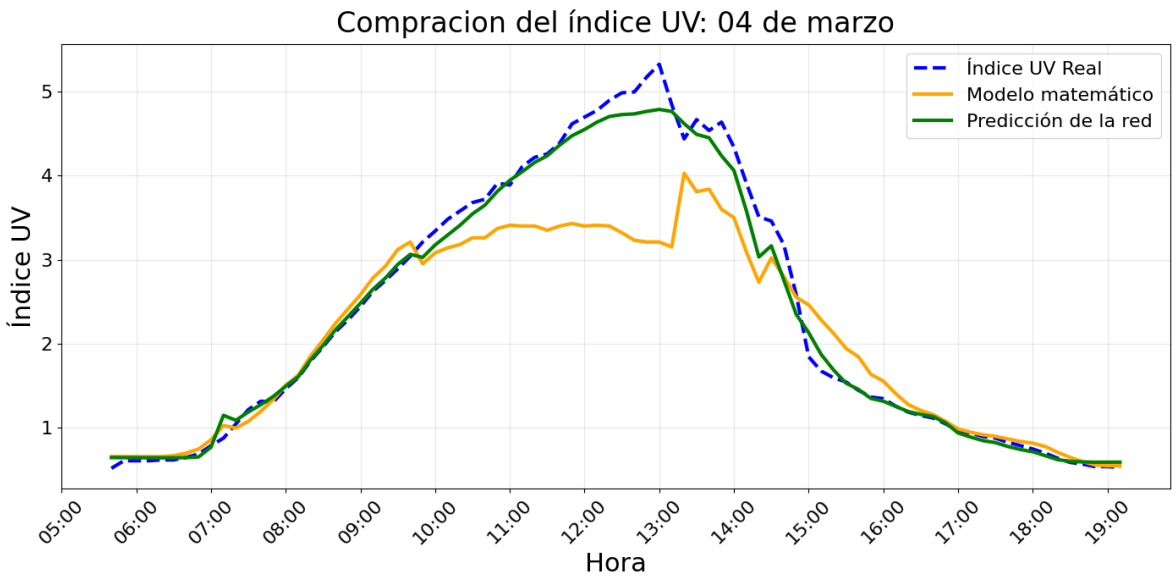


Fig. 47.-Comparación entre el modelo matemático y la RNA del día 04 de marzo del 2024.

Como conclusión de este prototipo, se puede afirmar que la red neuronal presenta un mayor ajuste con los datos obtenidos por el sensor ML8511, manteniéndose constante en diferentes días. Para una validación más exigente, se realizó una última comparación utilizando los datos del día 08 de abril del 2024, fecha en la que ocurrió un eclipse solar.

En la Fig. 48, se muestra que la RNA sobreestimó ligeramente los valores de radiación UV durante el eclipse, excepto en el punto de máxima ocultación solar, donde se acercó más al valor real. En contraste, el modelo matemático subestimó los valores antes del eclipse y los sobreestimó después del mismo.

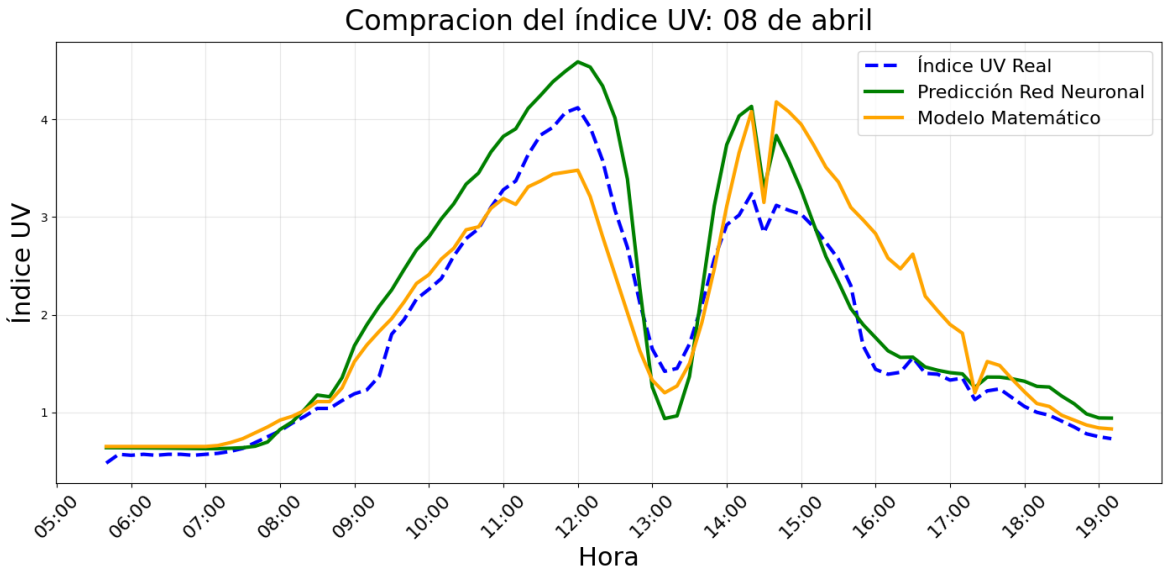


Fig. 48.- Comparación entre el modelo matemático y la RNA del día 08 de abril del 2024 (Eclipse solar).

Por lo tanto, el siguiente paso en el desarrollo del prototipo consiste en migrar los parámetros de la RNA entrenada (a partir de un solo día de datos) hacia una placa Arduino, con el objetivo de realizar estimaciones en tiempo real. En esta nueva fase, se realizará la comparación entre la predicción generada por la RNA embebida en la placa y los valores obtenidos en tiempo real por el sensor UV.

Prototipo 4

Análisis de requerimientos

El objetivo de este prototipo es la implementación de la red neuronal en un dispositivo portátil, en este caso una tarjeta de desarrollo Arduino. La intención es del dispositivo es realizar las mediciones de radiación UV con un sensor comercial y compararlas con la estimación que proporcione la RNA a partir de la medición de la intensidad luminosa con una fotorresistencia.

Por lo cual se plantean los siguientes requerimientos para este prototipo:

- LA RNA se implementará en una tarjeta Arduino NANO.
- Los parámetros obtenidos del entrenamiento de la RNA serán ingresados mediante el programa.
- La RNA estimara el índice UV en tiempo real utilizando una fotorresistencia.
- El sensor ML8511 para obtener el índice UV en tiempo real.
- Este prototipo debe mostrar en tiempo real a través de un led RGB la estimación del índice UV de la RNA y del sensor ML8511.

Diseño

Diagrama a bloques.

Como se muestra en el diagrama a bloques (Ver Fig. 49), el prototipo tiene conectado los siguientes componentes:

- El divisor de voltaje: Se implementa el divisor de voltaje con la resistencia de 330Ω y fotorresistencias (LDR), la cual va a generar un voltaje de acuerdo con la radiación de luz visible, este dato se enviará la placa Arduino para posteriormente enviar el dato a la placa Arduino será procesada por la RNA.
- RTC: este sensor nos ayudará a obtener la hora para que su valor analógico sea enviado a la RNA.
- La red neuronal: está se integra dentro de la placa Arduino para poder procesar la información del divisor de voltaje
- El sensor ML8511

- Los leds RGB nos servirá para el indicar el nivel del índice uv uno a través de la RNA y el otro del sensor ml8511

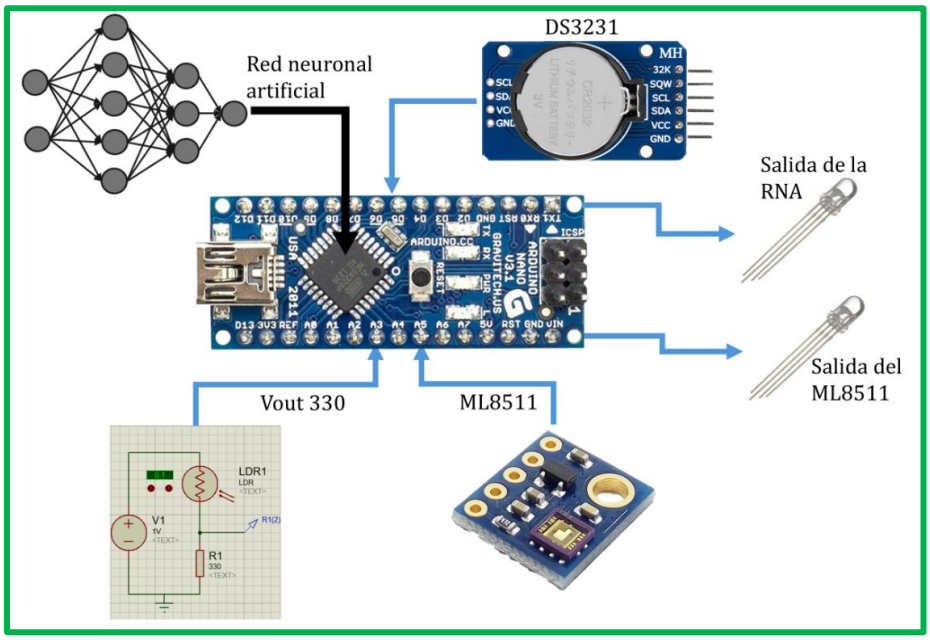


Fig. 49.- Diagrama a bloques del prototipo 4.

Diagrama eléctrico.

El diagrama eléctrico (Fig. 50) se presentan las conexiones de los componentes principales del sistema: el divisor de voltaje con su resistencia de 330 ohms, el sensor ML8511, un RTC DS3231 y dos leds RGB, todo esto conectado a la placa Arduino, esta opera con una fuente de 5V y 3.3V y todos conectados a tierra (GND).

Conexión de los sensores

- La salida del divisor de voltaje se envía al pin analógico A3 del Arduino.
- El sensor ML8511 envía el dato en una señal analógica al pin analógico A0 del Arduino.
- El Arduino enviara voltaje de 3.3 al sensor ML8511, al pin EN y al pin analógico A3.

Conexión del DS3231

- El RTC DS3231 envía los datos en una señal a los pines A4 y A5 del Arduino.

Conexión de los leds RGB

- El Arduino enviara voltaje de 5V a ambos leds.
- Un led RGB recibe la señal de salida del sensor ML8511.
- El otro led RGB recibe la señal de salida de la RNA.

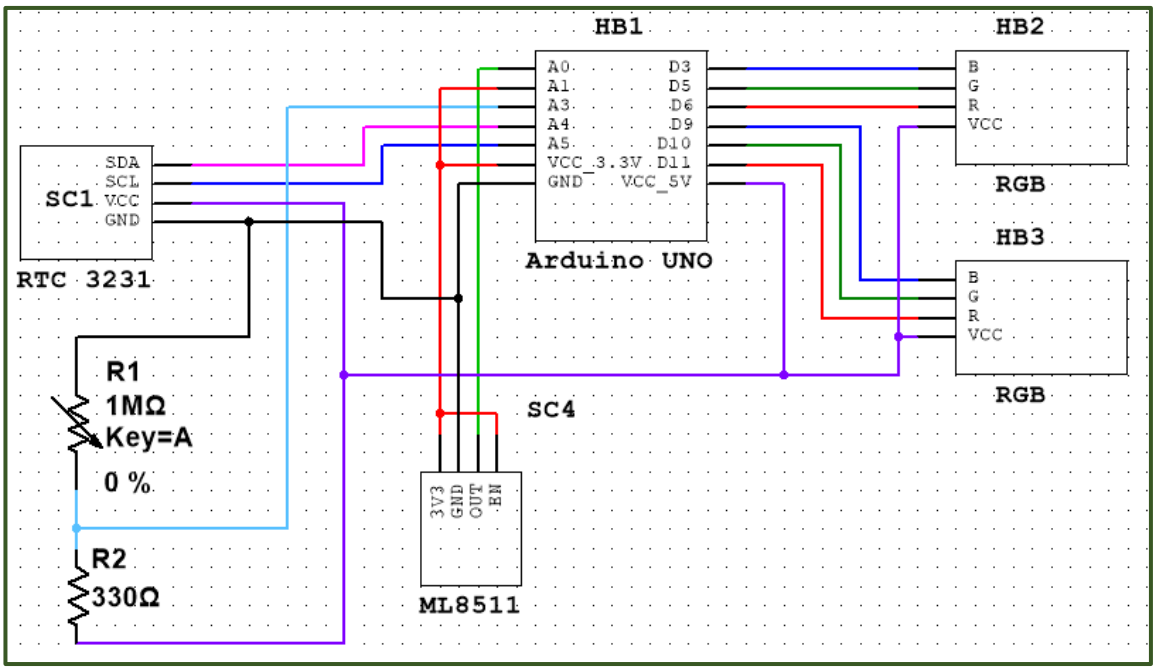


Fig. 50.- Diagrama eléctrico del prototipo 4.

Diagrama de flujo.

Como se puede observar en el diagrama de flujo (Fig. 51) el programa inicia con la declaración de variables, la configuración de los pines y la inicialización de los sensores. También declaramos la estructura, pesos y bias de la RNA, así como sus funciones de activación, seguido de la desviación estándar y media de cada entrada y salida.

A continuación, comienza el proceso de lectura de los sensores. El primer paso es realizar la lectura del divisor de voltaje y posteriormente convertir ese dato en voltaje.

Como siguiente paso es la lectura de la hora del RTC.

Una vez que tenemos estos datos, se envían a la RNA la cual se encargara de estimar el índice UV con esos datos de entrada (Voltaje del divisor y la hora del RTC), una vez que la RNA termino su proceso, el resultado obtenido es pasado a la escala del índice UV.

El siguiente paso es obtener la lectura del sensor ML8511 y posteriormente convertir esa lectura en voltaje. Este dato se envía a una función la cual se encarga de convertir el voltaje a la escala del índice UV

Una vez que tenemos el índice UV de la RNA y del sensor pasamos a mostrar el resultado en los leds RGB.

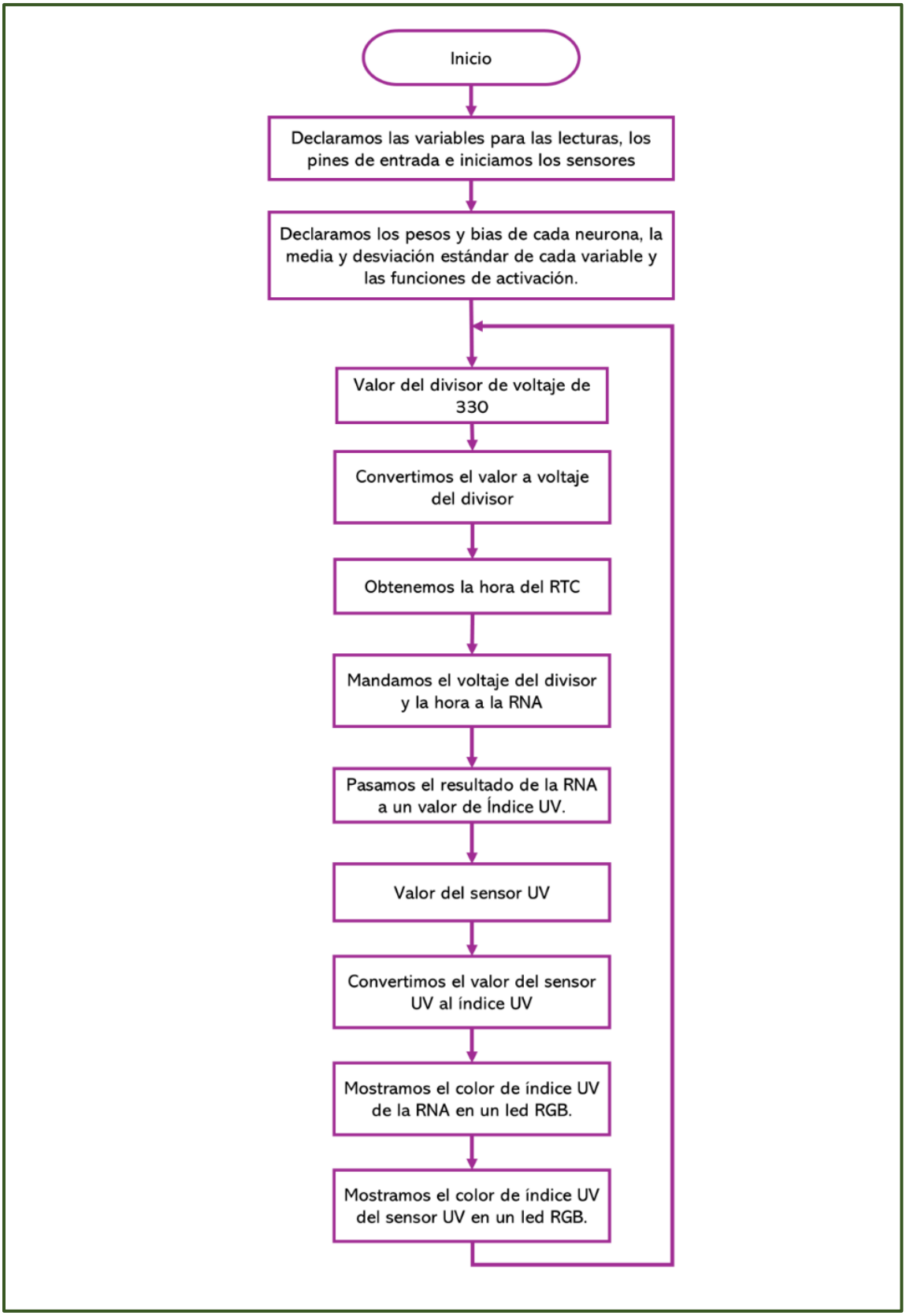


Fig. 51.- Diagrama de flujo del prototipo 4.

Implementación

El siguiente paso es la codificación del diagrama de flujo y la implementación del prototipo en físico, tomando en cuenta el diagrama eléctrico descrito anteriormente. Para esto, utilizamos Arduino IDE y es necesario cargar la librería del RTC.

Inicialización del Código

Definimos las librerías y los pines de conexión del RTC y del sensor UV, Declaramos las variables necesarias para los sensores y los pines de los leds RGB. Ver Fig. 52.

```
1   #include <RTClib.h>
2   #include <Wire.h>
3
4   RTC_DS3231 rtc;
5
6   const int Rf = 6;
7   const int Gf = 5;
8   const int Bf = 3;
9
10  const int Ru = 11;
11  const int Gu = 10;
12  const int Bu = 9;
13
14  float uv;
15  int UVOUT = A3;
16  int REF_3V3 = A1;
```

Fig. 52.- Librería y variables del código.

Inicialización de la RNA

Después de la declaración de los pines y las librerías, codificamos la RNA en el Arduino IDE (ver Fig. 53).

Como se observa, de la línea 20 a la 23 se define la primera capa oculta, utilizando la variable w_capa_oculta1, la cual es una matriz de 2x5, donde el número 2 representa las neuronas de la capa de entrada, y el número 5 corresponde a las 5 neuronas de la capa oculta 1.

Aquí se definen los pesos de cada neurona. En la línea 21 se establecen 5 valores que representan los pesos de las 5 neuronas de la capa oculta conectadas con la primera neurona de la capa de entrada (Hora). Lo mismo sucede en la línea 22, donde se definen los pesos correspondientes a la segunda neurona de entrada (Voltaje del divisor de voltaje).

De la línea 25 a la 27 se definen los bias de cada neurona de la capa oculta 1, utilizando un arreglo simple.

De la línea 29 a la 35 se define la segunda capa oculta, con la variable w_capa_oculta2, la cual es una matriz de 5x3. Aquí, el número 5 representa las neuronas de la capa oculta 1, y el 3 representa las 3 neuronas de la capa oculta 2.

En la línea 30, se definen 3 pesos que representan las conexiones entre la primera neurona de la capa oculta 1 y las 3 neuronas de la capa oculta 2. Lo mismo ocurre en las líneas 31, 32, 33 y 34 para las demás neuronas.

En la línea 37, se definen los bias de cada neurona de la capa oculta 2, nuevamente usando un arreglo simple.

La línea 38 define la variable w_capa_salida, que es un arreglo simple, ya que la capa de salida cuenta con una sola neurona. Por ello, solo se definen 3 pesos, correspondientes a las conexiones con la capa oculta 2. En la línea 39, se define el bias de la neurona de salida.

De la línea 41 a la 43, se definen las medias de cada variable:

- La media de la Hora, que es la primera entrada de la RNA.
- La media del Voltaje del divisor, que es la segunda entrada.
- La media del Índice UV, que representa la salida.

Finalmente, en las líneas 45, 46 y 47, se define la desviación estándar de cada variable, correspondiente a las mismas entradas y salida mencionadas anteriormente.

```
20  double w_Capa_oculta1[2][5] = {
21    { 2.8060021, -1.1711409, 4.122045, 2.350187, 0.81597537 },
22    { 2.3484783, 2.2630205, -3.2768, 2.1375208, -2.6784663 }
23  };
24
25  double b_Capa_oculta1[5] = {
26    -1.4936172, -1.3820752, -0.48103565, -1.0889859, 2.4271436
27  };
28
29  double w_Capa_oculta2[5][3] = {
30    { 2.4358888, -1.8404518, -1.6751294 },
31    { 1.1724445, -0.6933083, -1.5696467 },
32    { -2.957599, 1.1990123, 1.3012772 },
33    { 1.4084017, -1.1040592, -1.2870497 },
34    { -2.26612, 2.7496896, -0.6572841 }
35  };
36
37  double b_Capa_oculta2[3] = { -0.9334416, 0.62091994, -0.20813814 };
38  double w_Capa_salida[3] = { 2.4138823, -1.6681055, -0.36259282 };
39  double b_Capa_salida = 0.6181946;
40
41  double mediaVout = 2.2540182926829266;
42  double mediaHora = 755.0;
43  double mediaUV = 2.4275609756097563;
44
45  double desviacionVout = 1.0734817809845172;
46  double desviacionHora = 238.15261213488014;
47  double desviacionUV = 1.593268680677765;
```

Fig. 53.- Parámetros y estructura de la RNA.

Por último, definimos las funciones de activación, en el caso de esta RNA se usaron la función sigmoid y linear. Ver Fig. 54.

```
49  //------------------- Funciones de activación -------------------//
50
51  double sigmoid(double x) {
52    return 1.0 / (1.0 + exp(-x));
53  }
54
55  double linear(double x) {
56    return x;
57  }
```

Fig. 54.- Funciones de activación.

Procesamiento de los datos en la RNA

Definimos la función la cual representa el proceso de la RNA (Fig. 55). Esta función se llama IndiceUV donde tendrá dos entradas:

- a que es la hora
- b que es el voltaje del divisor.

En la línea 154 y 155, se declara dos arreglos (salida_oculta1 y salida_oculta2), donde se almacenan los valores de salida de cada capa oculta, así como una variable sum, que representa la capa de salida.

En la línea 158, se inicia un ciclo for de 5 iteraciones, lo cual corresponde a las 5 neuronas de la capa oculta 1.

Dentro del ciclo, en la línea 159, se calcula la suma ponderada de la neurona correspondiente (ver Ecuación 3). A este valor se le suma el bias, y el resultado se guarda en la posición correspondiente del arreglo salida_oculta1. Luego, ese valor pasa por la función sigmoide, y el resultado final se guarda en la misma posición. Este proceso se repite para las 4 neuronas restantes.

(3)

$$Suma\ ponderada = (a * w_capa_oculta1) + (b * w_capa_oculta1) + bias$$

Una vez obtenidos los valores de salida de la primera capa oculta, se inicia un ciclo for anidado para procesar la segunda capa oculta. En la línea 163 el primer for recorre las 3 neuronas de esta capa. Dentro de esta cada neurona se inicializa con un valor de 0.

En la línea 165, otro for recorre las salidas de la capa oculta 1. En la línea 166, se realiza la suma ponderada y se acumula el valor en la neurona actual. Este proceso se repite 5 veces más. Posteriormente, se suma el bias correspondiente y el resultado pasa

por la función sigmoide, guardándose de nuevo en la posición correspondiente. Este proceso se repite 2 veces más.

Finalmente, en la línea 172 se ejecuta un último ciclo for para calcular la suma ponderada de la capa de salida. En cada iteración, el resultado se acumula en la variable sum.
En la línea 176, se retorna el resultado de la función lineal, a la cual se le pasa el valor acumulado (sum) más el bias correspondiente.

```
153  double IndiceUV(double a, double b) {
154    double salida_oculta1[5];
155    double salida_oculta2[3];
156    double sum = 0;
157
158    for (int i = 0; i < 5; i++) {
159      salida_oculta1[i] = a * w_Capa_oculta1[0][i] + b * w_Capa_oculta1[1][i] + b_Capa_oculta1[i];
160      salida_oculta1[i] = sigmoid(salida_oculta1[i]);
161    }
162
163    for (int i = 0; i < 3; i++) {
164      salida_oculta2[i] = 0;
165      for (int j = 0; j < 5; j++) {
166        salida_oculta2[i] += salida_oculta1[j] * w_Capa_oculta2[j][i];
167      }
168      salida_oculta2[i] += b_Capa_oculta2[i];
169      salida_oculta2[i] = sigmoid(salida_oculta2[i]);
170    }
171
172    for (int i = 0; i < 3; i++) {
173      sum += salida_oculta2[i] * w_Capa_salida[i];
174    }
175
176    return linear(sum + b_Capa_salida);
177  }
```

Fig. 55.- Función para hacer el proceso de la RNA.

Lectura y Procesamiento de datos en loop()

Durante cada interacción del loop(), lo primero es realizará la lectura del rtc y del divisor de voltaje, la normalización de esto dos datos para la RNA y por último la conversión índice UV de la estimación de la red neuronal,

Como se puede ver en la Fig. 56 se guarda la hora del RTC en la variable fecha y el valor del divisor de voltaje (A0) en las variables valor.

El siguiente paso es convertir la hora en minutos esto se ve en las líneas 87 y 88 donde dividimos la hora medida en horas y minutos y en la línea 89 en la variable HoraTotal guardamos la hora en minutos.

En la línea 91 convertimos el valor analógico del divisor en voltaje en voltaje.

A continuación, normalizamos los dos valores utilizando la fórmula de Z-score, esto a que el inicio del código declaramos la media y desviación estándar de cada dato, y resultado de la formula se guarda en dos variables, esto se puede observar en las líneas 93 y 94.

Una vez estandarizados los datos los mandamos a nuestra red neuronal la cual se definimos anteriormente y el valor resultante se guarda en la variable indexUV. Como este resultado se encuentra normalizado, es necesario aplica una desnormalización para convertirlo a la escala original para eso usando la fórmula de Z-score despejamos la variable X y el resultado se muestra en la Ecu. 4. El proceso de despeje se encuentra en el **Anexo A.5**.

$$x = (z * \sigma) + \mu \tag{4}$$

Este valor se guarda en una nueva variable la cual se usará más adelante.

```
84  DateTime fecha = rtc.now();
85  float valor = analogRead(A0);
86
87  int hora = fecha.hour();
88  int minutos = fecha.minute();
89  int HoraTotal = hora * 60 + minutos;
90
91  double vout330 = 5.0 - (valor * 5.0 / 1023.0);
92
93  double horaEstandar = (HoraTotal - mediaHora) / desviacionHora;
94  double VoutEstandar = (vout330 - mediaVout) / desviacionVout;
95  double indexUV = IndiceUV(horaEstandar, VoutEstandar);
96  double indiceUVDes = indexUV * desviacionUV + mediaUV;
```

Fig. 56.- Lectura del divisor de voltaje.

El siguiente paso es la lectura del sensor UV para convertir su valor analógico a la escala del índice UV. Como se puede observar en la Fig. 57, en la línea 104 y 105 guardamos las lecturas de este sensor. En la línea 106 convertimos el valor analógico a voltaje para que en la línea 107 pueda convertir ese valor de voltaje a una escala de índice UV. Por último, guardamos ese valor en una variable llamada uv.

```
103 // Sensor UV ML8511
104 float uvLevel = analogRead(UVOUT);
105 float refLevel = analogRead(REF_3V3);
106 float outputVoltage = uvLevel * 3.3 / refLevel;
107 float uvIntensity = mapfloat(outputVoltage, 0.99, 2.9, 0.0, 15.0);
108 uv = uvIntensity;
109
```

Fig. 57.- Lectura del sensor UV.

Como último paso del loop() es mostrar el resultado del índice UV de la RNA y del sensor a través de unos leds RGB. Como se puede observar en la Fig. 58, de la línea 111 a la 121 dependiendo el valor del índice UV el led prendera de la siguiente manera:

- Si el valor es menor a 2 el led prendera de color verde.
- Si el valor es menos de 6 el led prendera de color amarillo.
- Si el valor es menor de 8 el led prendera de color naranja.
- Si el valor es menor de 11 el led prendera de color rojo
- Si el valor no está en este rango prendera de color morado.

Esta misma escala se aplica con el valor obtenido del sensor UV esto se puede ver en las líneas 123 hasta la línea 133. Con esto logramos que cada led prenda de acuerdo con el valor de cada sensor para nosotros podemos ver si el resultado es similar o no.

```
110   // LED RNA
111   if (indiceUVDes < 2.0) {
112     setColor(255-0, 255-255, 255-0); // Verde
113   } else if (indiceUVDes < 6.0) {
114     setColor(255-255, 255-255, 255-0); // Amarillo
115   } else if (indiceUVDes < 8.0) {
116     setColor(255-255, 255-165, 255-0); // Naranja
117   } else if (indiceUVDes < 11.0) {
118     setColor(255-255, 255-0, 255-0); // Rojo
119   } else {
120     setColor(255-128, 255-0, 255-128); // Morado
121   }
122
123   if (uv < 2.0) {
124     setColorUV(255-0, 255-255, 255-0); // Verde
125   } else if (uv < 6.0) {
126     setColorUV(255-255, 255-255, 255-0); // Amarillo
127   } else if (uv < 8.0) {
128     setColorUV(255-255, 255-165, 255-0); // Naranja
129   } else if (uv < 11.0) {
130     setColorUV(255-255, 255-0, 255-0); // Rojo
131   } else {
132     setColorUV(255-128, 255-0, 255-128); // Morado
133   }
134
```

Fig. 58.- Condiciones para prender ambos leds RGB.

Código Completo

El código completo de este prototipo se encuentra en el **Anexo A.6**.

Resultados

En la Fig. 59 se presentan los datos correspondientes al 10 de agosto del 2025, obtenidos a partir del sensor UV. Estos datos fueron comparados con tres redes neuronales artificiales (RNAs):

- La primera fue entrenada únicamente con los datos del 3 de marzo de 2024 (línea naranja).
- La segunda fue entrenada con todo el conjunto de datos recopilado desde enero hasta septiembre de 2024 (línea verde).
- La tercera fue entrenada exclusivamente con los datos del 10 de agosto de 2025 (línea roja)

Como se puede observar, entre las 8:00 a.m. y las 10:30 a.m., las salidas de ambas RNAs se aproximan a los valores medidos por el sensor (línea azul punteada). Posteriormente, desde las 10:30 a.m. hasta alcanzar el cénit solar, la RNA entrenada con los datos del 3 de marzo mantiene una buena aproximación, mientras que la RNA entrenada con todo el conjunto de datos tiende a subestimar los valores reales, situándose por debajo de las mediciones del sensor.

No obstante, después del cénit, ambas RNAs comienzan a sobreestimar los valores del sensor. En este periodo, la RNA entrenada con los datos del 3 de marzo muestra una mayor desviación positiva, mientras que la RNA entrenada con todo el dataset se aproxima más a los datos reales.

Por último, la RNA entrenada con los datos del mismo día (10 de agosto de 2025) muestra el mejor desempeño general, ya que se ajusta de manera más precisa al comportamiento del sensor.

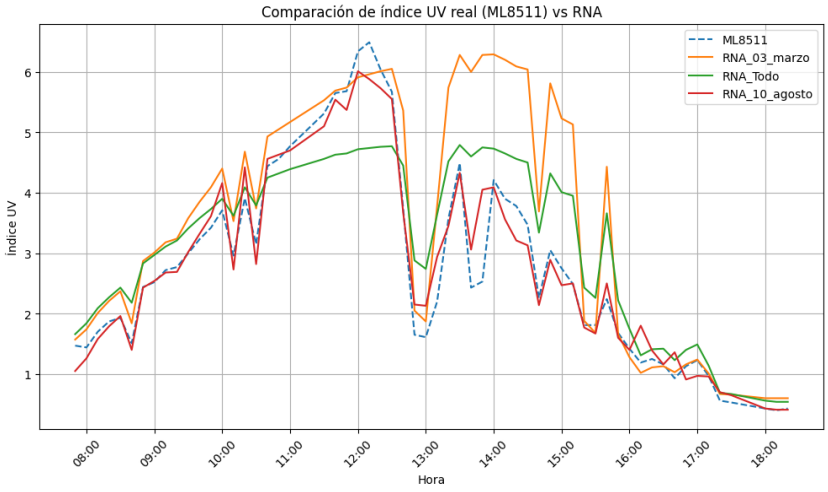


Fig. 59.- Resultados

En conclusión, este prototipo demuestra que una red neuronal artificial (RNA) es capaz de estimar la radiación UV utilizando como entrada la luz visible, medida mediante un sensor LDR conectado a un divisor de voltaje. Aunque aún presenta ciertas limitaciones, el modelo ha demostrado ser capaz de estimar la radiación UV en distintos momentos del año. Dichas limitaciones pueden estar relacionadas a que el conjunto de datos está incompleto.

Para comprender la relación entre las variables de entrada y la radiación UV, se utilizó el análisis de cofactores. Este análisis permite evaluar qué tan relacionadas están las variables, particularmente en este caso, con la radiación UV. Como se muestra en la Fig. 60, se presentan mapas de calor que indican el porcentaje de correlación entre las variables. En el caso del 3 de marzo, se observa una correlación del 80%, mientras que al considerar todo el conjunto de datos (de enero a septiembre), la correlación aumenta ligeramente a un 82%.

Estos resultados permiten validar nuestra hipótesis: *"Si la luz visible en exteriores es mayor, el índice UV será alto; y si la luz visible disminuye, el índice UV también disminuirá."*

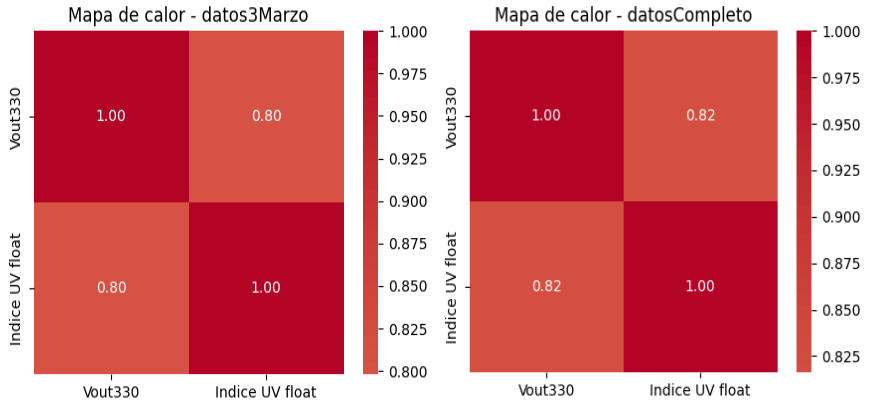


Fig. 60.- Mapas de calor

A partir de estos resultados, se observa que podrían existir otras variables que ayuden a mejorar la estimación del índice UV. Entre las variables analizadas, se encontró que la temperatura también guarda una relación con el índice UV. Por lo tanto, reformulamos la hipótesis inicial de la siguiente manera: *"Si la luz visible y la temperatura son altas, el índice UV será alto; y si la luz visible y la temperatura son bajas, el índice UV también será bajo."*

Con esta nueva hipótesis, se realizó un análisis de correlación entre las variables, tanto para cada mes individualmente como para el conjunto completo de datos. Como se muestra en la Fig. 62, se seleccionaron los tres meses con mayor correlación entre el índice UV, el voltaje de salida del divisor de voltaje y la temperatura. Estos resultados se describen a continuación:

- El mes de julio presentó una correlación del 89% entre el índice UV y el voltaje de salida del divisor, y del 85% con la temperatura.

- El mes de agosto mostró una correlación del 87% entre el índice UV y el voltaje de salida del divisor, y del 84% con la temperatura.
- El mes de septiembre evidenció una correlación del 84% entre el índice UV y el voltaje de salida del divisor, y del 81% con la temperatura.

Estos resultados respaldan la nueva hipótesis, y permiten concluir que la temperatura puede ser utilizada como una variable de entrada adicional para mejorar la estimación del índice UV. Todos los mapas de calor correspondientes a cada mes se encuentran en el **Anexo A.7**.

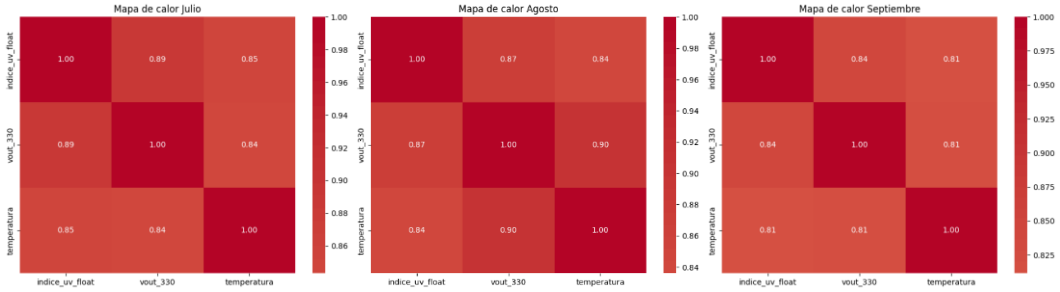
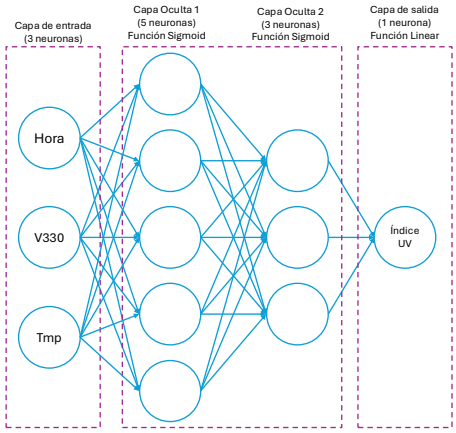


Fig. 61.- Mapas de calor de los 3 mejores meses.

Con base en los resultados obtenidos, se modificó la arquitectura de la red neuronal artificial (RNA) para incluir una nueva variable de entrada: la temperatura. La nueva configuración de la RNA se muestra en la Fig. 62, donde únicamente se ajustaron las variables de entrada, manteniendo iguales los demás parámetros del modelo.

Para el entrenamiento de esta nueva RNA, se utilizaron dos conjuntos de datos:

- Un conjunto que incluye los datos del 03 de marzo de 2024.
- Un conjunto que contiene todas las mediciones desde enero hasta septiembre de 2024.
- Un conjunto que tiene exclusivamente los datos del 10 de agosto de 2025.



Hiperparámetro	Descripción	Valor en el modelo
Estructura de la red	Número de neuronas en cada capa de la red (capa de entrada, ocultas, salida).	3 (entrada) 5 - 3 (Oculta) 1 (salida)
Función de activación	La función utilizada en cada capa para introducir no linealidad.	Sigmoid (capas ocultas), Lineal (capa de salida)
Optimización	Algoritmo utilizado para ajustar los pesos de la red.	Adam
Tasa de aprendizaje	La velocidad a la que el modelo ajusta sus pesos durante el entrenamiento.	0.1
Función de pérdida	Métrica utilizada para calcular el error del modelo.	Error cuadrático medio (MSE)
Número de épocas	El número de veces que todo el conjunto de datos de entrenamiento pasa por la red.	300

Fig. 62.- Nueva RNA con 3 variables de entrada.

Una vez entrenada la RNA (utilizando el prototipo 3), se probaron los modelos con los datos del 10 de agosto de 2025. Los resultados se presentan en la Fig. 63, donde se observa el comportamiento de las RNAs en relación con la variable de temperatura:

- RNA entrenada con el 03 de marzo (línea naranja): Antes del zenit, muestra un buen ajuste con los datos del sensor; sin embargo, entre las 11:30 y 12:30, se encuentra por debajo de las mediciones reales. Después del zenit, sobreestima los valores del sensor, aunque a partir de las 16:00 los datos vuelven a aproximarse a las mediciones reales.
- RNA entrenada con todo el dataset: Antes del zenit, presenta una subestimación respecto a los datos del sensor. Tras el zenit, aproximadamente entre las 14:00 y 16:00, se observa una sobreestimación de los valores reales.
- RNA entrenada con los datos del mismo día (10 de agosto de 2025): Antes del zenit, se ajusta casi completamente a las mediciones del sensor. Posteriormente, entre las 13:30 y 14:30, los datos no replican completamente el comportamiento del sensor, pero después de las 15:00, las predicciones se aproximan nuevamente a las mediciones reales.

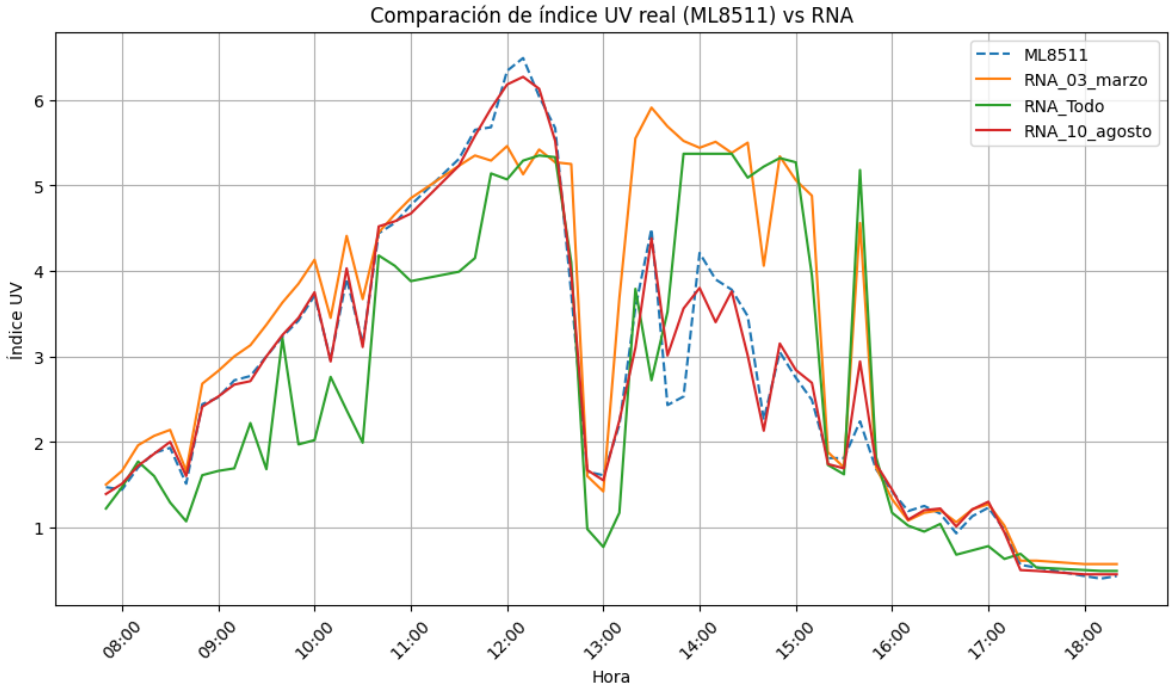


Fig. 63.- Resultados con RNA entrenada con temperatura

Como conclusión, se demuestra que la temperatura puede ser una variable que contribuye a mejorar la estimación del índice UV en ciertos intervalos horarios. Por lo tanto, con un conjunto de datos más amplio y la inclusión de otras variables relevantes, es posible mejorar la precisión del modelo hasta alcanzar una estimación más exacta del índice UV.

Conclusiones

Este trabajo de investigación demostró la viabilidad de un sistema de bajo costo para la estimación del índice de radiación ultravioleta (UV) mediante redes neuronales artificiales, utilizando como entrada principal mediciones de intensidad de luz visible obtenidas con sensores LDR. Las principales contribuciones se resumen a continuación:

1. **Desarrollo de una metodología innovadora:** Se diseñó e implementó un enfoque basado en redes neuronales artificiales que supera las limitaciones de los modelos matemáticos tradicionales para capturar la relación no lineal entre la radiación UV y la intensidad luminosa. La arquitectura propuesta, un perceptrón multicapa con dos entradas (intensidad luminosa y hora) y dos capas ocultas, demostró ser efectiva para esta tarea de regresión.
2. **Creación de un conjunto de datos público y especializado:** Se construyó y puso a disposición de la comunidad científica un dataset único que contiene mediciones simultáneas de variables ambientales (intensidad luminosa, temperatura, humedad) junto con el índice UV de referencia, recolectado en Ecatepec de Morelos, Estado de México. Este recurso, disponible en https://indiceuv.com/ , constituye una base valiosa para futuras investigaciones en monitoreo ambiental.
3. **Validación experimental con resultados significativos:** El modelo alcanzó un coeficiente de determinación (R^2) de 0.99 tanto en entrenamiento como en validación para datos del mismo día, confirmando su capacidad para estimar con precisión el índice UV a partir de mediciones de luz visible en condiciones controladas.
4. **Implementación exitosa en hardware embebido**: La red neuronal fue programada y ejecutada satisfactoriamente en una plataforma Arduino UNO, demostrando la factibilidad técnica de implementar esta solución en sistemas de bajo costo y consumo energético, con potencial para aplicaciones portátiles o desplegadas en campo.
5. **Contribución al monitoreo ambiental accesible:** El trabajo presenta una alternativa económica y escalable para el desarrollo de sistemas de alerta temprana de radiación UV, con aplicaciones potenciales en salud pública, educación ambiental y protección personal contra los riesgos de sobreexposición solar.

Trabajos futuros

A partir de los hallazgos de esta investigación, se identifican las siguientes líneas de desarrollo futuro:

Expansión y diversificación del conjunto de datos:

- Recolectar mediciones a lo largo de diferentes estaciones del año para capturar la variabilidad estacional en la relación UV-luz visible.

- Extender las mediciones a diversas regiones geográficas con condiciones climáticas contrastantes, con el fin de desarrollar modelos más robustos y generalizables.
- Establecer un sistema de monitoreo continuo que proporcione datos abiertos en tiempo real para la comunidad académica.

Optimización del modelo predictivo:

- Investigar la incorporación de variables meteorológicas adicionales (presión atmosférica, nubosidad estimada) que puedan mejorar la precisión del modelo.
- Explorar arquitecturas alternativas de redes neuronales (como redes recurrentes LSTM) capaces de capturar dependencias temporales en series de datos.
- Desarrollar técnicas de adaptación continua que permitan al modelo ajustarse a cambios estacionales sin requerir reentrenamiento completo.

Desarrollo de aplicaciones y prototipos:

- Diseñar un dispositivo portátil de alerta UV en formato de pulsera o wearable, implementando una versión simplificada del semáforo de la OMS con indicación por colores.
- Adaptar el modelo para su integración en aplicaciones móviles, aprovechando los sensores de luz ambiental presentes en smartphones.
- Implementar sistemas de visualización de datos en tiempo real para educación y concientización pública sobre riesgos de exposición solar.

Investigación de aspectos fundamentales:

- Estudiar sistemáticamente el impacto de condiciones atmosféricas específicas (nubosidad, contaminación, humedad) en la correlación entre radiación UV y luz visible.
- Desarrollar metodologías de calibración y validación que permitan la interoperabilidad entre dispositivos con diferentes características de sensor.
- Realizar estudios de usabilidad y aceptación social de sistemas de alerta UV basados en esta tecnología.
- La implementación de estas líneas de trabajo permitirá transformar este prototipo de investigación en una solución tecnológica madura, con potencial para contribuir significativamente a la prevención de enfermedades relacionadas con la exposición a radiación ultravioleta.

Referencias.

[1] Castanedo-Cázares, J. P., Torres-Álvarez, B., Portales-González, B., Martínez-Rosales, K., & Hernández-Blanco, D. (2016). *Análisis de la radiación solar ultravioleta acumulada en México*. Revista Médica del Instituto Mexicano del Seguro Social, 54(1), 26-31.

[2] V. Caviedes, J. C. Guzmán & L. F. Escobar *Análisis de la exposición a radiaciones solares por parte del personal del curso primario de vuelo de la escuela militar de aviación 'Marco Fidel Suárez', durante el primer semestre de 2019*. [online]. Disponible en: http://hdl.handle.net/20.500.12749/20563.

[3] D. A. R. Leal, J. P. V. Porto, y C. A. R. Algarín, El camino a las redes neuronales artificiales, Primera. Editorial Unimagdalena, 2021.

[4] E. Serna M. Desarrollo e Innovación en Ingeniería (2a ed.). Editorial Instituto Antioqueño de Investigación, 2017.

[5] R. F. López y J. M. F. Fernández, Las redes neuronales artificiales. Netbiblo, 2008.

[6] Matich, D. (2001). Redes Neuronales: Conceptos Básicos y Aplicaciones. Documento de trabajo. Universidad Tecnológica Nacional.

[7] P. Celso y B. Grado, "Redes Neuronales Artificiales: Historia y Actualidad," Mount Scopus Journal, vol. 2, no. 3, pp. 35–54, 2022.

[8] A. Abeliuk y C. Gutiérrez, «Historia y evoluación de la inteligencia artificial», Revista Bits de Ciencia, n.o 21, pp. 14-21, ago. 2021, [En línea]. Disponible en: https://revistasdex.uchile.cl/index.php/bits/article/view/2767

[9] "Introducción a los modelos de redes neuronales artificiales. El perceptrón simple y multicapa - Universidad de Zaragoza Repository", 2018. https://zaguan.unizar.es/record/69205

[10] J. R. H. González y V. J. M. Hernando, Redes neuronales artificiales: fundamentos, modelos y aplicaciones. RA-Ma Editorial, 1995.

[11] Palacio, F. (2003). Herramientas en GNU/Linux para estudiantes universitarios - Redes Neuronales con GNU/Linux. The Public's Library and Digital Archive. Online [Jun 2017].

[12] S. A. F. Ángel, «Análisis del índice UV para radiación UVB desde 2000 hasta 2022 en la Ciudad de México», 1 de mayo de 2024. https://hdl.handle.net/20.500.12371/21043

[13] E. Luccini, F. Orte, J. Lell, F. Nollas, G. Carbajal, y E. Wolfram, "Una revisión integral a la paleta de colores del Índice UV," Nota Técnica SMN 2022-127, 2022.

[14] Global Solar UV Index: A Practical Guide. 2002.

[15] Q. C. N. Eduardo, «Medidor índice UV portátil», 15 de noviembre de 2018. http://hdl.handle.net/11349/14225

Anexos.

Anexo A: Códigos Fuentes

A.1 Código en Arduino para la medición de la intensidad luminosa.

A continuación, se presenta el código correspondiente al Prototipo 1, el cual permite realizar la lectura de los divisores de voltaje, medir la temperatura y la humedad, así como almacenar los datos obtenidos en la memoria SD.

```
#include <SD.h>
#include <SPI.h>
#include <DHT.h>
#include <DHT_U.h>

#define sensor 7
#define SSpin 10

File archivo;
DHT dht(sensor, DHT11);

int i = 1;
float pv1 = 0, pv2 = 0, pv3 = 0;
float prt = 0, prh = 0;

void setup() {
  pinMode(A0, INPUT);
  pinMode(A1, INPUT);
  pinMode(A2, INPUT);
  Serial.begin(9600);

  dht.begin();

  Serial.println("Iniciando SD");
  if (!SD.begin(SSpin)) {
    Serial.println("Fallo SD");
    return;
  }
}

void loop() {
  for (int j = 1; j <= 180; j++) {
    float valor = analogRead(A0);
    float valor2 = analogRead(A1);
    float valor3 = analogRead(A2);
```

```
  float V = valor * (5.0 / 1023);
  float o = (V * 330) / (5.0 - V);
  float V2 = valor2 * (5.0 / 1023);
  float o2 = (V2 * 1000) / (5.0 - V2);
  float V3 = valor3 * (5.0 / 1023);
  float o3 = (V3 * 10000) / (5.0 - V3);

  if (o > 10000000) {
    o = 10000000;
  }

  if (o2 > 10000000) {
    o2 = 10000000;
  }

  if (o3 > 10000000) {
    o3 = 10000000;
  }

  float Vout = (5.0 * o) / (o + 330);
  float Vout2 = (5.0 * o2) / (o2 + 1000);
  float Vout3 = (5.0 * o3) / (o3 + 10000);

  float tmp = dht.readTemperature();
  float hum = dht.readHumidity();

  pv1 = pv1 + Vout;
  pv2 = pv2 + Vout2;
  pv3 = pv3 + Vout3;
  prt = prt + tmp;
  prh = prh + hum;

  delay(1000);
}

if (SD.begin(SSpin)) {
  archivo = SD.open("PRUEBA2.csv", FILE_WRITE);

  if (archivo) {

    float rv1 = pv1 / 180;
    float rv2 = pv2 / 180;
    float rv3 = pv3 / 180;
    float rvt = prt / 180;
```

```
float rvh = prh / 180;

float v1 = 5.0 - rv1;
float v2 = 5.0 - rv2;
float v3 = 5.0 - rv3;

archivo.print(i);
archivo.print(",");
archivo.print(v1, 4);
archivo.print(",");
archivo.print(v2, 4);
archivo.print(",");
archivo.print(v3, 4);
archivo.print(",");
archivo.print(rvt);
archivo.print(",");
archivo.println(rvh);

archivo.close();

i++;

Serial.print(" V(330): ");
Serial.print(v1, 4);
Serial.print("  V(1K): ");
Serial.print(v2, 4);
Serial.print("  V(10K): ");
Serial.println(v3, 4);
Serial.print(" Temperatura: ");
Serial.print(rvt);
Serial.print("°C   Humedad: ");
Serial.print(rvh);
Serial.println("%");

pv1 = 0;
pv2 = 0;
pv3 = 0;
prt = 0;
prh = 0;

rv1 = 0;
rv2 = 0;
rv3 = 0;
rvt = 0;
```

```
      rvh = 0;

    } else {
      Serial.println("no se encuentra el archivo");
      delay(1000);
    }
  } else {
    Serial.println("No se encuentra la tarjeta SD");
    delay(1000);
  }

  for (int k = 1; k <= 57; k++) {
    delay(60000);
    Serial.print("Minuto: ");
    Serial.println(k);
  }
}
```

El código completo se encuentra disponible en el siguiente enlace:
https://github.com/Italiatonik/IndiceUVAnexos/tree/main/Prototipo_1

A.2 Código en Arduino para la Estación de Monitoreo de Índice UV

A continuación, se presenta el código utilizado en la implementación del prototipo 2. Este código permite la lectura del índice UV, la lectura de la radiación visible, la medición de la temperatura y humedad, así como la escritura de los datos en la memoria SD.

```
#include <SD.h>
#include <SPI.h>
#include <DHT.h>
#include <Wire.h>
#include <RTClib.h>

#define sensor 8
#define SSpin 10

File archivo;
RTC_DS3231 rtc;
DHT dht(sensor, DHT22);

char bufferDatoFecha[19];
char bufferDatoHora[19];
int UVOUT = A2;
int REF_3V3 = A3;
```

```
int i = 0;

float pv1 = 0, puv = 0;
float prt = 0, prh = 0;
float rv1 = 0, rvt = 0, rvh = 0, ruv = 0;

void setup() {
  Wire.begin();
  rtc.begin();
  pinMode(A0, INPUT);
  pinMode(UVOUT, INPUT);
  pinMode(REF_3V3, INPUT);

  Serial.begin(9600);
  dht.begin();

  rtc.adjust(DateTime(F(__DATE__), F(__TIME__)));

  Serial.println("Iniciando SD");
  if (!SD.begin(SSpin)) {
    Serial.println("Fallo SD");
  }

  DateTime now = rtc.now();
  Serial.print(now.year(), DEC);
  Serial.print('/');
  Serial.print(now.month(), DEC);
  Serial.print('/');
  Serial.print(now.day(), DEC);
  Serial.print(" ");
  Serial.print(now.hour(), DEC);
  Serial.print(':');
  Serial.print(now.minute(), DEC);
  Serial.print(':');
  Serial.print(now.second(), DEC);
  Serial.println();

  pinMode(2, OUTPUT);
  pinMode(3, OUTPUT);
  pinMode(4, OUTPUT);
  pinMode(5, OUTPUT);
  pinMode(6, OUTPUT);
  pinMode(7, OUTPUT);
}
```

```
void loop() {
  if (SD.begin(SSpin)) {
    DateTime fecha = rtc.now();

    // Toma de datos cada 10 minutos
    if (fecha.minute() == 7 || fecha.minute() == 17 || fecha.minute() == 27 ||
        fecha.minute() == 37 || fecha.minute() == 47 || fecha.minute() == 57) {

      limpiar();

      for (int j = 1; j <= 180; j++) {
        float valor = analogRead(A0);
        float V = valor * (5.0 / 1023);
        float o = (V * 330) / (5.0 - V);

        if (o > 10000000) {
          o = 10000000;
        }

        float Vout = (5.0 * o) / (o + 330);

        float tmp = dht.readTemperature();
        float hum = dht.readHumidity();

        int uvLevel = averageAnalogRead(UVOUT);
        int refLevel = averageAnalogRead(REF_3V3);
        float outputVoltage = 3.33 / refLevel * uvLevel;

        float uvIntensity = mapfloat(outputVoltage, 0.98, 2.9, 0.0, 15.0);

        puv += uvIntensity;
        pv1 += Vout;
        prt += tmp;
        prh += hum;

        delay(1000);
      }

      rv1 = pv1 / 180;
      rvt = prt / 180;
      rvh = prh / 180;
      ruv = puv / 180;
```

```
      i = 1;

    } else {
      if (i == 1) {
        archivo = SD.open("PRUEBA3.csv", FILE_WRITE);

        if (archivo) {
          float valor = analogRead(A0);
          float V = valor * (5.0 / 1023);
          float o = (V * 330) / (5.0 - V);
          float Vout = (5.0 * o) / (o + 330);
          float v1 = 5.0 - rv1;

          sprintf(bufferDatoFecha, "%02d/%02d/%04d", fecha.day(), fecha.month(),
fecha.year());
          sprintf(bufferDatoHora, "%02d:%02d", fecha.hour(), fecha.minute());

          archivo.print(bufferDatoFecha);
          archivo.print(",");
          archivo.print(bufferDatoHora);
          archivo.print(",");
          archivo.print(round(ruv));
          archivo.print(",");
          archivo.print(ruv, 2);
          archivo.print(",");
          archivo.print(v1, 4);
          archivo.print(",");
          archivo.print(rvt);
          archivo.print(",");
          archivo.println(rvh);

          archivo.close();

          semaforo();
          limpiar();
          i = 0;

        } else {
          Serial.println("No se encuentra el archivo");
        }
      }
    }
  } else {
    Serial.println("No se encuentra la tarjeta SD");
```

```
    delay(1000);
  }
}

void limpiar() {
  pv1 = 0;
  prt = 0;
  prh = 0;
  puv = 0;
  rv1 = 0;
  rvt = 0;
  rvh = 0;
  ruv = 0;
}

void semaforo() {
  if (ruv >= 0 && ruv < 1) {
    Serial.println("Nula");
    digitalWrite(7, HIGH);
    digitalWrite(6, LOW);
    digitalWrite(5, LOW);
    digitalWrite(4, LOW);
    digitalWrite(3, LOW);
    digitalWrite(2, LOW);
  } else if (ruv >= 1 && ruv < 3) {
    Serial.println("Baja");
    digitalWrite(7, LOW);
    digitalWrite(6, HIGH);
    digitalWrite(5, LOW);
    digitalWrite(4, LOW);
    digitalWrite(3, LOW);
    digitalWrite(2, LOW);
  } else if (ruv >= 3 && ruv < 6) {
    Serial.println("Moderada");
    digitalWrite(7, LOW);
    digitalWrite(6, LOW);
    digitalWrite(5, HIGH);
    digitalWrite(4, LOW);
    digitalWrite(3, LOW);
    digitalWrite(2, LOW);
  } else if (ruv >= 6 && ruv < 8) {
    Serial.println("Alta");
    digitalWrite(7, LOW);
    digitalWrite(6, LOW);
```

```
    digitalWrite(5, LOW);
    digitalWrite(4, HIGH);
    digitalWrite(3, LOW);
    digitalWrite(2, LOW);
  } else if (ruv >= 8 && ruv < 11) {
    Serial.println("Muy alta");
    digitalWrite(7, LOW);
    digitalWrite(6, LOW);
    digitalWrite(5, LOW);
    digitalWrite(4, LOW);
    digitalWrite(3, HIGH);
    digitalWrite(2, LOW);
  } else if (ruv >= 11) {
    Serial.println("Extremadamente alta");
    digitalWrite(7, LOW);
    digitalWrite(6, LOW);
    digitalWrite(5, LOW);
    digitalWrite(4, LOW);
    digitalWrite(3, LOW);
    digitalWrite(2, HIGH);
  }
}

int averageAnalogRead(int pinToRead) {
  byte numberOfReadings = 8;
  unsigned int runningValue = 0;

  for (int x = 0; x < numberOfReadings; x++)
    runningValue += analogRead(pinToRead);

  runningValue /= numberOfReadings;
  return (runningValue);
}

float mapfloat(float x, float in_min, float in_max, float out_min, float out_max) {
  return (x - in_min) * (out_max - out_min) / (in_max - in_min) + out_min;
}
```

El código completo se encuentra disponible en el siguiente enlace:
https://github.com/Italiatonik/IndiceUVAnexos/tree/main/Prototipo_2

A.3 Procedimiento de Extracción de Datos

Para la extracción de los datos se utilizó la fórmula descrita en el **prototipo 2** en donde primero es tomar 4 puntos (2 puntos en la parte lineal y 2 de la parte exponencial) con su valor en el *eje x* y el *eje y* del amanecer y del atardecer. En la Tabla 3 se muestran los puntos que se seleccionaron para la extracción de los parámetros.

Tabla 3.- Puntos tomados para calcular las variables.

	Amanecer		Atardecer	
Punto	x	y	x	y
1	0.0329	0.65	0.0586	0.55
2	1.8339	1.11	1.474	0.87
3	3.1478	4.86	3.1425	5.37
4	2.8579	2.22	2.8696	2.76

Como ejemplo del proceso solo se mostrará la extracción de los parámetros del amanecer. Lo primero fue despejar la variable “a”, esto tomando la mitad de la formula y la escribimos dos veces.

$$y1 = ax1 + b$$
$$y2 = ax2 + b$$

Multiplicamos la primera por -1 esto para eliminar la variable b y queda de la siguiente manera

$$-y1 = -ax1 - b$$
$$y2 = \quad ax2 + b$$

Una vez restamos ambas formulas nos queda así

$$y2 - y1 = ax2 - ax1$$

Factorizamos a y después despejamos a

$$y2 - y1 = a(x2 - x1)$$

Y como resultado tenemos el despeje de a

$$a = (y2 - y1)/(x2 - x1)$$

Sustituimos los valores de y2, y1, x2 y x1

$$a = (1.11 - 0.65)/(1.8339 - 0.0329)$$

Y obtenemos el valor de a

$$a = \ 0.2554$$

Una vez obtenido el valor de a la sustituimos junto con "x" y "y" en la formula y despejamos b

$$0.65 = 0.2554(0.0329) + b$$
$$b = 0.65 - (0.2554 * 0.0329)$$

Y tenemos el valor b

$$b = 0.6416$$

Ya tenemos los valores de a y b lo siguiente es sustituir estos valores junto con "x" y "y" de los puntos de la exponencial para poder despejar c y d. a continuación se describe el despeje de c de la siguiente formula.

$$y = ax + b + ce^{dx}$$

Primero escribimos la formula dos veces una para cada punto de la parte exponencial y sustituimos los valores que conocemos

$$4.86 = 0.2554(3.1478) + 0.6416 + ce^{d(3.1478)}$$
$$2.22 = 0.2554(2.8576) + 0.6416 + ce^{d(2.8576)}$$

Realizamos las operaciones

$$ce^{d(3.1478)} = 3.4145$$

$$ce^{d(2.8576)} = 0.8486$$

Despejamos c de ambas ecuaciones

$$c = \frac{3.4145}{e^{d(3.1478)}}$$

$$c = \frac{0.8486}{e^{d(2.8576)}}$$

Lo siguiente es despejar d, para esto podemos igualar la variable c

$$\frac{3.4145}{e^{d(3.1478)}} = \frac{0.8486}{e^{d(2.8576)}}$$

Una vez igualadas movemos la variable d a un lado y queda de la siguiente manera

$$\frac{e^{d(2.8576)}}{1}\left(\frac{3.4145}{e^{d(3.1478)}}\right) = 0.8486$$

$$\frac{e^{d(2.8576)}(3.4145)}{e^{d(3.1478)}} = 0.8486$$

$$\frac{e^{d(2.8576)}}{e^{d(3.1478)}} = \frac{0.8486}{3.4145}$$

Resolvemos la operación utilizando la ley de los logaritmos donde

$$\log\frac{a}{b} = \log a - \log b$$

Entonces queda de la siguiente manera

$$e^{d(2.8576)-d(3.1478)} = 0.2485$$

Factorizamos d y realizamos la operación

$$e^{d(2.8576-3.1478)} = 0.2485$$

$$e^{d(-0.2902)} = 0.2485$$

Lo siguiente es igual las bases en este caso lo haremos con un logaritmo natural y resolvemos

$$\ln e^{d(2.8576-3.1478)} = \ln 0.2485$$

$$d(2.8576 - 3.1478) = -1.3923$$

Resolvemos las operaciones y despejamos d

$$d = \frac{-1.3923}{-0.2902}$$

Y obtenemos el valor de d

$$d = 4.7977$$

Ahora sustituimos el valor de d en una de las fórmulas de c y realizamos las operaciones

$$c = \frac{3.4145}{e^{(4.7977*3.1478)}}$$

$$c = \frac{3.4145}{e^{(15.1022)}}$$

Obtenemos el valor de c

$$c = 9.4302e^{-7}$$

Con esto ya obtenemos los valores de a, b, c y d. Ahora repetiremos el mismo proceso para extraer las variables, pero con los datos del atardecer.

A.4 Código en Python de la red neuronal.

A continuación, se presenta el código utilizado en la implementación del prototipo 3. Este código permite la creación y entrenamiento de la red neuronal artificial.

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras import optimizers, backend
from sklearn.model_selection import train_test_split
from google.colab import drive

dataset = pd.read_csv('datos.csv')

dataset.head()

plt.plot(dataset['Vout330'])
plt.show()

#----------------------------------Nuevo Bloque----------------------------------#
dataset['Hora'] = pd.to_datetime(dataset['Hora'], format='%H:%M').dt.hour * 60 +
pd.to_datetime(dataset['Hora'], format='%H:%M').dt.minute

caracteristicas_estandarizar = ['Hora', 'Vout330','Indice UV float']

medias = []
desviaciones = []
for caracteristica in caracteristicas_estandarizar:
    mean = dataset[caracteristica].mean()
    std = dataset[caracteristica].std()
```

```
    dataset[caracteristica] = (dataset[caracteristica] - mean)/std
    medias.append(mean)
    desviaciones.append(std)

for i, caracteristica in enumerate(caracteristicas_estandarizar):
    print(caracteristica)
    print(medias[i])
    print(desviaciones[i])

dataset.head()

#----------------------------------Nuevo Bloque----------------------------------#
caracteristicas = dataset[['Hora','Vout330']]
targets = dataset[['Indice UV float']]

targets

#----------------------------------Nuevo Bloque----------------------------------#
x_train, x_test, y_train, y_test = train_test_split(caracteristicas, targets, test_size=0.20)

print(x_train.shape)
print(y_train.shape)
print(x_test.shape)
print(y_test.shape)

x_train = np.asarray(x_train).astype(np.float32)
y_train = np.asarray(y_train).astype(np.float32)
x_test = np.asarray(x_test).astype(np.float32)
y_test = np.asarray(y_test).astype(np.float32)

#----------------------------------Nuevo Bloque----------------------------------#
backend.clear_session()

modelo = Sequential()
modelo.add(Dense(5, activation='sigmoid', use_bias = True, input_shape = (2,)))
modelo.add(Dense(3, activation='sigmoid'))
modelo.add(Dense(1, activation='linear'))

modelo.summary()

adam = optimizers.Adam(learning_rate = 0.1)
modelo.compile(loss='mean_squared_error', optimizer=adam, metrics=['mse'])
```

```
M = modelo.fit(x_train, y_train, epochs=300, verbose=1, validation_data=(x_test, y_test), batch_size=100,shuffle=True)

#----------------------------------Nuevo Bloque----------------------------------#
plt.plot(M.history['mse'], label='entrenamiento')
plt.plot(M.history['val_mse'], label='validacion')
plt.xlabel('epocas')
plt.ylabel('mse')
plt.legend(loc='upper right')
plt.show()

#----------------------------------Nuevo Bloque----------------------------------#
for i, layer in enumerate(modelo.layers):
    weights, biases = layer.get_weights()
    print(weights)
    print(biases)

modelo.save_weights('pesos_modelo.weights.h5')
```

El código completo se encuentra disponible en el siguiente enlace: https://github.com/Italiatonik/IndiceUVAnexos/tree/main/Prototipo_3

A.5 Despeje de la variable X en la formulade Z-score.

$$z = \frac{x - \mu}{\sigma}$$

$$z * \sigma = x - \mu$$

$$x = (z * \sigma) + \mu$$

A.6 Código en Arduino para la implementación de la RNA

A continuación, se presenta el código utilizado en la implementación del prototipo 4. Este código permite la lectura del índice UV, la lectura de la radiación visible, la estructura de la RNA y la salida de ambas representadas con los LEDs RGB.

```
#include <RTClib.h>
#include <Wire.h>

RTC_DS3231 rtc;
```

```
const int Rf  = 6;
const int Gf  = 5;
const int Bf  = 3;

const int Ru = 11;
const int Gu = 10;
const int Bu = 9;

float uv;
int UVOUT = A3;
int REF_3V3 = A1;

//------------------- Parámetros de la RNA --------------------//

double w_Capa_oculta1[2][5] = {
  { 2.8060021, -1.1711409, 4.122045, 2.350187, 0.81597537 },
  { 2.3484783, 2.2630205, -3.2768, 2.1375208, -2.6784663 }
};

double b_Capa_oculta1[5] = {
  -1.4936172, -1.3820752, -0.48103565, -1.0889859, 2.4271436
};

double w_Capa_oculta2[5][3] = {
  { 2.4358888, -1.8404518, -1.6751294 },
  { 1.1724445, -0.6933083, -1.5696467 },
  { -2.957599, 1.1990123, 1.3012772 },
  { 1.4084017, -1.1040592, -1.2870497 },
  { -2.26612, 2.7496896, -0.6572841 }
};

double b_Capa_oculta2[3] = { -0.9334416, 0.62091994, -0.20813814 };
double w_Capa_salida[3] = { 2.4138823, -1.6681055, -0.36259282 };
double b_Capa_salida = 0.6181946;

double mediaVout = 2.2540182926829266;
double mediaHora = 755.0;
double mediaUV = 2.4275609756097563;

double desviacionVout = 1.0734817809845172;
double desviacionHora = 238.15261213488014;
double desviacionUV = 1.593268680677765;
```

```
//------------------- Funciones de activación -------------------//

double sigmoid(double x) {
  return 1.0 / (1.0 + exp(-x));
}

double linear(double x) {
  return x;
}

//-----------------------------------------------------------------//
void setup() {
  Serial.begin(9600);
  rtc.begin();

  rtc.adjust(DateTime(F(__DATE__), F(__TIME__)));

  DateTime now = rtc.now();
  Serial.print(now.year(), DEC); Serial.print('/');
  Serial.print(now.month(), DEC); Serial.print('/');
  Serial.print(now.day(), DEC); Serial.print(" ");
  Serial.print(now.hour(), DEC); Serial.print(':');
  Serial.print(now.minute(), DEC); Serial.print(':');
  Serial.println(now.second(), DEC);

  pinMode(A0, INPUT);
  pinMode(A1, INPUT);
  pinMode(A3, INPUT);
  pinMode(A4, INPUT);
  pinMode(A5, INPUT);
  pinMode(Rf, OUTPUT);
  pinMode(Gf, OUTPUT);
  pinMode(Bf, OUTPUT);
  pinMode(Ru, OUTPUT);
  pinMode(Gu, OUTPUT);
  pinMode(Bu, OUTPUT);
}

void loop() {
  DateTime fecha = rtc.now();
  float valor = analogRead(A0);

  int hora = fecha.hour();
  int minutos = fecha.minute();
```

```
int HoraTotal = hora * 60 + minutos;

double vout330 = 5.0 - (valor * 5.0 / 1023.0);

double horaEstandar = (HoraTotal - mediaHora) / desviacionHora;
double VoutEstandar = (vout330 - mediaVout) / desviacionVout;
double indexUV = IndiceUV(horaEstandar, VoutEstandar);
double indiceUVDes = indexUV * desviacionUV + mediaUV;


// Sensor UV ML8511
float uvLevel = analogRead(UVOUT);
float refLevel = analogRead(REF_3V3);
float outputVoltage = uvLevel * 3.3 / refLevel;
float uvIntensity = mapfloat(outputVoltage, 0.99, 2.9, 0.0, 15.0);
uv = uvIntensity;

if (uv < 0) {
  uv = 0;
}

// LED RNA
if (indiceUVDes < 2.0) {
  setColor(255-0, 255-255, 255-0); // Verde
} else if (indiceUVDes < 6.0) {
  setColor(255-255, 255-255, 255-0); // Amarillo
} else if (indiceUVDes < 8.0) {
  setColor(255-255, 255-165, 255-0); // Naranja
} else if (indiceUVDes < 11.0) {
  setColor(255-255, 255-0, 255-0); // Rojo
} else {
  setColor(255-128, 255-0, 255-128); // Morado
}

if (uv < 2.0) {
  setColorUV(255-0, 255-255, 255-0); // Verde
} else if (uv < 6.0) {
  setColorUV(255-255, 255-255, 255-0); // Amarillo
} else if (uv < 8.0) {
  setColorUV(255-255, 255-165, 255-0); // Naranja
} else if (uv < 11.0) {
  setColorUV(255-255, 255-0, 255-0); // Rojo
} else {
  setColorUV(255-128, 255-0, 255-128); // Morado
```

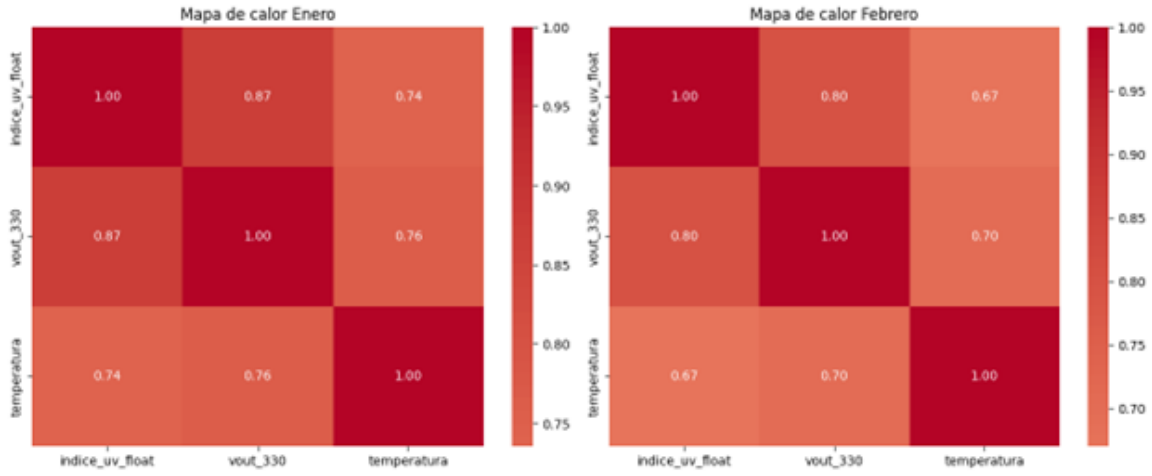
```
  }

  Serial.print("Hora: ");
  Serial.print(hora); Serial.print(':'); Serial.print(minutos);
  Serial.print(" | Índice UV (RNA): "); Serial.print(indiceUVDes);
  Serial.print(" | Índice UV (ML8511): ");
  Serial.println(uv, 2);


  /*Serial.println("");
  Serial.println(valor);
  Serial.println(vout330);
  Serial.println(outputVoltage);*/

  delay(32000);
}

void setColorUV(int red, int green, int blue) {
  analogWrite(Ru, red);
  analogWrite(Gu, green);
  analogWrite(Bu, blue);
}

void setColor(int red, int green, int blue) {
  analogWrite(Rf, red);
  analogWrite(Gf, green);
  analogWrite(Bf, blue);
}

double IndiceUV(double a, double b) {
  double salida_oculta1[5];
  double salida_oculta2[3];
  double sum = 0;

  for (int i = 0; i < 5; i++) {
    salida_oculta1[i] = a * w_Capa_oculta1[0][i] + b * w_Capa_oculta1[1][i] +
b_Capa_oculta1[i];
    salida_oculta1[i] = sigmoid(salida_oculta1[i]);
  }

  for (int i = 0; i < 3; i++) {
    salida_oculta2[i] = 0;
    for (int j = 0; j < 5; j++) {
      salida_oculta2[i] += salida_oculta1[j] * w_Capa_oculta2[j][i];
```

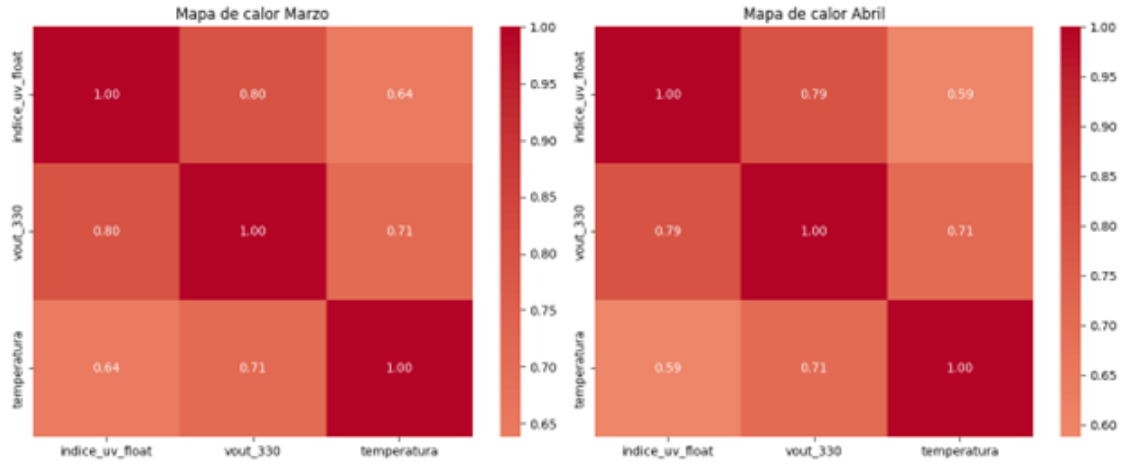
```
    }
    salida_oculta2[i] += b_Capa_oculta2[i];
    salida_oculta2[i] = sigmoid(salida_oculta2[i]);
  }

  for (int i = 0; i < 3; i++) {
    sum += salida_oculta2[i] * w_Capa_salida[i];
  }

  return linear(sum + b_Capa_salida);
}

int averageAnalogRead(int pinToRead) {
  byte numberOfReadings = 8;
  unsigned int runningValue = 0;
  for (int x = 0; x < numberOfReadings; x++)
    runningValue += analogRead(pinToRead);
  return runningValue / numberOfReadings;
}

float mapfloat(float x, float in_min, float in_max, float out_min, float out_max) {
  return (x - in_min) * (out_max - out_min) / (in_max - in_min) + out_min;
}
```

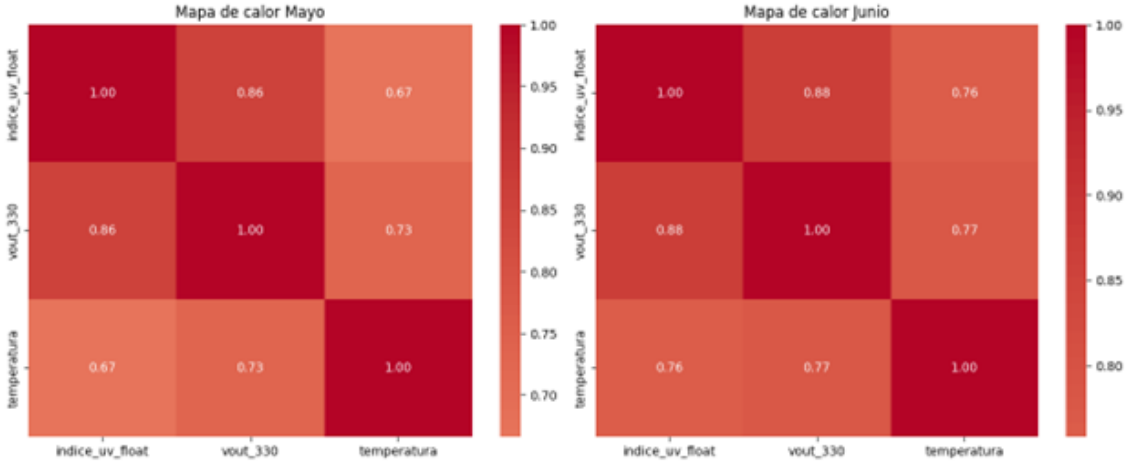
El código completo se encuentra disponible en el siguiente enlace: https://github.com/Italiatonik/IndiceUVAnexos/tree/main/Prototipo_4

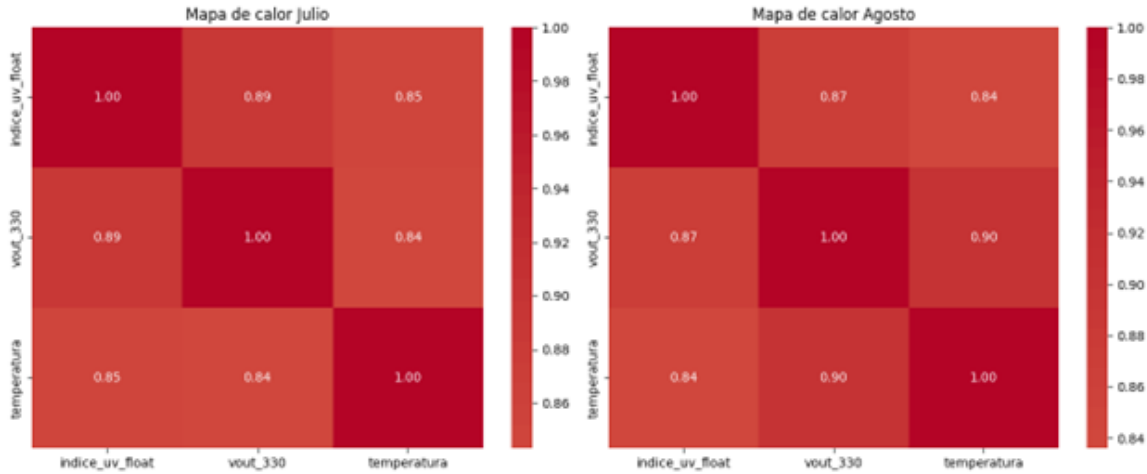
A.7 Mapas de calor de cada mes.

A continuación, se muestra en la Fig. 64, la correlación del índice UV con las variables de temperatura y del voltaje de salida del divisor con respecto a cada mes y de todo el conjunto de datos.




Mapa de calor Marzo
indice_uv_float
vout_330
temperatura
1.00
0.80
0.64
0.80
1.00
0.71
0.64
0.71
1.00
1.00
0.95
0.90
0.85
0.80
0.75
0.70
0.65
Mapa de calor Abril
indice_uv_float
vout_330
temperatura
1.00
0.79
0.59
0.79
1.00
0.71
0.59
0.71
1.00
1.00
0.95
0.90
0.85
0.80
0.75
0.70
0.65
0.60


Mapa de calor Mayo
indice_uv_float
vout_330
temperatura
1.00
0.86
0.67
0.86
1.00
0.73
0.67
0.73
1.00
1.00
0.95
0.90
0.85
0.80
0.75
0.70
Mapa de calor Junio
indice_uv_float
vout_330
temperatura
1.00
0.88
0.76
0.88
1.00
0.77
0.76
0.77
1.00
1.00
0.95
0.90
0.85
0.80


Mapa de calor Julio
indice_uv_float
vout_330
temperatura
1.00
0.89
0.85
0.89
1.00
0.84
0.85
0.84
1.00
1.00
0.98
0.96
0.94
0.92
0.90
0.88
0.86
Mapa de calor Agosto
indice_uv_float
vout_330
temperatura
1.00
0.87
0.84
0.87
1.00
0.90
0.84
0.90
1.00
1.00
0.98
0.96
0.94
0.92
0.90
0.88
0.86
0.84

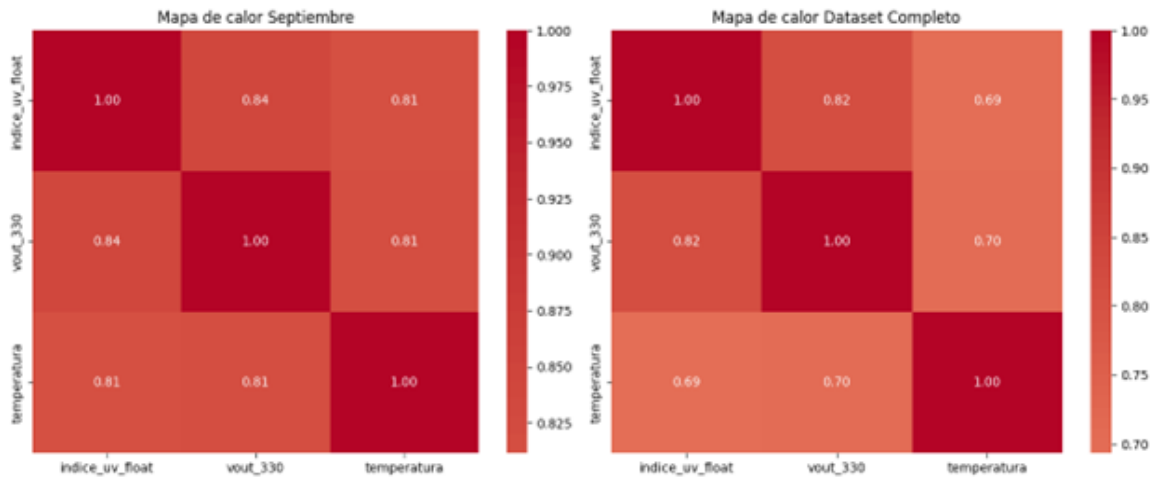


Fig. 64.- Correlación entre variables.

Anexo B: Instalación de Librerías en Arduino IDE (Opcional)

Para la correcta ejecución del código en los prototipos desarrollados, es necesario instalar las siguientes librerías en el entorno Arduino IDE, Estas librerías nos permiten la comunicación con los sensores y placas que se utilizaron en este proyecto.

B.1 Instalación de la librería DHT11 y DHT22

1. Abrimos Arduino IDE y nos dirigimos a la pestaña Programa > Incluir biblioteca > Administrar bibliotecas….
2. En la barra de búsqueda, escribimos el nombre de la librería deseada (Fig. X).
3. Seleccionamos la opción correspondiente y hacemos clic en Instalar.
4. Esperamos a que la instalación finalice y verificamos que la librería esté disponible en el menú de Incluir biblioteca (ver Fig. 65).

Fig. 65.- Librería DHT Sensor

B.2 Instalación de la librería RTClib

1. Abrimos Arduino IDE y nos dirigimos a la pestaña Programa > Incluir biblioteca > Administrar bibliotecas….
2. En la barra de búsqueda, escribimos el nombre de la librería deseada (Fig. X).
3. Seleccionamos la opción correspondiente y hacemos clic en Instalar.
4. Esperamos a que la instalación finalice y verificamos que la librería esté disponible en el menú de Incluir biblioteca (Ver Fig. 66).

Fig. 66.- Librería RTClib.

Jonathan Romero Rodriguez

Red neuronal artificial para estimar el índice UV

Universidad Autónoma del Estado de México

Detalles del documento

Identificador de la entrega
trn:oid:::3117:530746638

Fecha de entrega
20 nov 2025, 12:24 p.m. GMT-6

Fecha de descarga
14 ene 2026, 12:25 p.m. GMT-6

Nombre del archivo
Tesis JRR.docx

Tamaño del archivo
7.4 MB

102 páginas

17.840 palabras

94.071 caracteres

2% Similitud general

El total combinado de todas las coincidencias, incluidas las fuentes superpuestas, para ca...

Filtrado desde el informe

- Bibliografía
- Coincidencias menores (menos de 15 palabras)

Fuentes principales

2% Fuentes de Internet
1% Publicaciones
0% Trabajos entregados (trabajos del estudiante)

Marcas de integridad

N.º de alerta de integridad para revisión

Caracteres reemplazados
61 caracteres sospechosos en N.º de páginas
Las letras son intercambiadas por caracteres similares de otro alfabeto.

Los algoritmos de nuestro sistema analizan un documento en profundidad para buscar inconsistencias que permitirían distinguirlo de una entrega normal. Si advertimos algo extraño, lo marcamos como una alerta para que pueda revisarlo.

Una marca de alerta no es necesariamente un indicador de problemas. Sin embargo, recomendamos que preste atención y la revise.