UNIVERSIDAD CATÓLICA DE CUENCA

UNIVERSIDAD CATÓLICA DE CUENCA

Comunidad Educativa al Servicio del Pueblo

UNIDAD ACADÉMICA DE INGENIERIA, INDUSTRIA Y CONSTRUCCIÓN

CARRERA DE INGENIERÍA ELÉCTRICA

ESTUDIO PARA INCORPORACIÓN DE INTELIGENCIA ARTIFICIAL EN CIRCUITOS DE TRANSFORMADORES DE DISTRIBUCIÓN PARA ANÁLISIS DE DEMANDA, ENLACE Y ANÁLISIS DE INFORMACIÓN

PROYECTO DE TITULACIÓN PREVIO A LA OBTENCIÓN DEL TÍTULO DE INGENIERA ELÉCTRICA

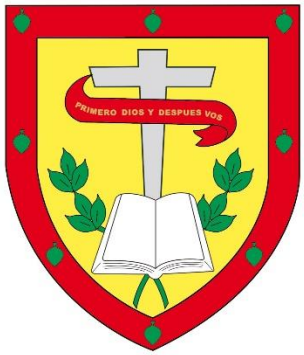
AUTOR: DIANA DENNISE MOLINA FÁREZ

DIRECTOR: ING. OSCAR MAURICIO SIGUENCIA SIGUENZA.MGS

CUENCA - ECUADOR

2023

DIOS, PATRIA, CULTURA Y DESARROLLO



UNIVERSIDAD CATÓLICA DE CUENCA

Comunidad Educativa al Servicio del Pueblo

UNIDAD ACADÉMICA DE INGENIERÍA, INDUSTRIA Y CONSTRUCCIÓN

CARRERA DE INGENIERÍA ELÉCTRICA

ESTUDIO PARA INCORPORACIÓN DE INTELIGENCIA ARTIFICIAL EN CIRCUITOS DE TRANSFORMADORES DE DISTRIBUCIÓN PARA ANÁLISIS DE DEMANDA, ENLACE Y ANÁLISIS DE INFORMACIÓN

PROYECTO DE TITULACIÓN PREVIO A LA OBTENCIÓN DEL TÍTULO DE INGENIERA ELÉCTRICA

AUTOR: DIANA DENNISE MOLINA FAREZ

DIRECTOR: ING. OSCAR MAURICIO SIGUENCIA SIGUENZA. MGS

CUENCA - ECUADOR

2023

DIOS, PATRIA, CULTURA Y DESARROLLO

DECLARATORIA DE AUTORÍA Y RESPONSABILIDAD

Diana Dennise Molina Farez portador de la cédula de ciudadanía Nº 1400770127. Declaro ser el autor de la obra: “Estudio para incorporación de inteligencia artificial en circuitos de transformadores de distribución para análisis de demanda, enlace y análisis de información”, sobre la cual me hago responsable sobre las opiniones, versiones e ideas expresadas. Declaro que la misma ha sido elaborada respetando los derechos de propiedad intelectual de terceros y eximo a la Universidad Católica de Cuenca sobre cualquier reclamación que pudiera existir al respecto. Declaro finalmente que mi obra ha sido realizada cumpliendo con todos los requisitos legales, éticos y bioéticos de investigación, que la misma no incumple con la normativa nacional e internacional en el área específica de investigación, sobre la que también me responsabilizo y eximo a la Universidad Católica de Cuenca de toda reclamación al respecto.

Cuenca, 07 de noviembre de 2023

F: ……..

Diana Dennise Molina Farez

1400770127

CERTIFICACIÓN

Certifico que el presente trabajo fue desarrollado por la Srta. Diana Dennise Molina Fárez, bajo mi supervisión.

Ing. Oscar Mauricio Siguencia. Mgs

Docente tutor del trabajo de titulación

DEDICATORIA

Quiero expresar mi gratitud a Dios, quien con su bendición llena mi vida. Este trabajo de titulación quiero dedicarlo principalmente a mi padre, por haberme brindado su apoyo incondicional, quien con su amor, paciencia y esfuerzo me ha permitido cumplir una de mis metas, gracias por inculcar en mí el ejemplo de esfuerzo y valentía, de no temer las adversidades porque siempre estoy bendecida de Dios.

Quiero dedicar también este trabajo a toda mi familia en especial a mi abuelita Elsita quien, con sus oraciones, consejos me animo a seguir de pie ante todas las adversidades, gracias a su apoyo en cualquier momento de mi carrera universitaria. A mi hermana Amy Molina por su cariño y apoyo incondicional, a mi amigo Christian Torres y a mis amigas Brigitte Fárez, Salome Quizhpi, Elizabeth Salazar por su cariño y apoyo durante este proceso, por estar conmigo en todo momento.

Gracias a todas las personas presentes en mi vida quienes me ayudaron a ser una mejor persona y de una u otra forma me acompañan en mis metas por cumplir a los que llamo yo mis más grandes sueños.

AGRADECIMIENTOS

Quiero expresar mi agradecimiento a los directivos y personal de la Empresa Eléctrica Azogues por confiar en mí y permitirme realizar todo el proceso investigativo dentro de su establecimiento. De igual manera agradecer a la Universidad Católica de Cuenca y a la Unidad Académica de Ingeniería, Industria y Construcción, al personal docente en general quienes con su valioso conocimiento me han ayudado a crecer como profesional día a día.

Así mismo quiero expresar mi más profundo agradecimiento al Ing. Oscar Mauricio Siguencia Siguenza, quien ha sido mi principal colaborador durante todo este proceso y cuya dirección, conocimiento, enseñanza han sido fundamental para el desarrollo de este trabajo de titulación.

RESUMEN

El presente trabajo se ha investigado la necesidad de desarrollar un modelo de predicción de imágenes térmicas que permitan clasificar las imágenes según su grado de severidad de degradación térmica. Se ha destacado la importancia de esta clasificación para la detección temprana de problemas y la toma de decisiones en diversos campos.

La metodología utilizada en este estudio se basó en la recopilación de un conjunto de datos de imágenes térmicas con diferentes grados de degradación térmica. Se aplicaron técnicas de procesamiento de imágenes y se utilizó un algoritmo de aprendizaje profundo para entrenar y evaluar el modelo de predicción.

Los resultados obtenidos mostraron que el modelo de predicción desarrollado fue capaz de clasificar imágenes térmicas según su grado de severidad de degradación térmica con una precisión aceptable. Se observará una evaluación significativa entre las características térmicas de las imágenes y el grado de degradación térmica.

En conclusión, este estudio demostró la viabilidad de desarrollar un modelo de predicción de imágenes térmicas para clasificar el grado de severidad de degradación térmica. Sin embargo, se identificaron algunas limitaciones, como la necesidad de un conjunto de datos más diversos y la exploración de técnicas de procesamiento de imágenes más avanzadas.

Se recomienda continuar investigando y mejorando el modelo de predicción, incluyendo la recopilación de un conjunto de datos más amplio y diverso. Además, se sugiere explorar técnicas de procesamiento de imágenes más avanzadas y considerar la integración de otras variables relevantes para mejorar la precisión del modelo.

Palabras clave: transformadores de distribución, degradación térmica, análisis termográfico, red neuronal convolucional, Aprendizaje profundo

ABSTRACT

This research focuses on the development of a thermal image prediction model that classifies images according to their degree of thermal degradation severity. The significance of this classification has been highlighted for the early detection of problems and decision-making in several fields.

The methodology used in this study was based on collecting a dataset of thermal images with different degrees of thermal degradation. Image processing techniques were applied, and a deep learning algorithm was used to train and evaluate the prediction model.

The results obtained showed that the developed prediction model was capable of classifying thermal images according to their degree of thermal degradation severity with an acceptable accuracy. A substantial evaluation was observed between the thermal characteristics of the images and the degree of thermal degradation.

In conclusion, this study demonstrated the feasibility of developing a thermal image prediction model for classifying the severity of thermal degradation. However, some limitations were identified, including the need for a more diverse dataset and the exploration of more advanced image processing techniques.

It is recommended to continue research and improve the prediction model, which includes collecting a more extensive and diverse dataset. Furthermore, it is suggested to explore more advanced image processing techniques and consider the integration of other relevant variables to improve the accuracy of the model.

Keywords: distribution transformers, thermal degradation, thermographic analysis, convolutional neural network, Deep learning

INDICE DE CONTENIDO

LISTA DE FIGURAS

LISTA DE TABLAS

LISTA DE ANEXOS

CAPÍTULO I

1. INTRODUCCIÓN

La inteligencia artificial ha experimentado un gran avance en los últimos tiempos, ya sea por su aplicación en diferentes ámbitos profesionales como en el sector de la salud, las finanzas, el marketing, la logística, el transporte o el sector eléctrico, en ámbitos recreativos como en sistemas que generan imágenes, a través de texto, editores de video automatizado, o en los usos más cotidianos como puede ser los asistentes personales entre otros ámbitos, la inteligencia artificial es aplicada cada vez en sectores cada uno totalmente diferente del otro (Llanos, 2023).

La aplicación de la inteligencia artificial en el sector eléctrico ha abierto nuevas oportunidades para mejorar la eficiencia, la seguridad y la sostenibilidad de la generación, transmisión y distribución de energía eléctrica. La investigación en el ámbito internacional y nacional ha desempeñado un papel fundamental en el avance del conocimiento en diversos campos. Estas investigaciones han contribuido significativamente al desarrollo de teorías, metodologías y prácticas en sus diferentes disciplinas. A nivel nacional, también se ha llevado a cabo a una importante investigación laboral, donde se han abordado problemáticas específicas y se han propuesto soluciones adaptadas al contexto local. Estas investigaciones a nivel nacional han permitido identificar desafíos y oportunidades únicas, generando conocimiento valioso que puede ser aplicado para abordar áreas problemáticas y contribuir al progreso de la sociedad. Se exploran algunas de las investigaciones y tesis destacadas a nivel internacional y nacional, con el objeto de comprender la relevancia y el impacto de estas investigaciones en sus respectivos campos.

A continuación, se describe algunas aplicaciones de la inteligencia artificial en el sector eléctrico como por ejemplo una investigación desarrollada en Piura, en la cual incluyeron Deep Learning, con la finalidad de realizar una clasificación binaria de imágenes térmicas de líneas y subestaciones eléctricas (Aguilar, 2022). De igual manera, (Jafrouni, Elbreki, Almaktar, Zakariya, & Mohamed, 2023) utilizan el programa de Electrolizador Transitorio (ETAP) para simular los diferentes sistemas de energía. De igual manera, (Kicheriavenkov, 2021) incorpora redes neuronales entrenadas en un análogo de funcionamiento en redes de distribución para desviaciones típicas de las fallas a tierra monofásicas o fugas a tierra debido a la disminución del nivel de aislamiento.

Del mismo modo, (Alava, 2022) presenta el diagnostico visual-térmico en sistemas eléctrico de subtransmisión y distribución con el uso de drones para efectuar mantenimientos en la cual incluyeron un software de detección encargado de evaluar las tomas y videos que

el dron capta en campo con cámaras termográficas. El estudio de (Jiménez León & Maza Pinza, 2008) presentan la implementación de un sistema basado en inteligencia artificial para la detección de regímenes anormales en transformadores de potencia, mediante redes neuronales artificiales creadas, entrenadas y simuladas en Matlab e implementadas en LabVIEW con el uso de tarjetas de adquisición de datos. Otro estudio considera un enfoque con la finalidad de detectar fallas en los circuitos de transformadores de distribución a las líneas de baja tensión (Cadme, 2022), para lo cual se apoyan en imágenes infrarrojas obteniendo un análisis mediante dos redes neuronales convolucionales.

Según (Mengyao, 2021), menciona que la aplicación de tecnología de inteligencia artificial al control de automatización eléctrica puede promover eficazmente el rápido desarrollo del control de la automatización de la ingeniería eléctrica.

Estas investigaciones entrenan determinados modelos de manera tradicional, sin embargo, en la actualidad se cuenta con distintas técnicas que contribuyen a mejorar la calidad de los entrenamientos y reducir los tiempos de ejecución.

1.1 Formulación del problema

La energía eléctrica, se ha convertido en una necesidad, dado que se utiliza en una amplia variedad de dispositivos y aplicaciones. Por tal motivo, se ha vuelto esencial para mejorar la calidad de vida. Un servicio técnico defectuoso no solo afecta al sector residencial, el confort o el desarrollo cotidiano de las personas, sino que también tiene un impacto en el sector comercial e industrial. Es necesario mejorar la eficiencia, la confiabilidad y la rentabilidad de los sistemas eléctricos de potencia. Así pues, los transformadores de distribución son componentes cruciales de estos sistemas, y cualquier fallo en su funcionamiento puede resultar en interrupciones del suministro eléctrico, lo que afecta a usuarios residenciales, comerciales e industriales.

En otras palabras, la mayoría de los sistemas de monitoreo y diagnóstico de los transformadores de distribución son manuales. Además, la gran cantidad de datos generados por los transformadores de distribución a diario hace que el proceso de análisis y evaluación sea tedioso y laborioso.

A tal efecto, la incorporación de un sistema de predicción de grado de severidad de degradación térmica utilizando imágenes infrarrojas de transformadores de distribución mediante redes neuronales puede mejorar la eficiencia y agilizar los mantenimientos preventivos en los transformadores de distribución.

1.2 Delimitación del problema

Para llevar a cabo el proyecto, es necesario realizar una investigación sobre la Inteligencia artificial y su aplicación en los transformadores de distribución. Es fundamental adquirir una comprensión sólida de las técnicas de inteligencia artificial que están disponibles en la actualidad y cómo se aplican al monitoreo de sistemas energéticos. Antes de evaluar la viabilidad y los beneficios de incorporar la Inteligencia Artificial en los circuitos de transformadores de distribución**,** con el fin de mejorar la eficiencia y la estabilidad de la distribución eléctrica, es fundamental apoyarse de tesis y proyectos realizados que permitan elaborar sistemas de predicción eficiente. Por lo tanto, este aspecto es motivo de análisis en el presente trabajo de titulación.

1.3 Justificación

El mantenimiento de los equipos en un sistema de distribución, como los transformadores, es de suma importancia, en vista de que deben estar en buen estado para garantizar la seguridad y la calidad continua del servicio. Por esta razón, se llevará a cabo un estudio de análisis termográfico. Este tipo de estudio no requiere la desconexión del equipo para su respectivo análisis y, al mismo tiempo, proporciona información confiable.

Además, se incorpora un sistema de predicción de grado de severidad para optimizar la gestión de la energía y agilizar los mantenimientos preventivos en los transformadores de distribución. En términos generales, la inteligencia artificial puede contribuir a mejorar la selección de grado de severidad, así como a reducir el tiempo asociado con el mantenimiento y la reparación.

En consecuencia, el análisis del estado del transformador mediante este tipo de estudio se vuelve difícil para quien lo realiza**,** pues implica la extracción de información a través de imágenes y la generación de un informe con los datos extraídos por el técnico encargado de la generación de reportes. Dado que este proceso debe llevarse a cabo en todos los transformadores de distribución a nivel de empresa, requiere mucho tiempo para su análisis. Por lo tanto, se justifica, por cuanto tiene como propósito mejorar su eficacia incorporando un sistema de predicción o clasificación de su grado de severidad.

1.4 Objetivos

1.4.1 Objetivo general

Desarrollar un sistema inteligente de detección del estado de transformadores mediante análisis de imágenes termográficas con redes neuronales detectando de manera rápida el estado de severidad en transformadores de distribución.

1.4.2 Objetivos específicos

- Analizar imágenes infrarrojas en campo con una cámara termográfica para una base datos.
- Desarrollar un sistema de detección del estado de transformadores mediante redes neuronales convolucionales que permita un análisis rápido y efectivo de mantenimiento en transformadores de distribución.
- Evaluar el sistema de predicción mediante análisis estadísticos evidenciando su eficiencia en la detección del estado de transformadores.

El presente trabajo se lo ha estructurado de la siguiente manera: en el CAPÍTULO II, se detalla la fundamentación teórica; en el CAPÍTULO III, se realiza el desarrollo, análisis y discusión de los resultados. Para culminar con el CAPÍTULO IV, donde se explican las conclusiones, recomendaciones, bibliografía y anexos de esta investigación.

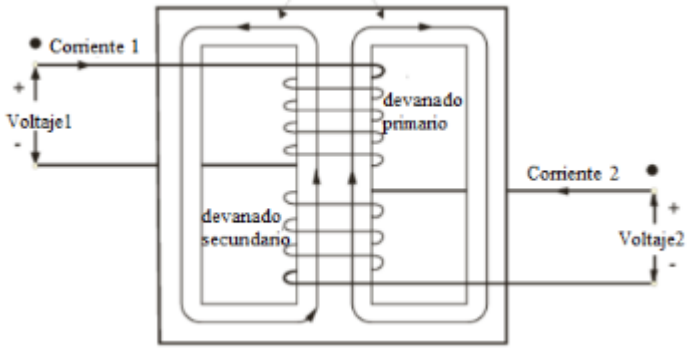
CAPÍTULO II
2. MARCO TEÓRICO

En el siguiente capítulo, se expondrán las teorías que se utilizarán como marco de referencia en la investigación. Se menciona de manera clara y concisa los conceptos fundamentales de los transformadores de distribución, así como los principales enfoques teóricos, modelos clave que guiaran el análisis y la interpretación de los resultados.

2.1 Transformadores

2.1.1 Definición

Un transformador es un dispositivo eléctrico que permite variar alguna función de corriente alterna, como el voltaje o la intensidad. Consiste en dos bobinas de alambre aislado, llamadas devanas, que están enrolladas alrededor de un núcleo de hierro laminado. El devanado primario está conectado a la fuente de energía y el devanado secundario suministra la energía transformada a la carga. Los transformadores son ampliamente utilizados en sistemas de distribución de energía eléctrica para elevar o reducir el voltaje de la corriente alterna, lo que permite una transmisión y distribución eficiente de la energía (Saldívar, 2018). En la Figura 2.1, se muestra el esquema de un transformador.



¡Error! No se encuentra el origen de la referencia. *Esquema de un transformador*

Fuente: Tomado de (Torresí, 2020).

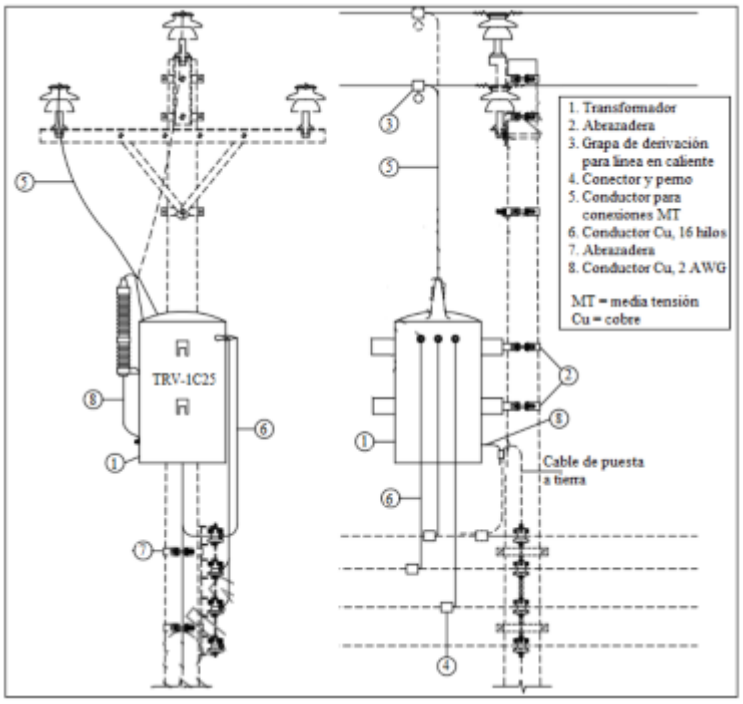
2.1.2 Transformadores de distribución

Un transformador de distribución es un tipo de transformador utilizado en sistemas de distribución de energía eléctrica. Su función principal es cambiar el nivel de voltaje de la corriente eléctrica para adaptarse a las necesidades de los consumidores. Estos transformadores se utilizan para elevar el voltaje en las subestaciones de distribución, lo que permite transmitir la energía eléctrica a largas distancias con pérdidas mínimas. Luego, en los

transformadores de distribución ubicados cerca de los consumidores finales, el voltaje se reduce nuevamente para que sea seguro y utilizable en los hogares, las empresas y las industrias.

Los transformadores de distribución están diseñados para ser eficientes, y generalmente se instalan en postes o en cajas subterráneas. Cualquier descuido puede resultar en costos significativos para la empresa eléctrica que los opera.

Al seleccionar la capacidad de un transformador, es necesario tener en cuenta la cantidad de carga que tendrá que soportar, prestando especial atención a factores como el factor de demanda y de coincidencia que están incluidos en la carga eléctrica. Los transformadores de distribución son dispositivos que suministran la última etapa de transformación en la red de energía eléctrica. Para llevar a cabo el montaje de un transformador de distribución instalado en poste, se requiere una serie de elementos como abrazaderas, conectores, conductores y grapas de derivación. La Figura 2.2 muestra la instalación de un transformador convencional monofásico de 25 kVA, donde los conductores de salida representan los circuitos del transformador a las líneas de baja tensión (Empresa Eléctrica Quito S.A., 2014).



¡Error! ***No se encuentra el origen de la referencia.*** *Transformador monofásico instalado en poste*

Fuente: Tomado de (Empresa Eléctrica Quito S.A., 2014)

2.1.3 Características generales

Existen diversos transformadores en las redes de distribución que pueden ser reconocidos a través de la nomenclatura:

- Primera parte: el Código TR identifica un transformador de distribución.
- Segunda parte: indica el nivel de voltaje que se emplea en la actualidad en Ecuador, tales como:
    - C= 120,121 O 127 [V].
    - E= 0 [V].
    - D= 240/120 O 220/127 [V].
    - U= 440/256 O 480/227 [V].
    - S= 6.3 [kV].
    - T= 13.8/7.96 o 13.2/7.62 [kV]
    - V= 22/12.7 o 22.8/13.2 [kV].
    - R= 34.5/19.92 [kV].
    - 0= se coloca el cero cuando no cumple con ninguna de las anteriores letras.
- Tercera parte: se indica el número de fases o hilos, ya sea monofásico (1), dos fases (2) o trifásico (3).
- Cuarta parte: este campo se utiliza para definir la ubicación y el tipo del transformador. Se debe detallar los que están instalados en postes, tales como:
    - A = Autoprotegidos.
    - C = Convencional.
    - P = Pedestal convencional.
    - E = Pedestal autoprotegido.
- Quinta parte: el último campo determina la potencia del transformador que va desde 3 kVA hasta 1 MVA.
- Los transformadores de distribución tienen varios componentes internos como externos que conforman su estructura física, tales como: *bushings* (bujes), placa, salidas a tierra, circuito electromagnético, papel diamantado, conmutadores, entre otros; vistos en la Figura 2.2.

2.2 Degradación de aislamientos

La degradación de los aislamientos puede ser el resultado del envejecimiento debido a la exposición a altas temperaturas y humedad, la contaminación causada por la acumulación de polvo y otras sustancias extrañas en el aislamiento, las vibraciones y movimientos que pueden dañarlo, las sobretensiones debidas al exceso de voltaje, y los cortocircuitos que resultan en fallas eléctricas, los cuales pueden causar daños significativos al aislamiento y reducir su capacidad para resistir el voltaje (Haya, 2015). Por lo tanto, es esencial llevar a cabo pruebas e inspecciones regulares para detectar cualquier signo de degradación en el aislamiento de los circuitos de los transformadores, con el fin de garantizar su seguridad y prolongar su vida útil.

2.2.1 Ensayo no destructivo

El ensayo no destructivo es una técnica experimental que permite evaluar la calidad, la fiabilidad y la seguridad de un producto sin causar daño alguno en este. Tales pruebas se llevan a cabo utilizando principios físicos, con el fin de determinar ciertas propiedades físicas de los materiales.

2.2.2 Métodos de detección de degradación de aislamiento

Existen varios métodos de detección de degradación de aislamiento (Chauvin Arnoux, 2010). Estos, se detallan en la Figura 2.3.

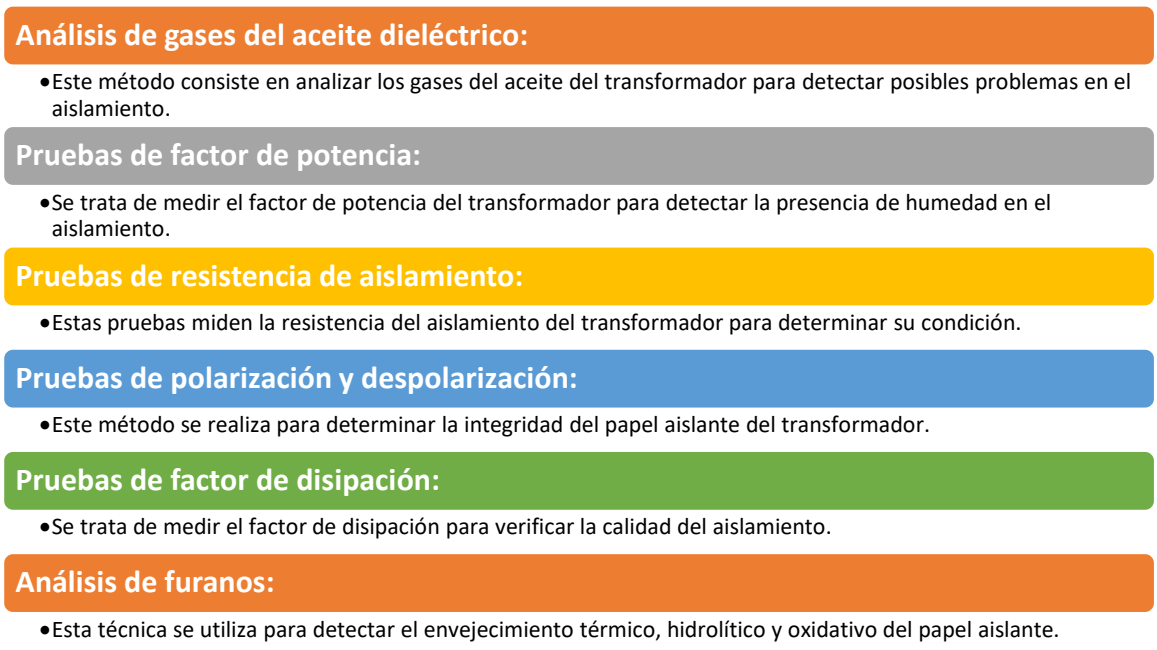


Figura 2.20 Capa de agrupamiento

Fuente: Elaboración propia

Detección de degradación de aislamiento

Fuente: (Chauvin Arnoux, 2010).

2.2.3 Análisis de datos y diagnóstico de la degradación de aislamientos

El análisis y diagnóstico es un proceso en el cual, se emplean diversas técnicas y métodos para evaluar la calidad del aislamiento y detectar signos de su degradación. Estas técnicas pueden incluir pruebas de resistencia, análisis de gases en el aceite, pruebas de factor de potencia, análisis de furanos, entre otras (Haya, 2015). Así pues, el objetivo de este proceso es detectar y prevenir posibles fallas en el transformador antes de que ocurran. Dentro del análisis y diagnóstico de la degradación de aislamientos existen diversas razones que pueden originar dicha degradación, como se indica en la Figura 2.4:

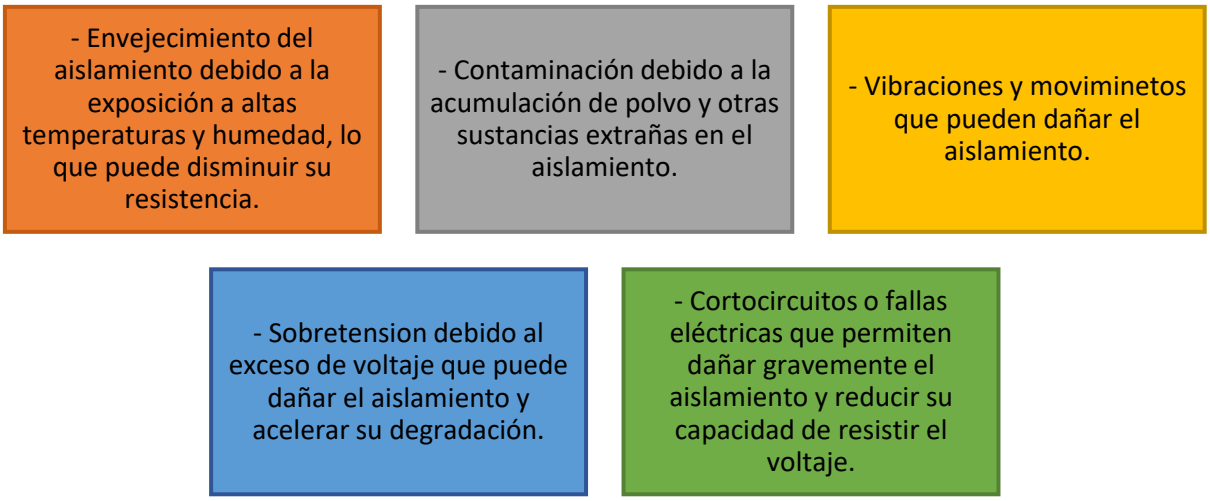


Figura 2.21 Función de pérdida por cada iteración

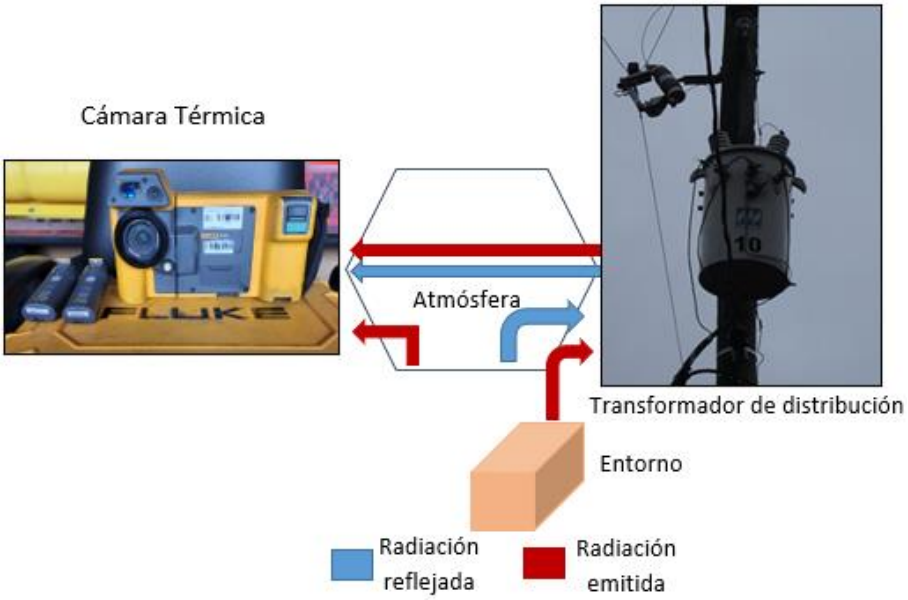
Fuente: Tomado de

Aprendizaje profundo *Posibles razones para la degradación de aislamientos*

Fuente: (Haya, 2015)

2.2.4 Interpretación de la imagen

Cuando se trata de interpretar una imagen térmica se basa en la visualización de las diferencias de temperatura en una escena u objeto. Las cámaras termográficas capturan la radiación infrarroja emitida por los objetos y a convierten en una imagen que representa las variaciones de temperatura. La interpretación de una imagen implica analizar los patrones de temperatura y buscar anomalías o variaciones inusuales que pueden indicar problemas o fallas en los objetos o sistemas que se están observando. En la Figura 2.5, se observa como una cámara termográfica en una imagen puede mostrar áreas de temperaturas más altas de lo normal, lo que podría indicar un sobrecalentamiento o un mal funcionamiento del transformador de distribución.



¡Error! No se encuentra el origen de la referencia. *Radiación térmica de un objeto*

Fuente: Elaboración propia

El objeto emite energía hacia la atmósfera en forma de radiación, tanto la radiación emitida por el objeto como la radiación reflejada del entorno. Sin embargo, en la cámara también se reciben estas dos radiaciones, aunque en menor proporción, debido a la absorción de la atmósfera y la radiación emitida por la propia atmósfera. Por ende, es importante mencionar que la medición de la radiación captada por la cámara infrarroja no solo depende de la temperatura del objeto, sino también de su emisividad. La emisividad de un objeto se refiere a su capacidad de emitir radiación en comparación con un cuerpo negro a la misma

temperatura (Cañada, M.; Royo, R., 2016). Hoy en día, en el mercado se pueden encontrar diversas cámaras térmicas que utilizan software para analizar las imágenes, como se muestra en la Figura 2.6.

Figura 2.14 Operación de convolución con kernel de filtro aumentando de nitidez 3x3

Fuente: Tomado de

Cámara Termográfica

Fuente: (Fluke, 2023).

2.2.5 Ventajas y limitaciones de la detección de degradación de aislamientos

La detección de degradación de aislamientos ofrece ventajas significativas al permitir la identificación temprana de problemas y la planificación de acciones preventivas (Haya, 2015), en la Figura 2.7 se ilustran las ventajas de manera más detallada.

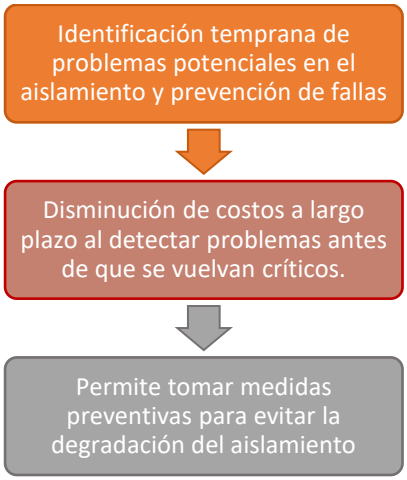


Figura 2.15 Redes Neuronales convolucionales clásicas a) LeNet b) AlexNet y c) VGG

Fuente: Elaboración propia

Ventajas de la detección de degradación

Fuente: (Haya, 2015).

Sin embargo, también presenta limitaciones en términos de costos, interpretación de resultados y la capacidad de detectar ciertos tipos de degradación (Haya, 2015), en la Figura 2.8 se detalla algunas limitaciones en cuanto al trabajo en campo que debe tener el objeto:

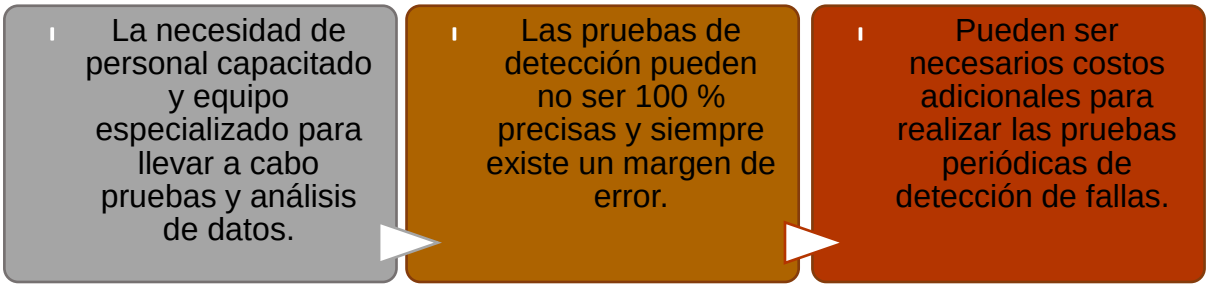


Figura 2.16 Detección de características de una imagen

Fuente: Elaboración propia

Limitaciones de la detección de degradación

Fuente: Tomado de (Gálvez C. & Cobián R., 2018)

2.2.6 Mantenimiento de degradación de aislamientos

El mantenimiento de degradación de aislamientos es un aspecto crucial en diversos sistemas y equipos. Este tipo de mantenimiento se enfoca en prevenir y abordar la degradación de los aislamientos utilizados en componentes eléctricos, estructuras y otros elementos. El principal objetivo del mantenimiento de degradación es garantizar el correcto funcionamiento y la seguridad de los sistemas. Al mantener los aislamientos en buen estado, se evita la perdida de eficiencia, se reducen los riesgos de fallas y se prolonga la vida útil de los quipos. En la Figura 2.9 se puede incluir varias acciones:

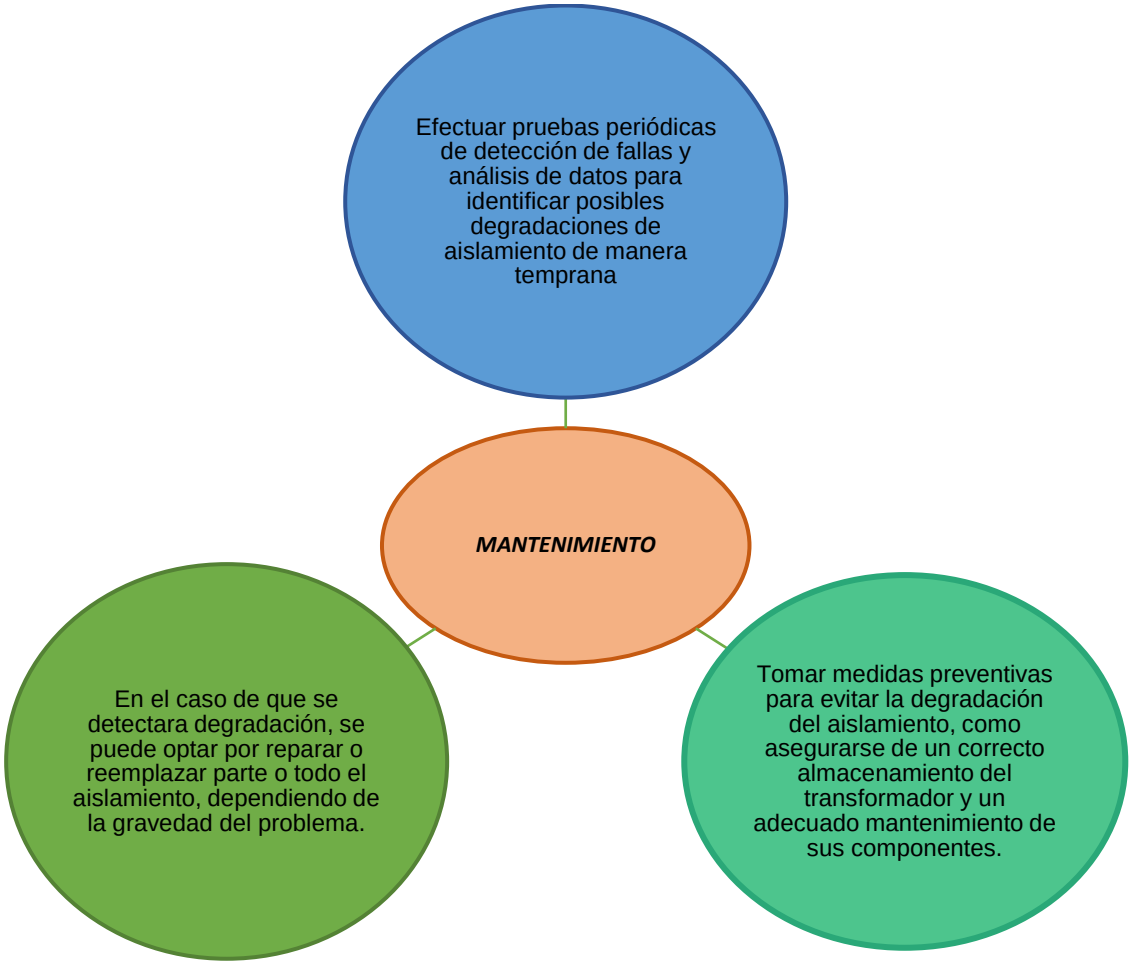


Figura 2.17 Convolución de un kernel sobre una imagen 3D

Fuente: Elaboración propia

Acciones preventivas en mantenimiento por degradación

Fuente: (Gálvez C. & Cobián R., 2018)

Ahora bien, es importante destacar que el mantenimiento de la degradación del aislamiento requiere de personal capacitado y equipo especializado para llevar a cabo pruebas y análisis de datos. Puede ser necesario realizar algunas periódicas de detección de fallas.

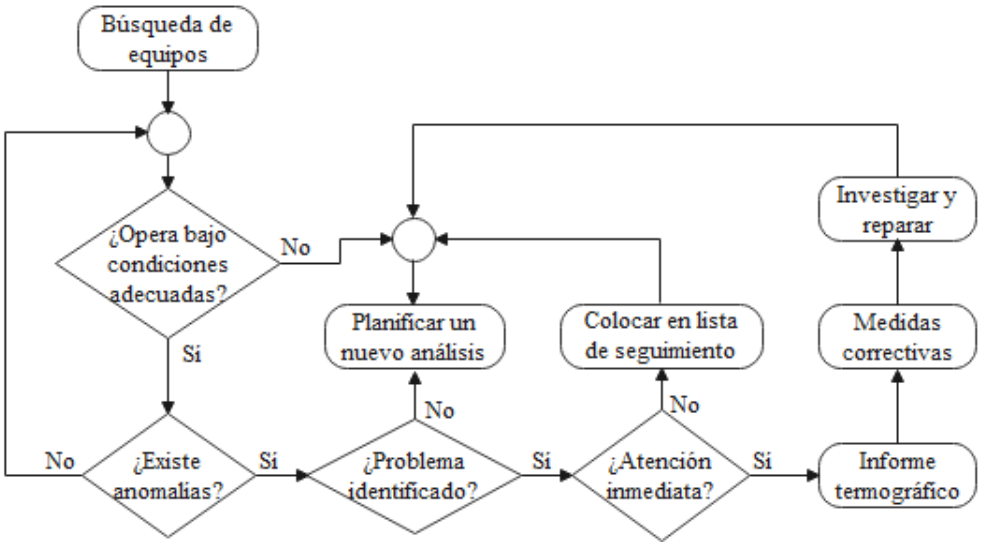


Figura 2.20 Capa de agrupamiento

Fuente: Elaboración propia

Proceso de mantenimiento preventivo

Fuente: Elaboración propia

En consecuencia, se examinan las condiciones y, si se detecta alguna anormalidad junto con el problema identificado, puede o no requerir atención inmediata. Una vez que se detecta la anomalía, el equipo de termografía prepara un informe y lo envía al encargado de la planificación de mantenimiento.

2.3 Deep Learning

2.3.1 Definición

El *Deep Learning* o Aprendizaje Profundo es una rama del *Machine Learning* que se enfoca en el entrenamiento de algoritmos de aprendizaje automático, conocidos como redes neuronales artificiales, para que puedan realizar tareas complejas de manera similar al ser humano (Denio, 2012).

Las redes neuronales artificiales algunas técnicas de aprendizaje automático para resolver problemas del mundo real, recurriendo a redes neuronales que simulan la toma de decisiones humanas. Para entrenar las redes neuronales se requieren cantidades masivas de datos de entrenamiento, ya que se debe tener en cuenta una gran cantidad de parámetros para que la solución sea precisa (Aguilar, 2022).

Estas redes neuronales son modelos matemáticos que se componen de muchas capas de nodos interconectados, lo que les permite aprender a través de la exposición a grandes cantidades de datos. Los algoritmos de *Deep Learning* se inspiran en la estructura del cerebro humano, y cada capa de la red neuronal se encarga de aprender características

cada vez más abstractas del conjunto de datos de entrada. A medida que la red neuronal aprende, ajusta los pesos de las conexiones entre sus nodos para optimizar su desempeño en la tarea que se le asignó.

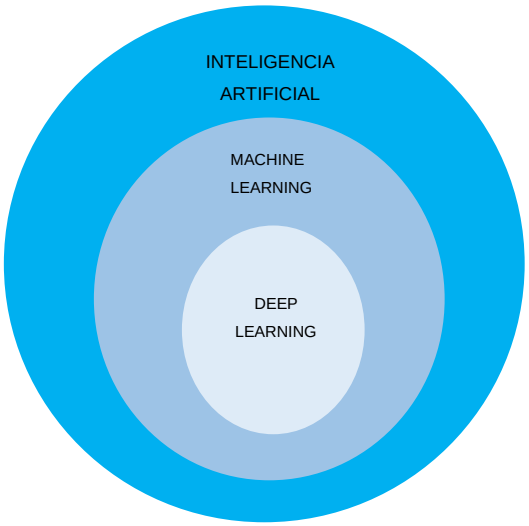


Figura 2.21 Función de pérdida por cada iteración

Fuente: Tomado de

Aprendizaje profundo

Fuente: Elaboración propia

2.3.2 Redes Neuronales Convolucionales

Las redes neuronales es una variación de las redes neuronales artificiales. Es una arquitectura de red neuronal de Deep Learning que aprende directamente de los datos. Estas suelen utilizarse en el reconocimiento de imágenes, clases o categorías. También pueden ser muy útiles para clasificar audios, series temporales y señales de datos (Llanos, 2023).

Las redes convolucionales están formadas por docenas o cientos de capas para procesar datos de entrada y convertirlos en datos de salida. Para entender su funcionamiento, se divide el esquema de funcionamiento en tres fases: campos locales receptivos, pesos y sesgos compartidos, y, por último, activación y puesta en común. En una red neuronal común, cada neurona de entrada está conectada a las capas ocultas de la arquitectura. Sin embargo, en las redes neuronales convolucionales sólo una pequeña región de las neuronas de entrada está conectada a las capas ocultas de la red, y a estas regiones a las que se les conoce como campos locales receptivos. Se puede usar convolución para crear un mapa de características de estas conexiones (Llanos, 2023).

Como las redes neuronales este algoritmo tiene pesos y sesgos que va actualizando a cada iteración. Pero en las redes convolucionales todos los sesgos y pesos son iguales para todas las neuronas de esa capa detectan la misma característica en una región de la imagen, y así hacerlo tolerante a los errores que pueda

presentar. Incluye un reconocimiento de imágenes y un clasificador de imágenes termográficas según su grado de severidad. De tal manera, se puede realizar una conversión del estilo deseado entre el contenido y el estilo aplicado, tal como se muestra en la Figura 2.22.2 Estructura con 5 mapas de características

Fuente: Elaboración propia

.

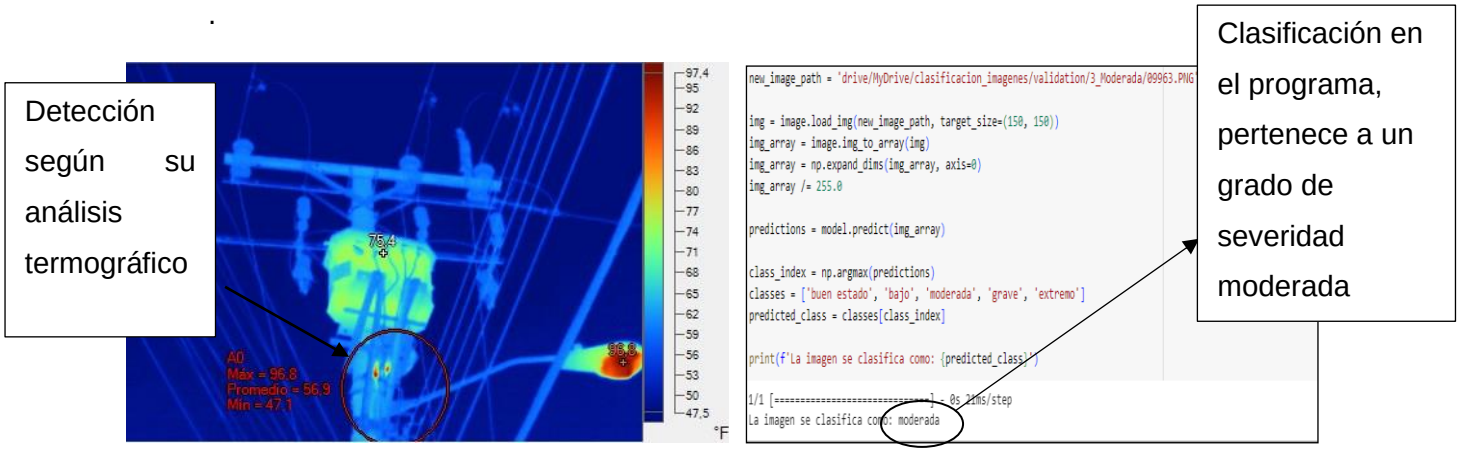


Figura 2.22.2 Estructura con 5 mapas de características

Fuente: Elaboración propia

Detección y clasificación del objeto

Fuente: Elaboración propia

2.3.3 La operación de la convolución

La operación de la convolución es un concepto fundamental en el campo del procesamiento de señales y la matemática aplicada. En términos simples, la convolución es una operación matemática que combina dos funciones para producir una tercera función que representa como una de las funciones influye con la otra.

En el contexto del procesamiento de señales, la convolución se utiliza para analizar como una señal de entrada interactúa con un sistema o filtro. La operación de convolución se realiza multiplicando cada punto de la señal de entrada por el punto correspondiente de la función del sistema, y luego sumando los resultados. Esto se repite para cada punto de la señal de entrada, lo que da como resultado una nueva señal que representa la respuesta del sistema de entrada. La convolución se utiliza en una amplia gama de aplicaciones, como el procesamiento de imágenes, el procesamiento de audio, la comunicación inalámbrica y el análisis de datos. Es una herramienta para comprender como las señales interactúan entre sí y como se ven afectadas por diferentes sistemas.

La operación de convolución en las redes neuronales convolucionales se refiere a la aplicación de una pequeña matriz conocida como *kernel* a una imagen de entrada. Este se desplaza sobre la imagen, lo que resulta en una operación punto a punto entre el *kernel* y los píxeles gráficos, generando una matriz resultante con características detectadas (Moreno M., 2013). La convolución es un procedimiento matemático que combina dos señales para

producir una tercera. Si se tienen dos funciones, por ejemplo, f(t) y g(t), la convolución es un cálculo integral que representa la magnitud de superposición de la función g a medida que se desplaza sobre la función f. La convolución se expresa como se muestra en la **¡Error! No se encuentra el origen de la referencia.**:

Ecuación 1

$$(f * g)(t) \approx^{de\ f} \int_{-\infty}^{\infty} f(\tau)g(t-\tau)dr$$ *Ecuación 2*

Donde:

- f(t) representa la entrada.
- g(t) es el filtro o *kernel*.

La convolución en el procesamiento de señales, se utiliza en el procesamiento digital de señales para estudiar y diseñar sistemas lineales de tiempo invariante (LTI), como los filtros digitales. La señal de salida de los sistemas LTI, $y(n)$ es la convolución de la señal de entrada $x(n)$ y la respuesta al impulso $h(n)$ del sistema.

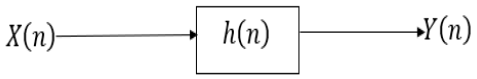


Figura 2.22 Iteración de neuronas
Fuente: Elaboración propia

Fuente: Tomado de (Mathworks, 2023)

Por otro lado, en la práctica, el teorema de convolución se utiliza para diseñar filtros en el dominio de la frecuencia. El teorema de convolución establece que la convolución en el dominio del tiempo es igual a la multiplicación en el dominio de la frecuencia.

La convolución en el procesamiento de imágenes en una operación matemática que se utiliza para combinar una imagen de entrada con un filtro o *kernel* para obtener una nueva imagen de salida. Esta operación se realiza mediante la superposición y multiplicación de los valores de los pixeles de la imagen de entrada con los valores de los píxeles de la imagen de entrada con los valores correspondientes del filtro, y luego sumando los resultados. El filtro o kernel es una matriz pequeña que se desliza sobre la imagen de entrada, multiplicando y

sumando los valores de los pixeles en cada posición. Esto permite resaltar ciertas características de la imagen, como bordes, texturas o detalles específico (Moreno M., 2013).

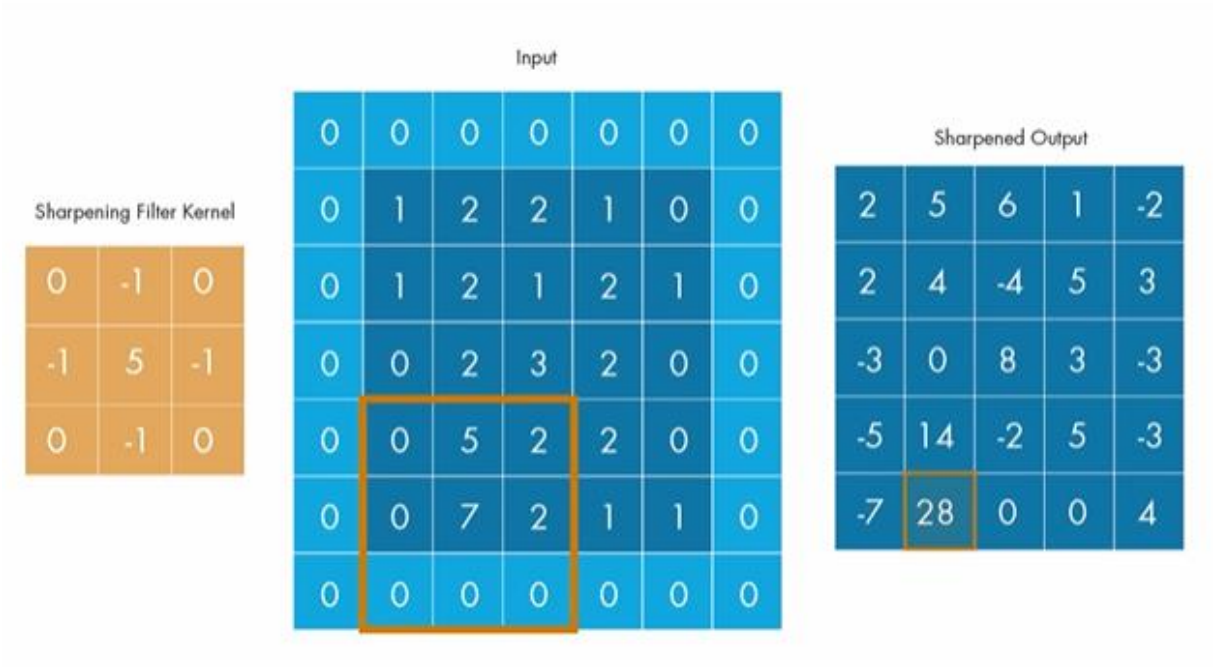


Figura 2.14 Operación de convolución con kernel de filtro aumentando de nitidez 3x3

Fuente: Tomado de *(Mathworks, 2023)*

2.3.4 Arquitectura de la red

Existen tres estructuras y operaciones clásicas en las redes convolucionales. Estas incluyen el modelo LeNet-5, que se utiliza para reconocer caracteres escritos en documentos. Es una red pequeña diseñada con un rango de diez a 100 millones de parámetros, consta de dos capas convolucionales y una capa totalmente conectada, como se muestra en la Figura 2.15. El segundo modelo se llama AlexNet, con casi 60 millones de parámetros, y utiliza ocho capas profundas para clasificar hasta 1000 objetos. Por último, se tiene la red VGG, que tiene una arquitectura profunda y uniforme, con un total de 138 millones de parámetros y su función es reducir el tamaño de las imágenes utilizando capas convolucionales y de agrupamiento de manera consecutiva. Sin embargo, para que funcione de forma correcta durante el proceso de entrenamiento requiere una gran cantidad de datos para ajustarse (Bosch et al., 2019).

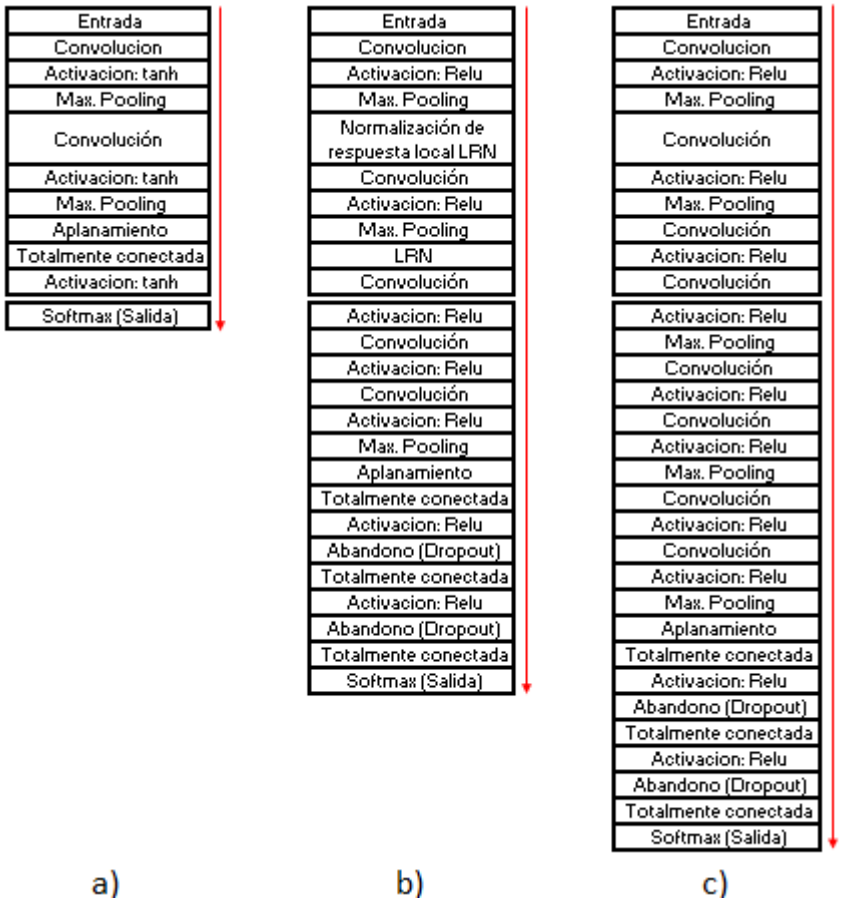


Figura 2.15 Redes Neuronales convolucionales clásicas a) LeNet b) AlexNet y c) VGG

Fuente: Elaboración propia

Así pues, la red CNN utiliza múltiples capas para detectar las características de una imagen, como se puede apreciar en la Figura 2.16. Entre capas, las más comunes son las de convolución, agrupamiento y activación. No obstante, también existen otras capas como la de normalización, totalmente conectada, aplanamiento y de abandono.

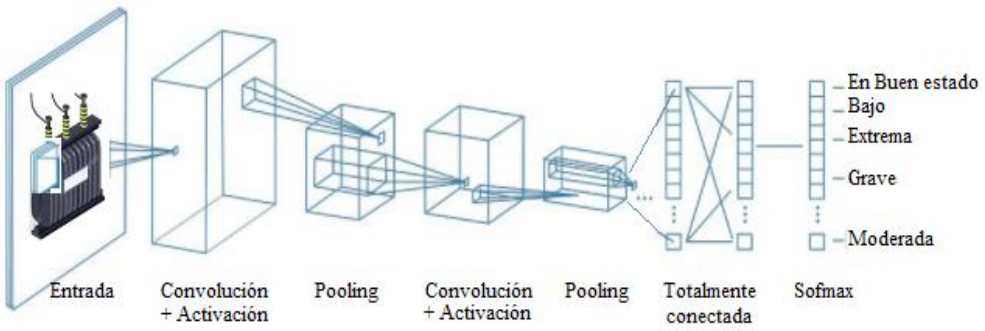


Figura 2.16 Detección de características de una imagen

Fuente: Elaboración propia

2.3.4.1 Filtros o *kernel*

El *kernel* es una matriz de pesos que pertenece a una capa de convolución. Por lo general, su tamaño es menor que la matriz de entrada, que puede ser una imagen a color que sigue los estándares RGB de dos o tres dimensiones (Bosch, Casas, & Lozano, 2019). Además, se utiliza una imagen 2D, pero cuando tiene una dimensión adicional se llama canal o volumen. En la Figura 2.17 se muestra un ejemplo en el que la entrada es de 6x6 píxeles y se utiliza un kernel de 3x3. La salida se calcula mediante la Ecuación 2, para una imagen 3D.

$$R * K_1 + G * K_2 + B * K_3$$ *Ecuación 2*

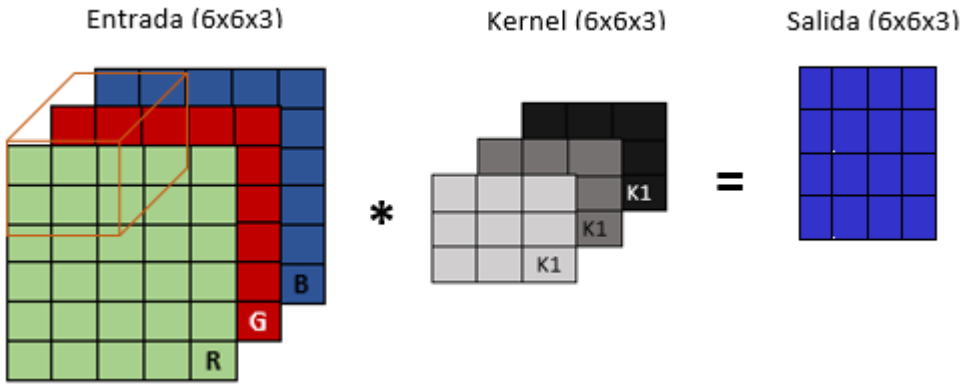


Figura 2.17 Convolución de un kernel sobre una imagen 3D

Fuente: Elaboración propia

2.3.4.2 Padding

Al mismo tiempo, la función del "*padding*" es agregar ceros al borde de la entrada, como se muestra en la Figura 2.18 para que las dimensiones de entrada sean iguales a las de la salida, incluso después de aplicar la convolución.

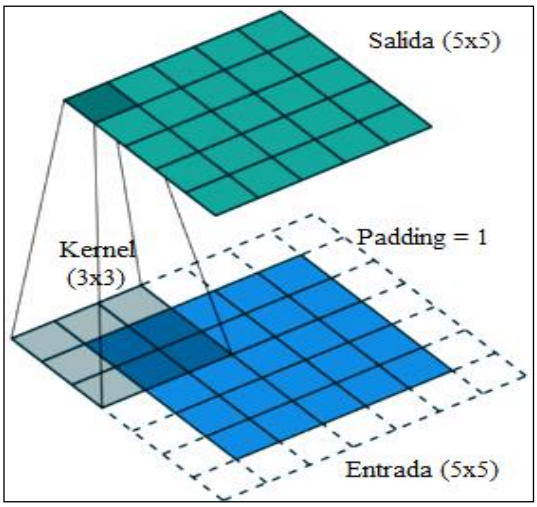


Figura 2.18 Aplicación de un relleno

Fuente: Elaboración propia

2.3.4.3 Convolución por pasos

Uno de los objetivos principales de las redes neuronales convolucionales es reducir las dimensiones de los datos de entrada para mejorar los tiempos de ejecución en la red. La convolución por pasos difiere del enfoque tradicional, dado que implica saltarse celdas (Moreno M., 2013), como se muestra en el ejemplo de la Figura 2.19 Donde se saltan dos celdas en las dimensiones horizontal y vertical, marcando con un punto las celdas de entrada. Por último, la dimensión de salida de la red D_{out} se calcula mediante la Ecuación 3.

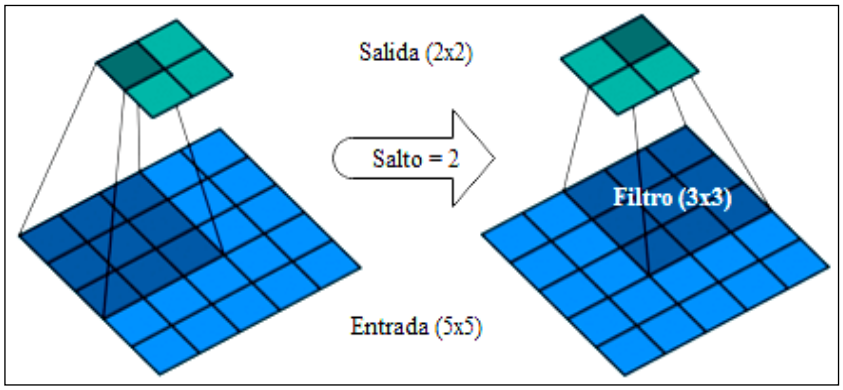


Figura 2.19 Convolución por pasos

Fuente: Elaboración propia

$$D_{out} = \left[\frac{n + 2p - f}{s} + 1\right] x \left[\frac{n + 2p - f}{s} + 1\right]$$

Ecuación 3

Donde:

- n representa la dimensión de entrada.
- p es el *Padding*.
- f representa la dimensión del kernel.
- s es el paso o stride.

2.3.4.4 Capa de agrupamiento

En general, las redes convolucionales utilizan una o varias capas de agrupamiento conocidas como "Pooling Layers" para hacer que las características sean más eficientes y robustas. En estas redes, se utiliza, ante todo, una capa llamada "máximo agrupamiento" que consiste en obtener el valor máximo de un conjunto de valores (Bosch, Casas, & Lozano, 2019). Por ejemplo, si se tiene una entrada de 4x4 y se aplica "Max-pooling", se obtendrá una salida de 2x2, como se muestra en la Figura 2.20 Además, se puede determinar la dimensión de salida utilizando la Ecuación 3, tan solo ajustando los valores del *kernel* .

Figura 2.20 Capa de agrupamiento

Fuente: Elaboración propia

2.3.4.5 Capas de activación o transferencia

Las capas de activación se encargan de seleccionar el valor calculado de las diferentes combinaciones lineales presentes en la transferencia de nodos de una red neuronal. Existen varias funciones de activación, tales como (Bosch et al., 2019):

- El escalón unitario es una función en la que la salida $y(x)$ toma valores binarios {-1, 1} o {0, 1} y $\propto$ representa el umbral de activación, como se muestra en la Ecuación 4. Otra forma de realizar una combinación lineal es a través de una función polinómica de primer orden.

$$y(x) = \begin{cases} 1 & si\ x \geq \propto \\ -1\ o\ 0 & si\ x < \propto \end{cases}$$

Ecuación 4

- La función sigmoide está formada por una exponencial, donde el exponente tiene un parámetro ρ que determina la forma de la curva. El propósito de esta curva es crear un separador suave en la salida, como se expresa en la Ecuación 5 y se muestra en la Figura 2.21.

$$y(x) = \frac{1}{1 + e^{-x/\rho}}$$

Ecuación 5

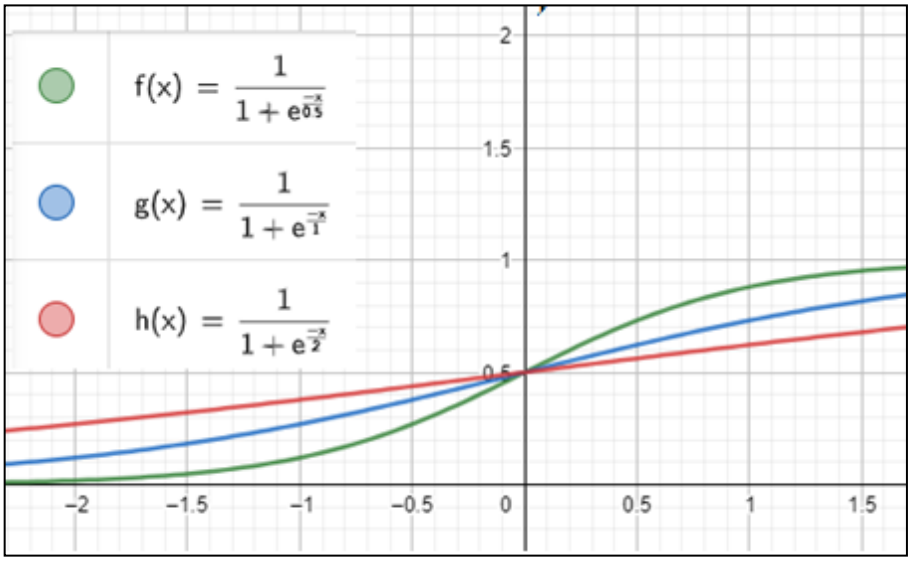


Figura 2.21 Función sigmoide

Fuente: Elaboración propia

- La combinación lineal también puede ser calculada utilizando la función de tangente hiperbólica, como se indica en la Ecuación 6.

$$y(x) = \tanh(x) = \frac{1 - e^{-2x}}{1 + e^{-2x}}$$

Ecuación 6

- La función de unidad lineal rectificada (ReLU) es otra opción disponible que busca asignar valores nulos a los números negativos, lo cual contribuye a aumentar las propiedades no lineales del modelo y la red, como se indica en la Ecuación 7.

$$f(x) = max(0, x)$$ *Ecuación 7*

Es común utilizar una capa no lineal después de la convolución, lo que permite emplear funciones como la sigmoide, la tangente hiperbólica y ReLU. Sin embargo, en la actualidad, la función ReLU es la más utilizada en las redes neuronales debido a su rápido proceso de entrenamiento y su efectividad en la red, sin afectar la precisión final del modelo.

2.3.4.6 Normalización por lotes

La capa de normalización por lotes se utiliza para regularizar las activaciones y gradientes en una red neuronal, con el objetivo de lograr un entrenamiento óptimo y un aprendizaje rápido. Asimismo, las activaciones normalizadas $\widehat{x_i}$ se calculan utilizando estadísticas como la media μ_B y la varianza σ_B^2 en las dimensiones de observación espacial y temporal para cada canal de forma independiente. Cuando las entradas x_i no son óptimas en términos de tener una media cero y una varianza unitaria, se aplican una transformación, desplazamiento y escala y_i como se muestra en la Ecuación 8. La constante $\in$ se utiliza para mejorar la estabilidad numérica en casos de baja varianza. Por otro lado, los parámetros de compensación β y el factor de escala, se pueden aprender y actualizar durante el entrenamiento de la red (Mathworks, 2023).

$$\begin{cases} \widehat{x_i} = \dfrac{x_i - \mu_B}{\sqrt{\sigma_B^2 + \in}} \\ y_i = \gamma \widehat{x_i} + \beta \end{cases}$$ *Ecuación 8*

2.3.4.7 Capa de clasificación

Esta capa calcula la pérdida de entropía cruzada en tareas y clasificación ponderada con clases mutuamente excluyentes mediante la Ecuación 9 (Matich, 2001).

$$Pérdida_{entropía} = -\frac{1}{N}\sum_{n=1}^{N}\sum_{i=1}^{K} w_i t_{ni} \ln y_{ni}$$ *Ecuación 9*

Donde:

- N y K son los números de muestras y clases, cada uno.
- w_i es el peso para la clase i.
- t_{ni} representa el indicador de la n-ésima muestra de la i-ésima clase.
- y_{ni} es la salida de la muestra n para la clase i determinado por la función Softmax.

La pérdida de entropía se emplea para ajustar los pesos del modelo durante la etapa de entrenamiento, con el objetivo de lograr un valor bajo en el aprendizaje profundo y mejorar el rendimiento del modelo (Moreno M., 2013).

2.3.4.8 Capa de abandono (*dropout*)

El *dropout*, o capa de abandono, es una técnica utilizada en el entrenamiento de redes neuronales para prevenir el sobreajuste. El sobreajuste ocurre cuando una red neuronal se ajusta a los datos de entrenamiento y no generaliza bien nuevos datos. La capa de abandono funciona desactivando de forma aleatoria un porcentaje de las neuronas durante el entrenamiento. Esto evita que las neuronas dependan demasiado de las otras neuronas y ayuda a que la red neuronal aprenda características más robustas y generalizables. La capa de abandono reduce la complejidad de la red y evita la memorización excesiva de los datos de entrenamiento. Esto permite que la red neuronal se vuelva más resistente y las variaciones en los datos (Matich, 2001).

2.3.4.9 Capa totalmente conectada

La capa completamente conectada proporciona un vector de N dimensiones como salida para cualquier tipo de entrada, ya sea convolución, agrupación o ReLU, donde N representa el número de clases del modelo (Matich, 2001). Además, utiliza un enfoque de clasificación basado en probabilidades, lo que significa que sus valores oscilan entre 0 y 1. La capa de salida puede ser una neurona tipo sigmoide, donde el valor de entrada z_j^L se calcula mediante la Ecuación 10.

$$z_j^L = \sum_k w_{jk}^L y_k^{L-1} + \propto_j^L \qquad \textit{Ecuación 10}$$

Donde:

- L representa la capa y j es la neurona.

- w_{jk}^{L} indica el peso asignado a la conexión entre las neuronas k y j de las capas $L-1$ y L, cada uno.
- y_{k}^{L-1} es el valor de salida de la neurona k y capa $L-1$.
- $\propto_{j}^{L}$ representa el sesgo de la neurona k y capa L.

La función Softmax también se puede utilizar para calcular la capa de salida, como se muestra en la Ecuación 11. Esto implica que la suma de los valores de salida siempre es positiva e igual a uno; lo que significa que la función Softmax se interpreta como una distribución de probabilidades (Bosch, Casas, & Lozano, 2019).

$$y_{j}^{L} = \frac{e^{z_{j}^{L}}}{\sum_{k} e^{z_{k}^{L}}} \qquad \textit{Ecuación 11}$$

La capa completamente conectada recibe la salida y las características de alto nivel o probabilidad correspondientes a cada clase específica.

2.3.5 Técnicas de optimización

El descenso del gradiente es un método de optimización utilizado en redes neuronales con el objetivo de determinar los pesos o coeficientes de los algoritmos de aprendizaje. De igual manera, se utiliza un modelo para hacer predicciones sobre los datos de entrenamiento, el cual se actualiza en función del error en dichas predicciones. Por último, existen diversos algoritmos de aprendizaje que se pueden aplicar, como el estocástico, por lotes o en mini lotes (Bosch, Casas, & Lozano, 2019).

2.3.5.1 Descenso del gradiente estocástico SGD

El descenso del gradiente estocástico (SGD) es utilizado para calcular el error y actualizar los parámetros (pesos y sesgos) de la red. Con el objetivo de reducir la función de pérdida $\nabla E(\theta_e)$ en cada iteración (Matich, 2001), como se muestra en la Figura 2.21 y puede ser calculado mediante la Ecuación 12. Además, el gradiente se considera estocástico debido a que es una estimación ruidosa, debido a que se aplica la actualización de los parámetros calculados en un mini lote.

$$\theta_{e+1} = \theta_e - \alpha \nabla E(\theta_e) \qquad \textit{Ecuación 12}$$

En donde *e* representa un número de iteración, α la tasa de aprendizaje y debe ser mayor a 0 y θ el vector de parámetros.

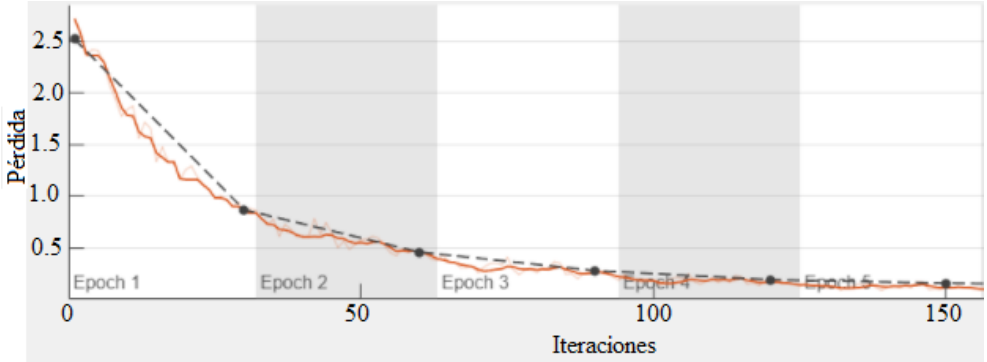


Figura 2.21 Función de pérdida por cada iteración

Fuente: Tomado de (Mathworks, 2023)

Se define tres tipos de algoritmos SGD, detallados a continuación (Mathworks, 2023):

- El Descenso de Gradiente Estocástico con Impulso (SGDM) es una variante del SGD que utiliza un impulso para reducir las oscilaciones y lograr un descenso más pronunciado durante la optimización. El SGDM se puede calcular utilizando la Ecuación 13, donde γ determina la contribución del paso de gradiente anterior en comparación con la iteración actual.

$$\theta_{e+1} = \theta_e - \alpha \nabla E(\theta_e) + \gamma(\theta_e - \theta_{e-1}) \qquad \textit{Ecuación 13}$$

- La propagación de la raíz cuadrada media (RMSProp) es una técnica que se utiliza junto con el SGDM para aplicar una tasa de aprendizaje a todos los parámetros. Sin embargo, se utiliza el algoritmo RMSProp para aplicar una tasa de aprendizaje a todos los parámetros, como se muestra en la Ecuación 14. En esta ecuación, β_2 representa la caída de la media móvil v_e.

$$v_e = \beta_2 v_{e-1} + (1 - \beta_2)[\nabla E(\theta_e)]^2 \qquad \textit{Ecuación 14}$$

- En la Ecuación 15 se presenta la forma de calcular el algoritmo RMSProp para normalizar las actualizaciones de los parámetros de manera individual. En esta ecuación, ε representa una constante pequeña que se utiliza para evitar divisiones por cero.

$$\theta_{e+1} = \theta_e - \frac{\alpha \nabla E(\theta_e)}{\sqrt{v_e} + \varepsilon}$$

Ecuación 15

- Derivación del algoritmo de estimación de omento adaptivo Adam: este algoritmo es una fusión de dos algoritmos mencionados antes, pues utiliza una actualización de parámetros similar al RMSProp, pero con un término de impulso adicional m_e, como se muestra en la Ecuación 16.

$$\begin{cases} m_e = \beta_1 m_{e-1} + (1-\beta_1)\nabla E(\theta_e) \\ \theta_{e+1} = \theta_e - \dfrac{\alpha m_e}{\sqrt{v_e} + \varepsilon} \end{cases}$$

Ecuación 16

2.3.5.2 Descenso del gradiente por lotes BGD

El algoritmo de Descenso de Gradiente en Lote (BGD) calcula el error para cada ejemplo durante el entrenamiento, pero se actualiza al final de cada época de entrenamiento, después de haber evaluado todos los ejemplos. El uso de esta técnica puede resultar en una convergencia prematura hacia los parámetros (Bosch, Casas, & Lozano, 2019).

2.3.5.3 Descenso del gradiente por mini lotes MBGD

El algoritmo de Descenso de Gradiente por Mini Lotes (MBGD) calcula el error y actualiza los coeficientes del modelo, dividiendo el conjunto de datos de entrenamiento en lotes pequeños o mini lotes con el objetivo de encontrar un equilibrio entre la eficiencia del Descenso de Gradiente Estocástico (SGD) (Bosch, Casas, & Lozano, 2019).

2.3.6 Ventajas de la convolución

A continuación, se detallan las ventajas derivadas de la convolución (Bosch et al., 2019):

- Emplea el producto de matrices en el que cada neurona se conecta con las demás que se encuentran en las capas adyacentes, como se muestra en la Figura 2.22. Asimismo, el tamaño del *kernel* debe ser menor que la entrada para permitir el trabajo con conexiones dispersas.

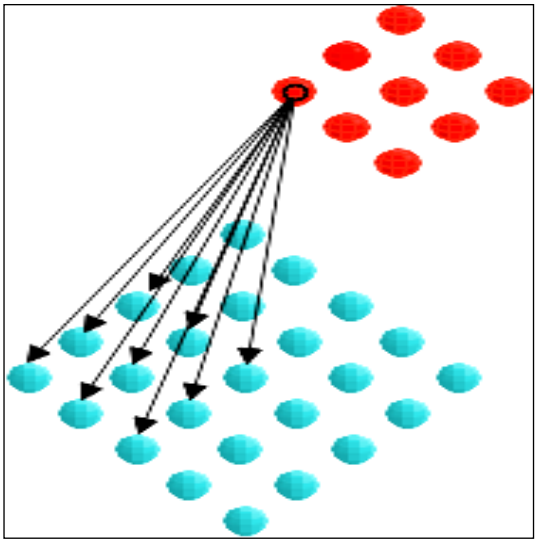


Figura 2.22 Iteración de neuronas

Fuente: Elaboración propia

- Cada neurona oculta tiene un conjunto de pesos y un sesgo que dependen del tamaño de la matriz, y la salida se calcula utilizando la Ecuación 17. Donde σ es la función de activación, b representa el valor compartido del sesgo $w_{l,m}$ son los pesos de las neuronas en la misma capa pertenecientes a la matriz cuadrad de orden n, y $a_{x,y}$ es la posición del valor de entrada.

Ecuación 17

$$\sigma\left(b + \sum_{l=0}^{n-1}\sum_{m=0}^{n-1} w_{l,m} a_{j+l,k+m}\right)$$

- Las características de los pesos y el sesgo definen el mapa de características, donde se relacionan las neuronas de las capas de entrada y oculta. Tal como se observa en la Figura 2.23.

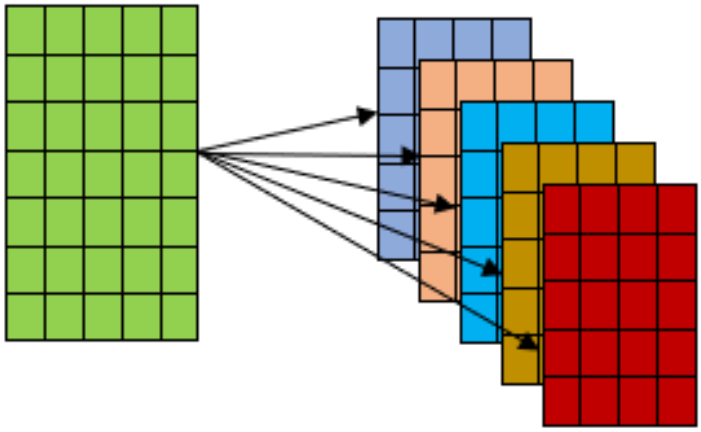


Figura 2.22.2 Estructura con 5 mapas de características

Fuente: Elaboración propia

El mapa ayuda a disminuir una gran cantidad de parámetros de la red convolucional, lo cual ayuda a mejorar el proceso de entrenamiento y, al mismo tiempo, reducir el tiempo y tamaño de la red.

2.4 Norma ANSI/NETA ATS 2021

El Instituto Nacional de Normas Estadounidense (ANSI) y la Asociación Internacional de Pruebas Eléctricas NETA han elaborado normas para las encuestas termográficas. Estas normas establecen un proceso que comienza con una inspección visual y mecánica, que debe incluir un estudio térmico cuando se aplica carga al sistema. En una segunda etapa, se utilizan todos los equipos de protección y dispositivos de seguridad necesarios para retirar la cubierta y llevar a cabo la inspección termográfica. Después, se elabora un informe que contiene los siguientes elementos: (ANSI/NETA-ATS, 2021):

- Paso 1. Descripción del equipo.
- Paso 2. Detallar las discrepancias encontradas.
- Paso 3. Detallar la diferencia de temperatura entre las áreas de interés y referencia.
- Paso 4. Definir la causa probable de la diferencia de temperatura.
- Paso 5. Identificar los equipos y áreas inaccesibles.
- Paso 6. Indicar las condiciones de carga en tiempo real.
- Paso 7. Proporcionar imágenes térmicas y/o fotografías del área deficiente.
- Paso 8. Colocar la acción sugerida con base en la Tabla 1 .

Por último, se determina los parámetros de prueba (ANSI/NETA-ATS, 2021):

- Paso 1. Utilizar cámaras termográficas capaces de detectar una diferencia de temperatura en el rango de 1 a 30 [°C].

- Paso 2. El equipo convierte la radiación emitida en una señal visual.
- Paso 3. Los estudios de termografía se realizan dentro de los períodos de la carga máxima posible, tal como indica la norma ANSI/NFPA 70B.

  Paso 4. Los resultados de la prueba se analizan por medio de la Tabla 1.

Tabla 1. Acciones sugeridas con base en la variación de temperatura ΔT

Nivel	Color	ΔT de puntos similares	ΔT con respecto al ambiente	Unidad de medida	Clasificación	Acción
1	Verde	1-3	1-10	[°C]	Posible deficiencia	Garantiza investigación.
2	Amarillo	4-15	11-20	[°C]	Posible deficiencia	Reparar en la próxima parada.
3	Tomate	>15	21-40	[°C]	Posible deficiencia	Reparar lo más pronto posible.
4	Rojo	>15	>40	[°C]	Posible deficiencia	Reparo inmediato.

Fuente: (ANSI/NETA-ATS, 2021)

Bajo estos parámetros se puede efectuar el diagnóstico termográfico de los diferentes elementos, en este caso del sistema eléctrico.

CAPÍTULO III

3. METODOLOGÍA

En el presente trabajo de investigación se presenta una metodología basada en un enfoque mixto, con un diseño transversal mediante la utilización de las herramientas de investigación, a través de la observación, recopilación de datos, los cuales se desarrollarán entre documentación y revisión de programas avanzados para realizar análisis con el objetivo de determinar la clasificación de grado de severidad por degradación térmica en los circuitos de los transformadores de distribución. Se parte del estudio de los conceptos básicos de transformadores de distribución, características, degradación, diagnóstico y mantenimientos preventivos, siendo estos dos últimos conceptos fundamentales para el desarrollo de esta investigación. El siguiente punto es conocer más a detalle las metodologías de diagnóstico en campo y las tecnologías que componen los dispositivos de análisis termográfico, mediante una revisión del estado actual del arte en bases de datos. Posteriormente, se obtiene una perspectiva general sobre los diferentes modelos y algoritmos de clasificación por computador para predecir los niveles de severidad en transformadores de distribución. En este punto, es esencial recopilar información técnica de imágenes termográficas y códigos de programación, entre otros, es decir conocer a detalle software que se implementará en el sistema de predicción. Al no contar con mucha base de datos, todo el análisis se realiza con datos de un año. Con todo lo detallado, se procede a desarrollar el sistema de predicción para tratar un mantenimiento preventivo- correctivo dependiendo la clasificación de grado de severidad. Finalmente, se realizan pruebas y los resultados obtenidos se comparan con guías, tablas e investigaciones anteriores. La Figura 3.1 muestra un esquema general de la metodología aplicada en esta investigación.

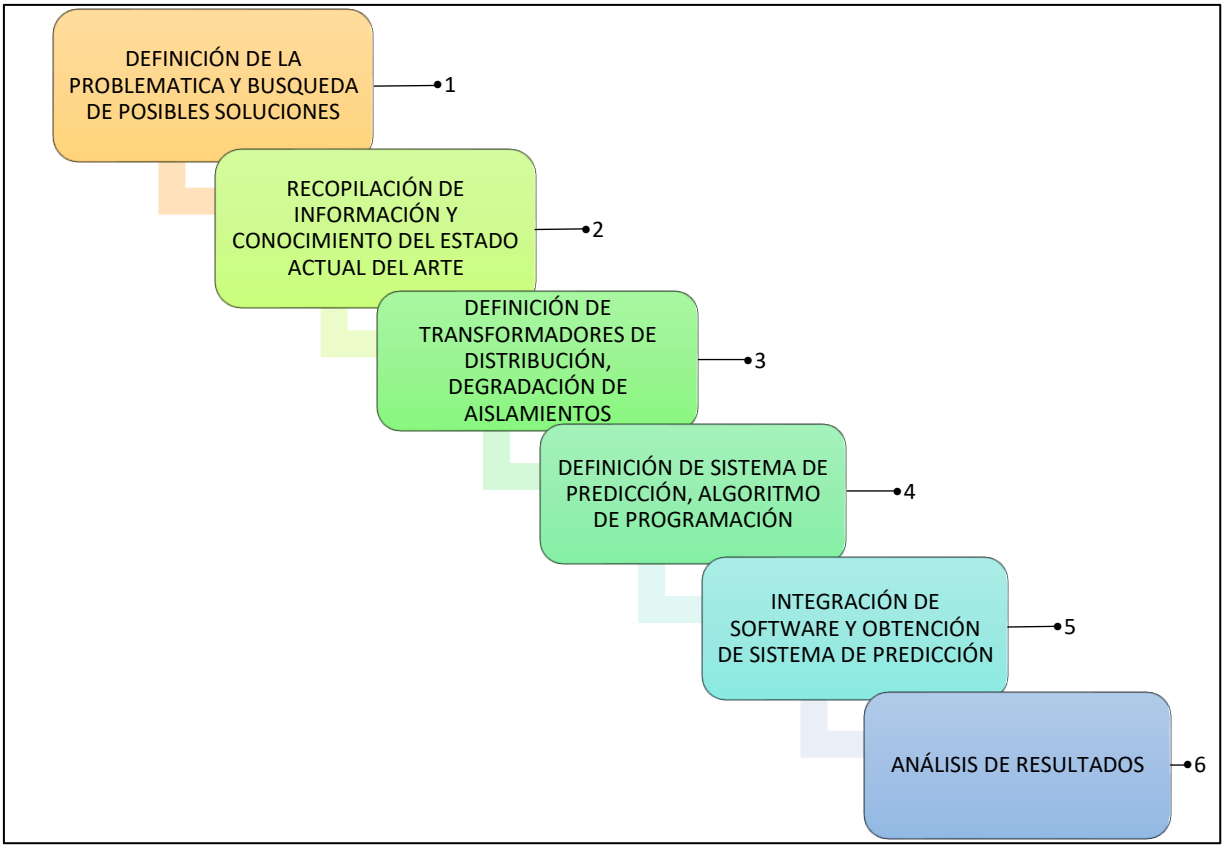


Figura 3.3.1 Metodología por seguir

Fuente: Elaboración propia

A. En la etapa inicial de esta investigación es necesario conocer diferentes investigaciones y trabajos realizados previamente, con respecto a la temática, la problemática y soluciones planteadas.
    - Se utiliza base de datos científicos como: IEEE, entre otros, utilizando palabras claves “inteligencia artificial”, “predicción”, “termografía”,” redes neuronales”, “transformadores de distribución”, “aprendizaje profundo”, incluyendo únicamente artículos académicos y de investigación de la última década.

B. Posteriormente, se plantea el sistema de predicción.
    - Inicialmente, con la información obtenida del apartado anterior y, con base en las normativas, se realiza un análisis termográfico con los niveles de severidad.
    - Se desarrolla un primer sistema, es decir con una cierta cantidad de datos de cada grada de severidad.

C. El siguiente paso es la definición del software:

- Por un lado, se desarrolla la programación del sistema de predicción utilizando librerías y conceptos de programación básica, que será capaz de clasificar las imágenes termográficas según su grado de severidad.

D. Previo a las pruebas, es necesario experimentar el algoritmo de predicción con una carpeta de datos de prueba.
    - Se utiliza una base de datos de prueba en One drive de manera que el algoritmo programado pueda analizar adecuadamente.
    - Se depura errores de programación y funcionamiento.
E. Se realiza pruebas.
    - Se implementa toda la base de datos recolectada y analizada en el sistema de predicción.
    - Se realiza pruebas de funcionamiento, corrección de errores y visualización de datos.
F. Se analiza la información obtenida, se documenta y se publica los resultados.

CAPÍTULO IV

4. DESARROLLO

En esta sección se detallan los diferentes pasos que se realizan para el análisis de grado de severidad en los que se encuentran los transformadores de distribución de ciertas zonas en la ciudad de Azogues y mediante un sistema de predicción clasificar cada grado para un mantenimiento preventivo-correctivo, tanto como para clasificación, utilizando un programa preentrenado con apoyo de Inteligencia artificial.

4.1 Diagnóstico termográfico en circuitos de transformadores de distribución

Se abordará el diagnóstico termográfico en los circuitos de transformadores de distribución de ciertas zonas de la ciudad de Azogues. A medida que los transformadores envejecen y se utilizan, es importante identificar y evaluar los factores que pueden afectar su rendimiento para garantizar una operación eficaz.

4.1.1 Generalidades

Los transformadores de distribución seleccionados para el diagnóstico de degradación por temperatura se encuentran ubicados en diversas áreas de la ciudad de Azogues-Ecuador, abarcando tanto zonas urbanas como rurales. Estos lugares incluyen Virgenpamba, Los Olivos, Jatumpamba, Azogues Centro, Caldera, Solano, entre otros, como se muestra en la Figura 4.1.

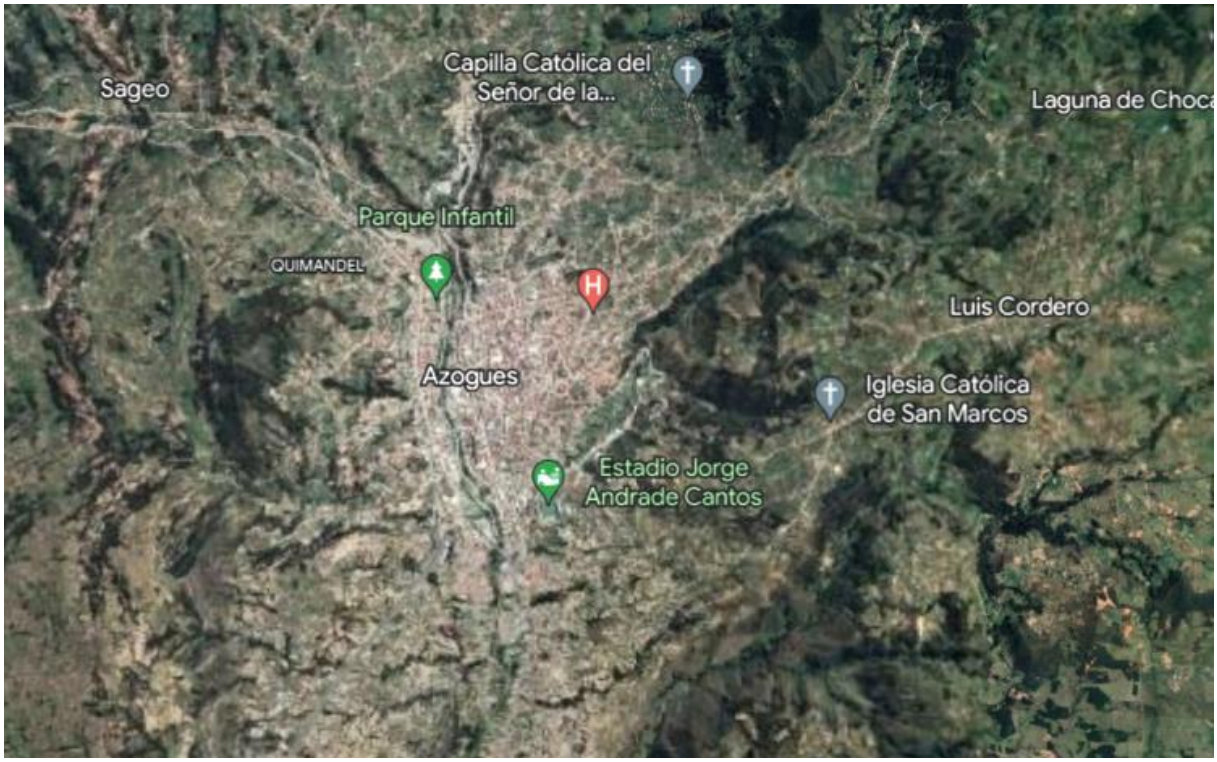


Figura 4.1 Ciudad Azogues-Ecuador

Fuente: Tomado de (Satellites, 2023)

Ahora bien, la Empresa Eléctrica Azogues lleva a cabo inspecciones termográficas de transformadores de distribución entre las 18:00 y las 22:00 en líneas energizadas. De igual manera, utiliza el espectro electromagnético (ver **¡Error! No se encuentra el origen de la referencia.**) para indicar, a través de colores, el nivel de severidad de los problemas encontrados, siguiendo la norma ANSI/NETA-ATS2021. No obstante, en este proyecto se ha agregado un nivel de severidad para cuando los circuitos del transformador se encuentran en buen estado.

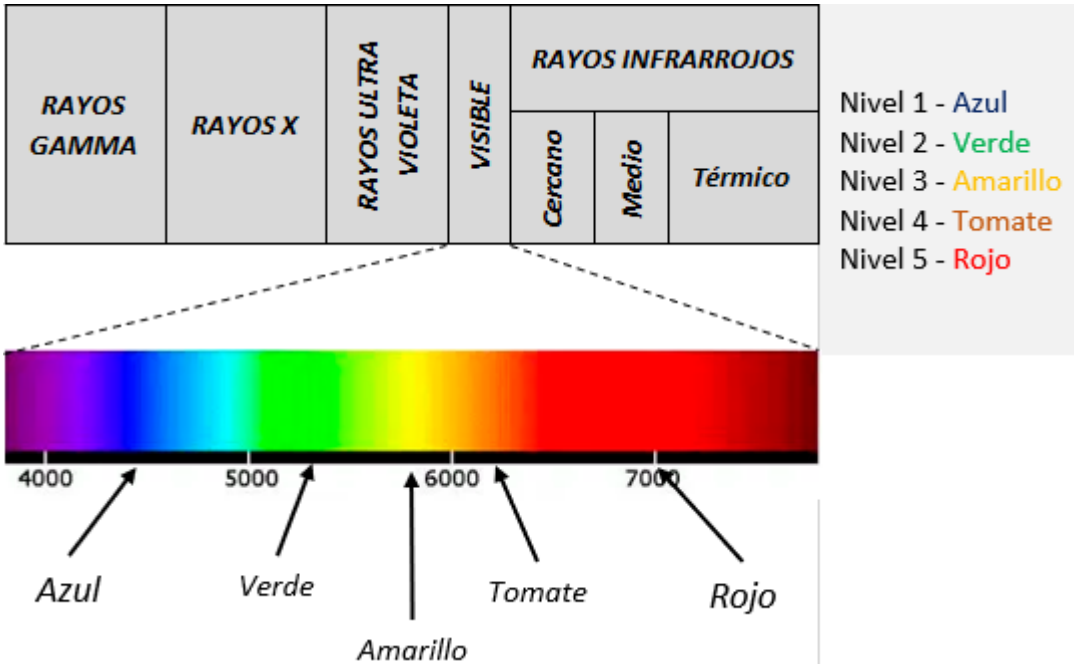


Figura 4.4.2 Espectro electromagnético

Fuente: Elaboración propia

4.1.2 Factores que afectan la degradación en circuitos de transformadores de distribución

4.1.2.1 Degradación por sobrecarga

El envejecimiento del aislamiento de los transformadores de distribución puede acelerarse debido a varios factores, como la sobrecarga y la corriente en el devanado. Cuando un transformador falla debido a una sobrecarga se presenta los siguientes efectos. En las conexiones de baja tensión hay salidas de cobre descolonizadas, el papel aislante de la bobina y salida es quebradizo, aceite dieléctrico ennegrecido o quemado con gran formación de lodo, paredes del tanque descoloridas, formaleta con gran contenido de lodo (Gálvez C. & Cobián R., 2018). En caso de ser por error de conexión, en las bobinas de baja tensión se podrá observar dañada y otra en buen estado. En la **¡Error! No se encuentra el origen de la referencia.**, se muestra un transformador fallado bajo esa condición.

Figura 4.4.3 Transformador fallado por sobrecarga
Fuente: Tomado de (Gálvez C. & Cobián R., 2018)

Conforme el aislamiento envejece, pierde su capacidad para soportar tensiones eléctricas, lo que puede dar lugar a fallos en el transformador. De tal manera, es crucial garantizar que la corriente en los devanados se mantenga dentro del rango permitido por el fabricante del transformador, y llevar a cabo inspecciones periódicas del estado del aislamiento en los circuitos del transformador, para detectar cualquier indicio de degradación y tomar medidas preventivas en consecuencia.

4.1.2.2 Degradación térmica

La degradación por temperatura en los transformadores de distribución es un proceso que ocurre debido a las elevadas temperaturas generadas por la carga de los transformadores. Estos dispositivos están diseñados para operar dentro de un rango de temperatura específico, y cuando la temperatura supera este límite puede provocar una degradación en los materiales empleados en los transformadores (Gálvez C. & Cobián R., 2018). La degradación por temperatura puede tener varios efectos adversos en los transformadores de distribución. Uno de los efectos más comunes es la reducción de la vida útil del transformador. Cuando los materiales utilizados se degradan, pueden perder sus propiedades y fallar más rápido, lo que, a su vez, puede dar lugar a una reducción en la vida útil del transformador. Otro efecto de la degradación por temperatura es la disminución de la eficiencia del transformador. A medida que los materiales se degradan, pueden generar más calor, lo que conlleva a una mayor carga térmica en el transformador. Esto, a su vez, puede hacer que el transformador sea menos eficiente. Cuando la temperatura de trabajo sobrepasa la temperatura establecida por el fabricante, teniendo como consecuencia degradación del

aceite dieléctrico de manera progresiva, lo que trae como consecuencia a mediano plazo el deterioro del equipo, por defecto de una sobrecarga.

4.1.3 Análisis termográfico de su grado de severidad

Ahora bien, cada nivel representa un grado diferente de severidad en la inspección termográfica, siendo el nivel 1 el más bajo y el nivel 5 el más alto. La norma ANSI/NETA ATS-2021 establece dos métodos para calcular la diferencia de temperatura en los circuitos del transformador de distribución a la línea de baja tensión. El primer método se basa en la temperatura ambiente; mientras que, el segundo método se refiere al valor de un cuerpo similar que opera bajo condiciones similares a las del transformador en condiciones normales de funcionamiento (ANSI/NETA-ATS, 2021). En la **¡Error! No se encuentra el origen de la referencia.**, se puede observar que el clima promedio en Azogues, Ecuador, varía entre 6 y 16 °C en el año 2023, por lo que no se puede considerar una temperatura ambiente constante.

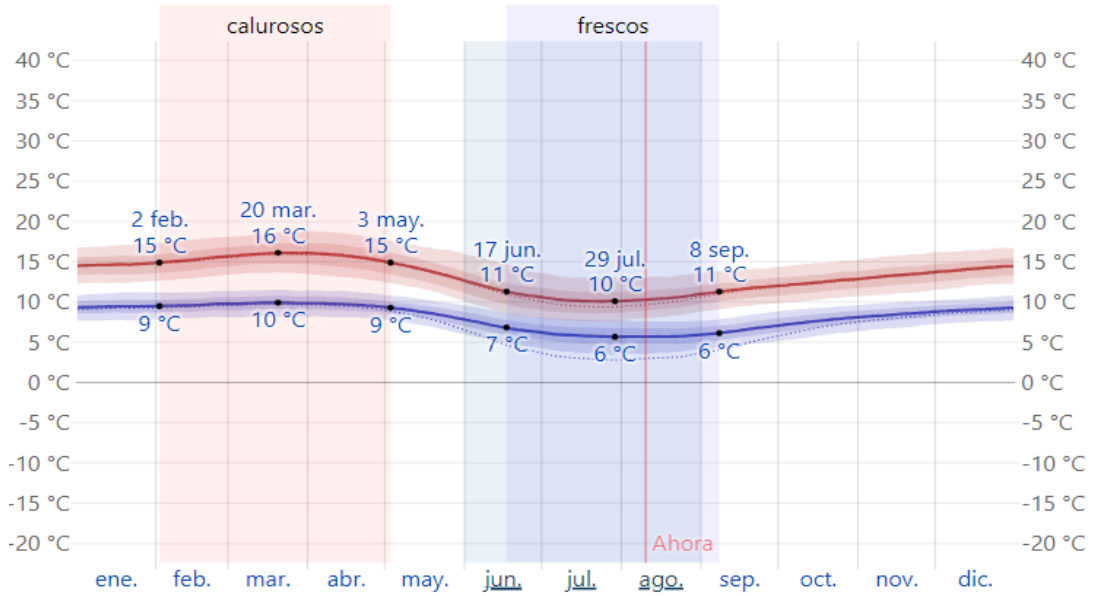


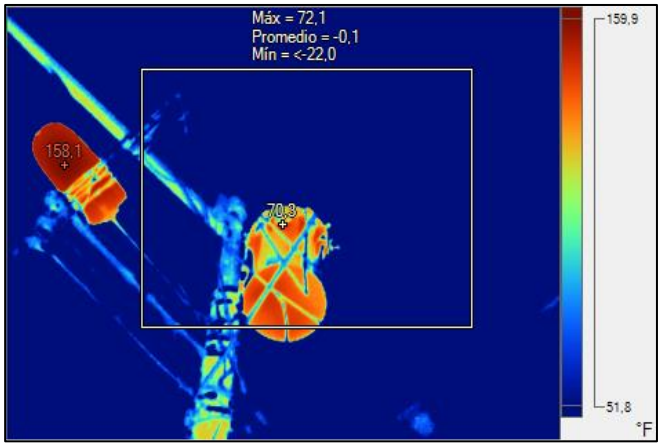
Figura 4.4.4 Promedio de temperatura en Azogues-Ecuador

Fuente: Tomado de (Weatherspark, 2023)

La Empresa Eléctrica Azogues realiza análisis termográfico en los componentes de transformadores de distribución en campo, esta técnica se utiliza para evaluar los posibles problemas que presenta cada uno de ellos, los encargados de realizar esta técnica es una cuadrilla, con el uso de una cámara termográfica para capturar imágenes infrarrojas de los componentes de cada transformador. Estas imágenes revelan las áreas de alta temperatura, lo que puede indicar posibles problemas, como sobrecalentamiento o conexiones defectuosas.

Para llevar a cabo un análisis termográfico, se debe realizar una inspección visual del transformador para identificar puntos calientes o áreas de preocupación. Luego, se utiliza una cámara termográfica para capturar las imágenes infrarrojas de los componentes del transformador, como los devanados, los terminales y los conectores como se muestra en la Figura 4.5.

(a)



(b)

```
new_image_path = 'drive/MyDrive/clasificacion_imagenes/test/2_Baja/08607.PNG'  # Ruta de la imagen

img = image.load_img(new_image_path, target_size=(150, 150))
img_array = image.img_to_array(img)
img_array = np.expand_dims(img_array, axis=0)
img_array /= 255.0

predictions = model.predict(img_array)

class_index = np.argmax(predictions)
classes = ['buen estado', 'bajo', 'moderada', 'grave', 'extremo']
predicted_class = classes[class_index]

print(f'La imagen se clasifica como: {predicted_class}')
```

```
1/1 [==============================] - 0s 18ms/step
La imagen se clasifica como: bajo
```

(c)

Figura 4.5 a) Recopilación de datos del análisis termográfico en campo, b) termografía en el transformador, c) Resultado obtenido en la predicción de grado de severidad de degradación térmica.

Fuente: Elaboración propia

Una vez que se han capturado las imágenes termográficas, se procede a analizarlas. Esto implica examinar las áreas de alta temperatura y compararla con los valores de referencia establecidos. Utilizamos herramientas del programa SmartView Classic 4.4 para realizar mediciones de temperatura y generar informes detallados, como podemos observar en la Figura 4.6.

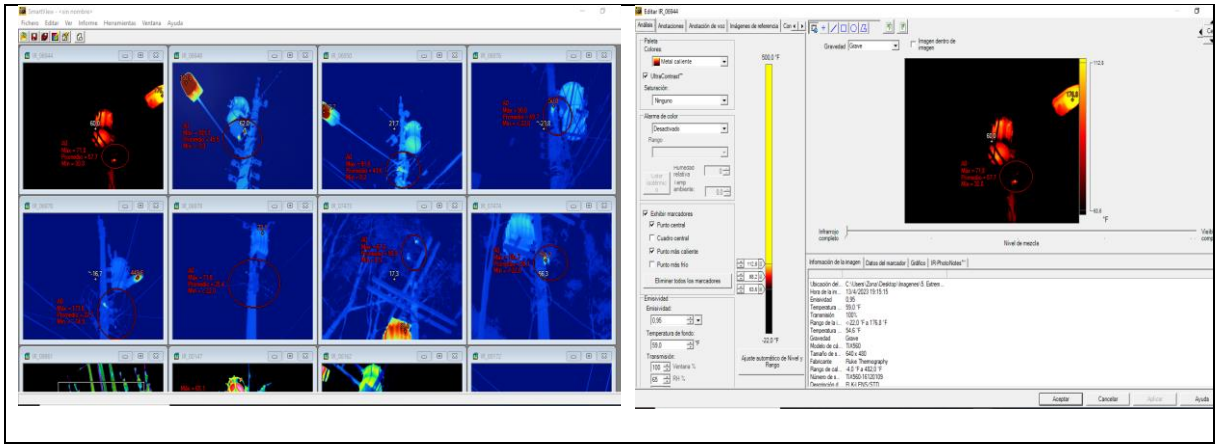


Figura 4.6 Análisis termográfico en SmartView Classic 4.4

Fuente: Elaboración propia

El siguiente paso es interpretar los resultados del análisis termográfico. Se evalúan las áreas de alta temperatura y se determina en que grado de severidad se encuentra el transformador. A continuación, el análisis dependiendo su grado de severidad de degradación térmica.

4.1.4 Severidad Nula

En el grado de severidad nula se refiere que no se observa ningún nivel de deterioro o desgaste significativo, indica que no hay evidencia de degradación o que es mínimo y no representa un problema grave. Esto implica que su resultado puede ser positivo, ya que no se han producido daños significativos. Sin embargo, en la degradación nula no implica necesariamente que no existan otros factores o problemas relacionados. En el presente trabajo se realizó un análisis en donde se encontró transformadore en Buen estado o severidad nula. Se detalla en la Tabla 2.

Tabla 1 Transformadores de distribución en severidad nula o buen estado

# Transformador	Archivo IR	Temperatura Máxima [°C]	ANSI/NETA ATS	Revisión	# Figura
525 37,5KVA	06914	15.7	<1	No necesita revisión	a)
2765 15KVA	06915	16.2	<1		b)

				No necesita revisión	
1698 15KVA	06919	15.2	<1	No necesita revisión	c)

Fuente: Elaboración propia

Nuevamente, se ha utilizado el espectro electromagnético para analizarla degradación térmica en los transformadores seleccionados. Los resultados se visualizan en la Figura4.7.

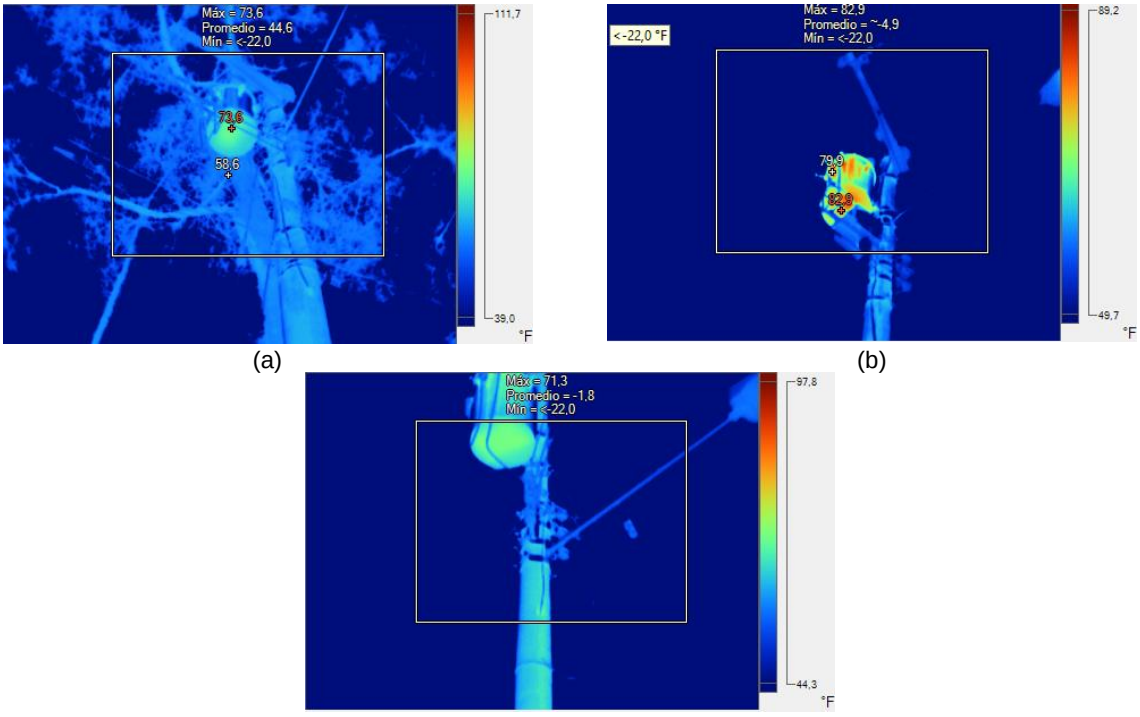


Figura 4.7 Transformadores de distribución- Severidad Nula a) IR 06914 b) IR 06915 c) IR 06919

Fuente: Tomado de (Calle, 2023)

4.1.5 Severidad baja

Severidad baja en grado de degradación se refiere a una situación en la que se observa un nivel relativamente bajo de deterioro o desgaste. Indica que existe algún grado de degradación no significativo. Sin embargo, es importante tener en cuenta que, aunque la gravedad sea baja, aun puede ser necesario tomar medidas preventivas o correctivas para

evitar mayor deterioro. En la Tabla 3. se observan los comentarios de revisión ante una severidad baja de los transformadores de distribución.

Tabla 3 Transformadores de distribución en severidad baja

# Transformador	Archivo IR	Temperatura Máxima [°C]	ANSI/NETA ATS	Revisión	# Figura
2455 37,5KVA	06925	14.4	1-3	Base Baja Tensión	a)
527 25KVA	08567	21.8	1-3	NH Fases A y B	b)
2245 37,5KVA	08568	20.5	1-3	Conector Neutro	c)

Fuente: Elaboración propia

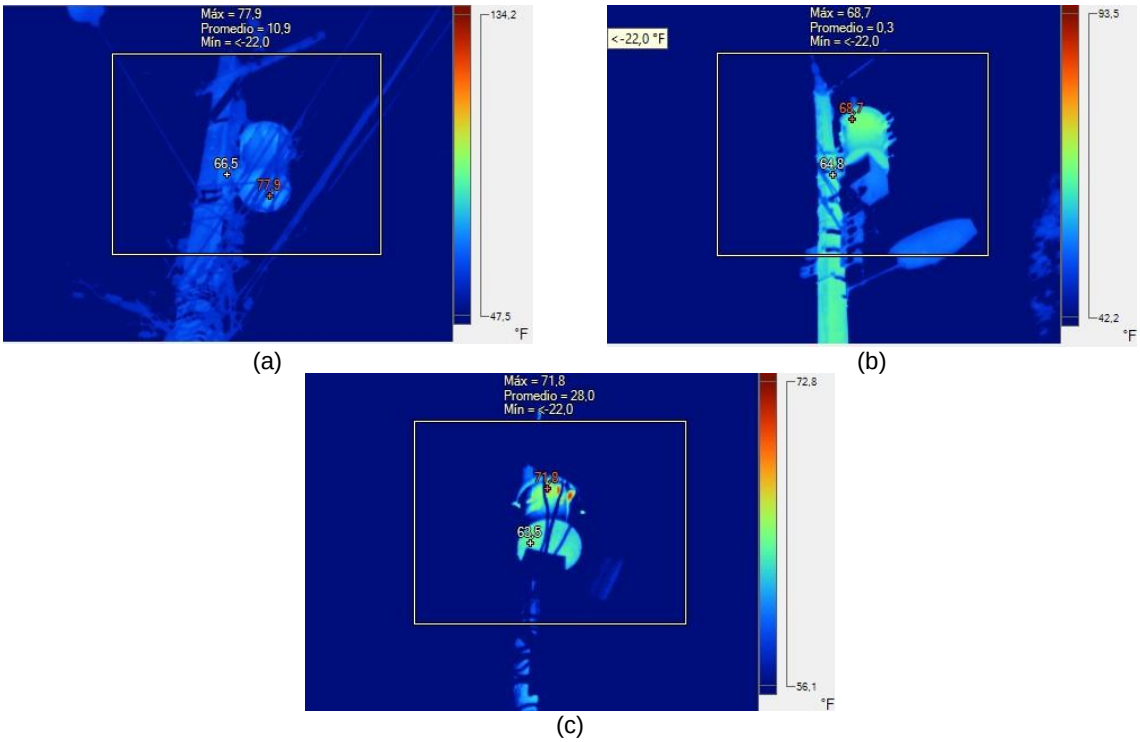


Figura 4.4.8 Transformadores de distribución con grado de severidad baja a) IR 06925 b) IR 08567 c) IR 08568

Fuente: Tomado de (Calle, 2023)

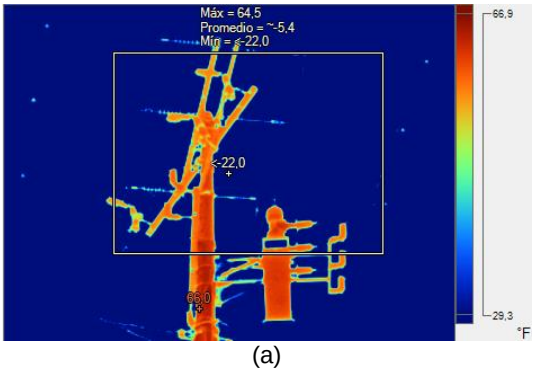
4.1.6 Severidad moderada

En este nivel de severidad indica que se observa un nivel significativo de deterioro o desgaste, pero no alcanza un nivel extremo. Esto implica que los efectos negativos de la degradación son moderados y pueden tener un impacto notable en el transformador. En la Tabla 4. se observan los criterios de revisión ante una severidad moderada de los transformadores de distribución.

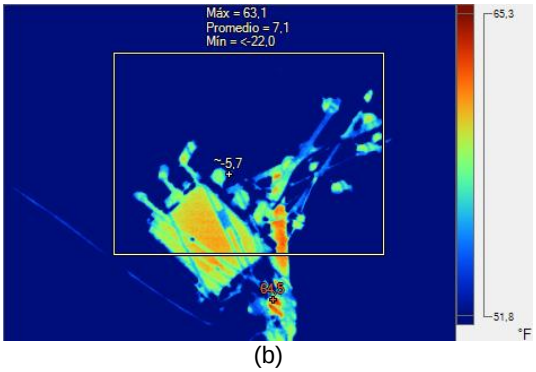
Tabla 4. Transformadores analizados en severidad moderada

# Transformador	Archivo IR	Temperatura Máxima [°C]	ANSI/NETA ATS	Revisión	# Figura
1455 37,5KVA	09944	25.9	4-15	NH Fase A	a)
667 25KVA	09940	19.3	4-15	Seccionador derecho	b)
245 37,5KVA	06931	32.0	4-15	Bushing Neurtro	c)

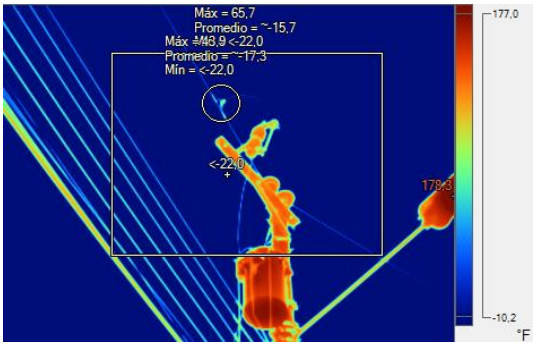
Fuente: Elaboración propia



(a)



(b)



(c)

Figura 4.9. Transformadores de distribución con grado de severidad moderada a) IR 09944 b) IR 09940 c) IR 06931

Fuente: Tomado de (Calle, 2023)

4.1.7 Severidad Grave

La severidad grave en el grado de degradación indica que se observa un nivel significativo de deterioro o desgaste en un contexto determinado, lo cual representa un problema serio. Los efectos negativos de la degradación son notables y pueden tener un impacto importante en el transformador de distribución. En la Tabla 5. se observan los criterios de revisión ante una severidad grave de los transformadores de distribución.

Tabla 5. Transformadores analizados en severidad grave

# Transformador	Archivo IR	Temperatura Máxima [°C]	ANSI/NETA ATS	Revisión	# Figura
275 25KVA	00181	45.3	>15	Base Izquierda	a)
101 15KVA	07472	39.2	>15	Pararrayo	b)
1245 37,5KVA	00007	21.4	>15	Conector	c)

Fuente: Elaboración propia

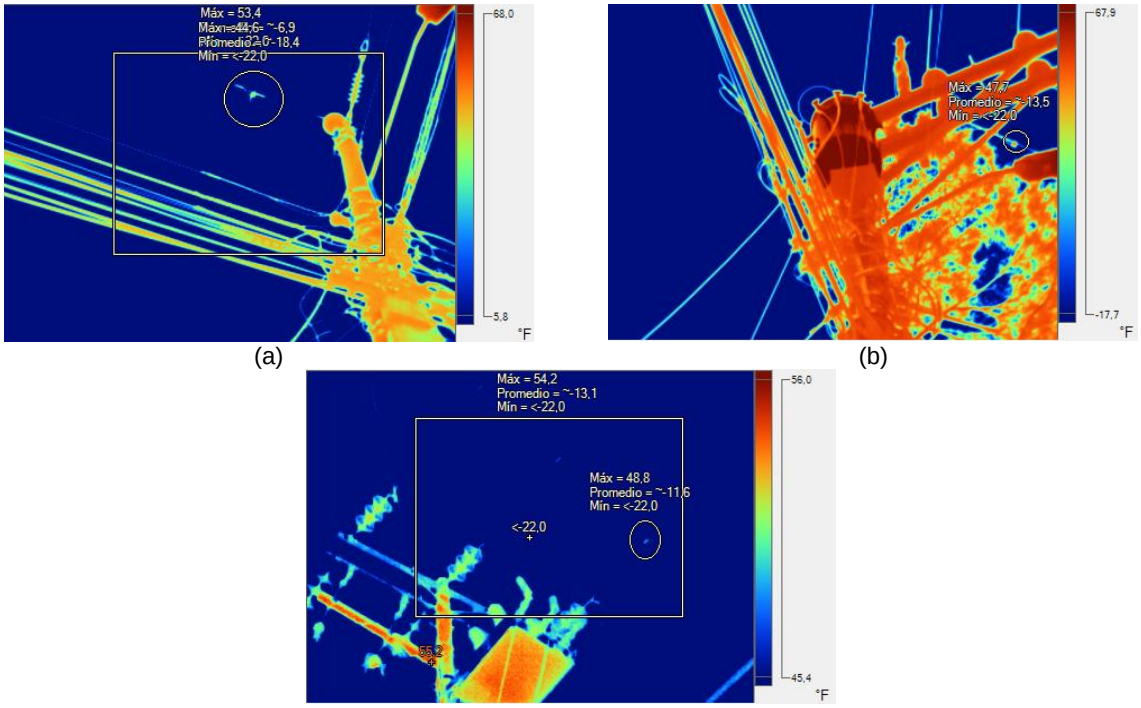


Figura 4.10 Transformadores de distribución con grado de severidad grave a) IR 00181 b) IR 07472 c) IR 00007

Fuente: Tomado de (Calle, 2023)

4.1.8 Severidad extrema

Este nivel es un problema critico que pueden tener consecuencias graves en su funcionamiento y rendimiento. Cuando un transformador experimenta una degradación extrema, pueden ocurrir fallas catastróficas que afectan la distribución de energía eléctrica y puede causar interrupciones en el suministro. En la Tabla 6. se observan los criterios de revisión ante una severidad extrema de los transformadores de distribución.

Tabla 6. Análisis termográfico de transformadores de distribución en severidad extrema

# Transformador	Archivo IR	Temperatura Máxima [°C]	ANSI/NETA ATS	Revisión	# Figura
1323 10KVA	00147	65.1	>15	Anomalía térmica conector NH Fase A	a)
1854 15KVA					

	06944	71.8	>15	Revisar conectores Baja Tensión	b)
761 25KVA	07548	76	>15	Revisar bases	c)

Fuente: Elaboración propia

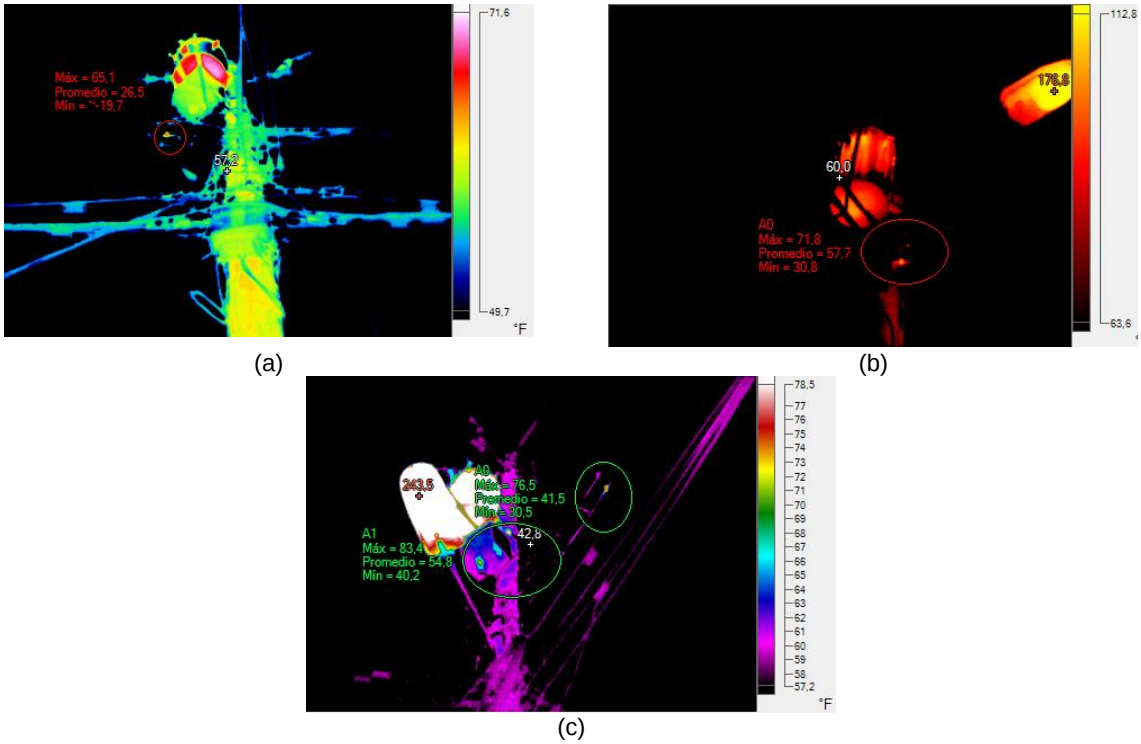


Figura 4.11 Transformadores de distribución con grado de severidad grave a) IR 00147 b) IR 06944 c) IR 07548

En caso de detectar problemas, se pueden tomar medidas correctivas, como ajustar las conexiones, equilibrar la carga o realizar reparaciones.

Es importante destacar que el análisis termográfico de circuitos de transformadores de distribución en campo debe ser realizado por personal capacitado y con experiencia en la interpretación de imágenes termográficas. La Empresa Eléctrica Azogues realiza estos análisis de forma periódica para detectar problemas potenciales antes de que conviertan en fallas mayores.

4.2 Desarrollo del sistema de predicción con IA

En esta sección, se presenta el desarrollo de un sistema de predicción con IA abordando la recopilación y preparación de datos, el entrenamiento del modelo, la evaluación y ajuste.

4.2.1 Diseño de un sistema de predicción con IA

4.2.1.1 Introducción

Se utiliza el software Python para el análisis profundo en imágenes térmicas de los transformadores de distribución. Este enfoque computacional emplea algoritmos de aprendizaje sin depender de una ecuación predefinida. Por lo tanto, se aplican modelos de aprendizaje profundo con el objetivo de determinar el nivel de gravedad de las fallas en los componentes de los transformadores de distribución, ubicados en redes aéreas de la ciudad de Azogues.

4.2.1.2 Datos de entrada

Los datos de entrada hacen referencia al conjunto de imágenes de los transformadores de distribución que han sido clasificados según los niveles de gravedad, considerando que la carpeta training es la que contiene todas las imágenes recopiladas, como se muestra en la Figura 4.12.

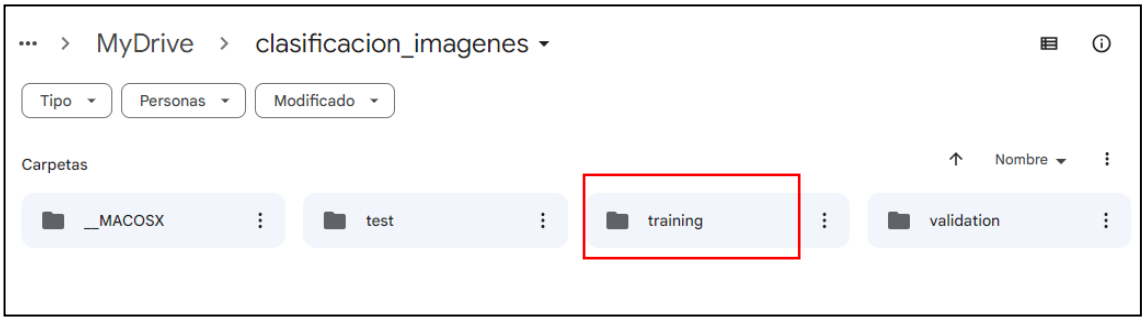


Figura 4.12 Base de datos de validación, entrenamiento y prueba

Fuente: Elaboración propia

Por consiguiente, en el programa establece un contador de imágenes, visto en la Tabla 7., determinando la cantidad numérica por cada carpeta de la Figura 4.12.

Tabla 7. Contador de imágenes termográficas de los transformadores de distribución

Etiqueta (nivel de severidad)	***Cantidad de imágenes***
En buen estado (severidad nula)	*42*
Baja	*46*
Extrema	*60*
Grave	*46*

Moderada	*26*
Total	***226***

Fuente: Elaboración propia

Luego, se asigna el identificador para leer las propiedades del objeto, tales como:

- Ruta específica de las 226 imágenes analizadas, en la Figura 4.13 se visualiza un conjunto de imágenes que se consideró para la predicción.

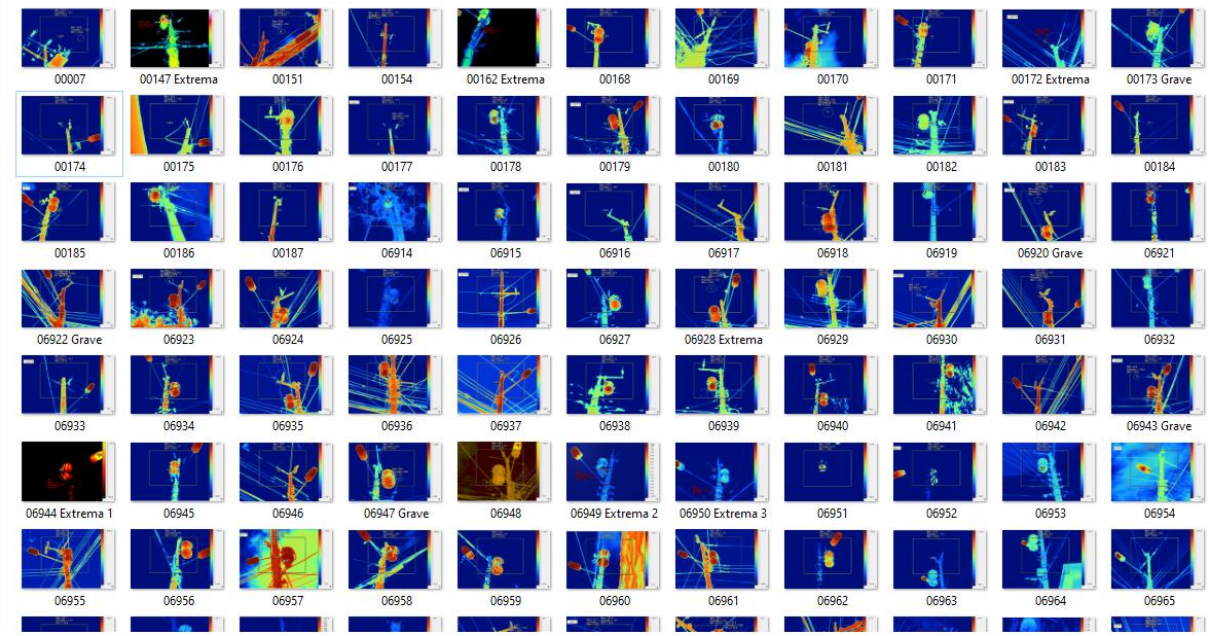


Figura 4.13 Base de datos de 226 imágenes infrarrojas

Fuente: Elaboración propia

4.2.2 Arquitectura CNN

4.2.2.1 Tipos de red CNN

A través de Python se definen los modelos estructurales de las redes neuronales. En este caso, se toma como base el que se muestra en la Figura 4.14, compuesto por cinco capas convolucionales y tres totalmente conectadas.

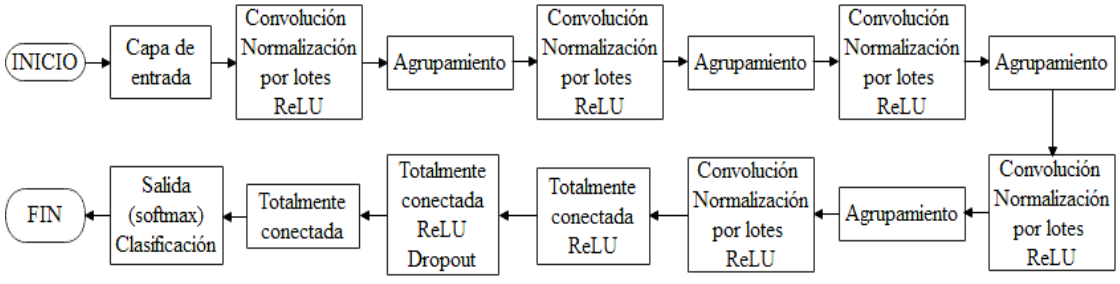


Figura 4.14 Arquitectura base de una red neuronal convolucional

Fuente: Elaboración propia

4.2.2.2 Capa de entrada

Se establece

Se aplica un tamaño normalizado a todos los datos de entrada, especificando la altura, el ancho y el número de canales. Este último toma el valor de uno cuando la imagen está en escala de grises o tres si es en color RGB. En la Figura 4.15, se puede observar cómo la imagen original tiene un tamaño de 784x4802 píxeles y se estandariza a 150x150 píxeles mediante la sintaxis "`img_width, img_height`".

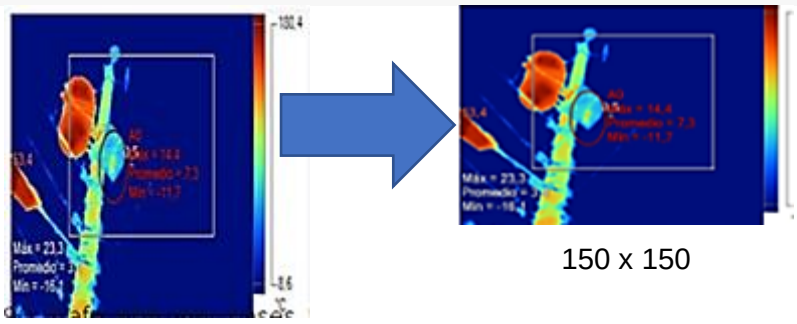


Figura 4.15 Tamaño estandarizado de las imágenes

Fuente: Elaboración propia

4.2.2.3 Convolucional 2D

Por medio de la estructura base, se crea un modelo secuencial utilizando la clase *Sequential* de keras. Este modelo se utiliza para construir una red neuronal convolucional, tal como se muestra en la Figura 4.16.

```
model = Sequential() # Crear un modelo secuencial

model.add(Conv2D(32, (3, 3), input_shape=(img_width, img_height, 3))) # Agregar una capa de convolución con 32 filtros de tamaño 3x3 y esp. la forma de entrada
model.add(Activation('relu')) #Agregar una capa de activación ReLu
model.add(MaxPooling2D(pool_size=(2, 2))) # Agregar una capa de agrupación maxima con un tamaño de agrupación de 2x2

model.add(Conv2D(64, (3, 3))) # Agregar otra capa de convolución con 64 filtros de tamaño 3x3
model.add(Activation('relu')) #Agregar una capa de activación ReLu
model.add(MaxPooling2D(pool_size=(2, 2))) # Agregar otra capa de agrupación maxima con un tamaño de agrupación de 2x2

model.add(Conv2D(128, (3, 3))) # Agregar otra capa de convolución con 128 filtros de tamaño 3x3
model.add(Activation('relu')) #Agregar una capa de activación ReLu
model.add(MaxPooling2D(pool_size=(2, 2))) # Agregar otra capa de agrupación maxima con un tamaño de agrupación de 2x2

model.add(Flatten()) # Aplanar la salida de las capas anteriores
model.add(Dense(64)) # Agregar una capa totalmente conectada con 64 unidades
model.add(Activation('relu')) #Agregar una capa de activación ReLu
model.add(Dropout(0.5)) # Agregar una capa de dropout con una tasa de dropout del 50%
model.add(Dense(5)) # Agregar otra capa completamente conectada con 5 unidades (correspondiente al número de clase)
model.add(Activation('softmax'))# Agregar una capa de activación softmax
```

Figura 4.16 Código

Fuente: Elaboración propia

- Conv2D: Agrega una capa de convolución al modelo. Se especifica el número de filtros, tamaño y la forma de entrada. En este caso agregamos capas de convolución con 32, 64 y 128 filtros de tamaño 3x3, especificando la forma de entrada.
- Activación: Agrega una capa de activación al modelo. En este caso, se utiliza la función de activación ReLU.
- MaxPooling2D: Añade una capa de agrupación máxima al modelo. Esta capa reduce la dimensionalidad de las características extraídas por las capas de convolución.
- Flatten: Aplana la salida de las capas anteriores en un vector unidimensional.
- Dense: Agrega una capa completamente conectada al modelo. Se especifica el número de unidades en la capa.
- Dropout: Agrega una capa de deserción al modelo. Esta capa ayuda a prevenir el sobre ajuste al apagar aleatoriamente un porcentaje de las neuronas durante el entrenamiento.
- Softmax: Agrega una capa de activación softmax al modelo. Esta capa se utiliza para la clasificación multiclase y produce una distribución de probabilidad sobre las clases.

4.2.2.4 Normalización por lotes y ReLU

Se agrega una capa de activación al modelo. En este caso, se utiliza la función de activación ReLU, con la finalidad de reducir la sensibilidad de inicio y acelerar el entrenamiento de la red CNN. Como se indica en la Figura 4.17 .

```
model.add(Activation('relu')) #Agregar una capa de activación ReLu
```

Figura 4.17 Capa de activación ReLU

Fuente: Elaboración propia

La capa ReLU se agrega para capas ocultas de la red neuronal. Esta función ayuda a introducir no linealidad en el modelo y a superar el problema de desvanecimiento del gradiente. La función de activación ReLU se agrega al modelo después de la primera capa oculta. Tanto la normalización por lotes como la función de activación de ReLU son técnicas en el campo del aprendizaje profundo y puede mejorar el rendimiento y la eficiencia de los modelos.

4.2.2.5 Agrupamiento máximo por pasos

Se agrega la capa de Maxpooling2D, la capa de agrupamiento máximo de 2x2 en el modelo base. De esta manera, se asegura que las regiones de agrupamiento no se superpongan cuando los pasos son menores o iguales al tamaño de esta. Lo que permite realizar un muestreo descendente.

```
model.add(Conv2D(64, (3, 3))) # Agregar otra capa de convolución con 64 filtros de tamaño 3x3
model.add(Activation('relu')) #Agregar una capa de activación ReLu
model.add(MaxPooling2D(pool_size=(2, 2))) # Agregar otra capa de agrupación maxima con un tamaño de agrupación de 2x2
```

Figura 4.18 Capa de agrupación máxima

Fuente: Elaboración propia

La línea de código menciona que se utiliza la función *add* del objeto *model* para agregar una capa de agrupación máxima (Maxpooling2D) al modelo. Esta capa se utiliza para combinadas en redes neuronales convolucionales para reducir la dimensionalidad de las características extraídas por las capas convolucionales anteriores.

La función *Maxpooling2D* realiza una operación de agrupación máxima en una matriz de características. En este caso, se especifica un tamaño de agrupación de 2x2 mediante el argumento *pool_size=(2, 2).*

La capa de agrupación máxima es útil porque ayuda a mejorar la invariancia a pequeñas traslaciones y deformaciones en las características extraídas. Al seleccionar el valor máximo de cada región, la capa de agrupación máxima presérvalas características más destacadas y descarta las menos relevantes.

4.2.2.6 Capa dropout

Se utilizo esta técnica de regularización en las redes neuronales para prevenir el sobreajuste durante el entrenamiento del modelo. Esto significa que, durante el entrenamiento, algunas neuronas no contribuirán a la propagación hacia adelante ni a la propagación hacia atrás de los gradientes. Esto ayuda a evitar que las neuronas se vuelvan dependientes de otras específicas y promueve una representación más robusta y generalizable de los datos.

Como se observa en la siguiente Figura 4.19 la capa Dropout que se agrega tiene una tasa de desactivación del 50% después de la primera capa oculta. Quiere decir que, durante el entrenamiento, el 50% de las neuronas en esa capa se desactivarán aleatoriamente en cada

```
model.add(Dropout(0.5)) # Agregar una capa de dropout con una tasa de dropout del 50%
```

Figura 4.19 Capa de abandono

Fuente: Elaboración propia

La línea de *model.add(Dropout(0.5))* agrega una capa de deserción al modelo, con una tasa de deserción del 50%. Esta capa ayuda a evitar el sobreajuste y mejora la generalización del modelo al desactivar aleatoriamente un porcentaje de las neuronas durante el entrenamiento.

4.2.2.7 Capa completamente conectada

Esta capa totalmente conectada, también conocida como Dense, permite que cada neurona realice una combinación lineal de las salidas de las neuronas de la capa anterior y aplica una función de activación no lineal para producir una salida. Se utiliza en la etapa final de la red neuronal, después de las capas convolucionales o de agrupación, para realizar la clasificación de los datos de entrada.

En el modelo se especifica una capa completamente conectada, está compuesta por 5 unidades, lo que significa que tendrá 5 neuronas. El número de unidades en esta capa es 5, corresponde a los niveles de severidad de degradación térmica en los datos de entrada, tales como: en buen estado, bajo, moderada, grave y extrema. En este caso la capa toma las características extraídas por las capas anteriores y las utiliza para asignar una probabilidad a cada clase posible. En la Figura 4.20 se establece la línea de código correspondiente a la capa completamente conectada.

```
model.add(Dense(5)) # Agregar otra capa completamente conectada con 5 unidades (correspondiente al número de clase)
```

Figura 4.20 Capa completamente conectada

Fuente: Elaboración propia

Cada unidad en la capa completamente conectada representa una clase y se encarga de calcular la probabilidad de que una imagen pertenezca a esa clase. Durante el entrenamiento, el modelo ajustara los pesos de las conexiones entre las neuronas para aprender a asignar las características de entrada a las clases correspondientes.

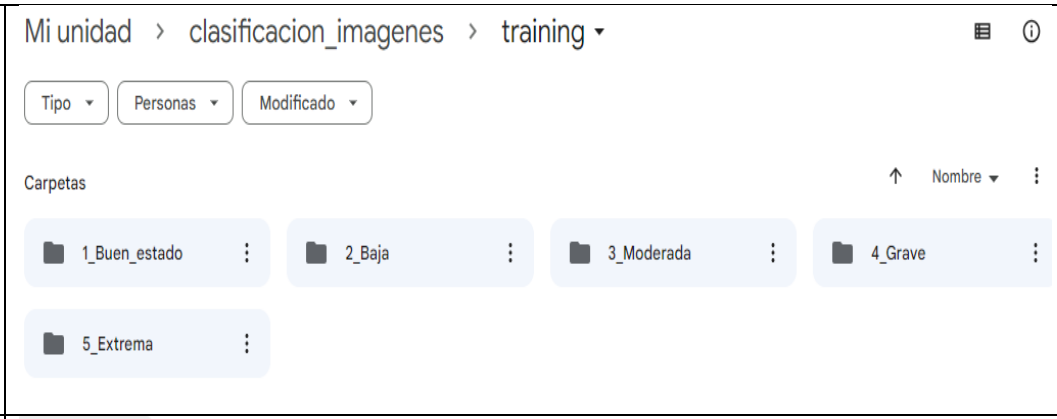
4.2.3 Entrenamiento y validación de la red

El entrenamiento y la validación son dos etapas fundamentales en el desarrollo de modelos de aprendizaje. Estas etapas se realizan para entrenar el modelo con datos de entrenamiento y evaluar su rendimiento con datos de validación.

4.2.3.1 Selección y preparación de conjunto de datos de entrenamiento y validación

Es importante destacar que el conjunto de datos de entrenamiento y de validación deben ser representativos de la distribución con la que se espera que el modelo funcione de manera adecuada. Lo anterior garantiza que el modelo pueda generalizar bien a datos nuevos y no solo memorice los ejemplos en el conjunto de entrenamiento. En la Figura 4.21 se muestra las dos carpetas que necesitamos en este desarrollo.

a)



b)

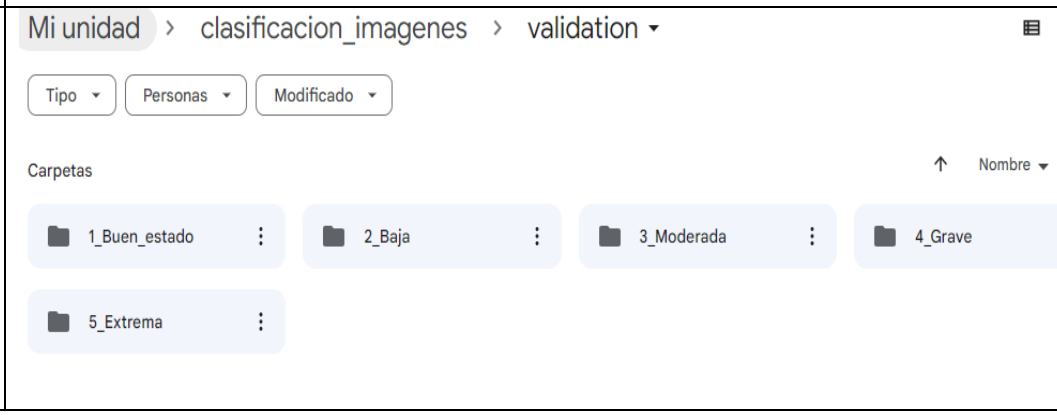


Figura 4.21 a) Datos de entrenamiento, b) Datos de validación

Fuente: Elaboración propia

Durante el entrenamiento, el modelo se ajusta a los datos de entrenamiento para aprender a realizar predicciones precisas. En esta etapa, se presenta al modelo ejemplos de entrada junto con salidas esperadas, y el modelo ajusta sus parámetros internos para minimizar la diferencia entre las predicciones y las salidas esperadas.

4.2.3.2 Implementación de la arquitectura de red neuronal convolucional adecuada.

En esta sección de implementa una arquitectura de red neuronal convolucional adecuada para abordar el problema de clasificación planteado. En particular, se desarrolla un modelo convolucional 2D capaz de clasificar las imágenes de termografía.

Para la elaboración de nuestro modelo importamos las bibliotecas necesarias para el código. Tensorflow es una biblioteca para el aprendizaje profundo. *ImageDataGenerator* se utiliza para generar lotes de imágenes para el entrenamiento del modelo. *Sequential* es una clase de modelo de *keras* que se utiliza para construir modelos de redes neuronales secuenciales. *matplotlib.pyplot* es utilizada para crear visualizaciones graficas.. Como se muestra en la Figura 4.22.

```
import tensorflow as tf # Importamos das bibliotecas necesarias
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D
from tensorflow.keras.layers import Activation, Dropout, Flatten, Dense
import matplotlib.pyplot as plt
```

Figura 4.22 Importar biblioteca para vincular con Google Drive

Fuente: Elaboración propia

En la siguiente Figura 4.23, se definen las rutas de los directorios de datos de entrenamiento y validación. Estas variables almacenan las rutas de los directorios donde se encuentran las imágenes que se utilizan para entrenar y validar un modelo de clasificación de imágenes.

```
train_data_dir = 'drive/MyDrive/clasificacion_imagenes/training' # Directorio de datos de entrenamiento
validation_data_dir = 'drive/MyDrive/clasificacion_imagenes/validation' # Directorio de datos de validación
```

Figura 4.23 Directorio de entrenamiento y validación

Fuente: Elaboración propia

En estas líneas de código, se define algunas variables importantes para el entrenamiento de un modelo de clasificación de imágenes. Como se visualiza en la Figura 4.24.

```
epochs = 50 # Número de épocas de entrenamiento
batch_size = 16 # Tamaño del lote (batch size)
img_width, img_height = 150, 150 # Ancho y alto de las imágenes
```

Figura 4.24 Definir variables

Fuente: Elaboración propia

- *epochs* representa el número de épocas de entrenamiento. Una época es una pasada completa por todo el conjunto de datos de entrenamiento. En este caso, se establece en 50, lo que significa que el modelo se entrenara durante 50 épocas.

- *batch_size* representa el tamaño del lote, es decir, el numero de muestras de entrenamiento que se utilizarán en cada iteración del entrenamiento. En este caso, se establece en 16, lo que significa que se utilizaran 16 imágenes en cada iteración del entrenamiento.
- *Img_width y img_height* representan el ancho y alto de las imágenes de entrada. Se establece en 150 pixeles para asegurar que todas las imágenes tengan el mismo tamaño antes de ser alimentadas al modelo.

En la Figura 4.25, se crea generadores de datos de imágenes para el conjunto de entrenamiento y el conjunto de validación. Estos generadores se utilizan para cargar y preprocesar las imágenes antes de alimentarlas al modelo de clasificación.

```
#Crear un generador de datos de imágenes
train_datagen = ImageDataGenerator(
    rescale=1.0/255, # Escalar los valores de pixeles en el rango de [0,1]
    rotation_range=30, # Rango de rotación aleatoria de las imágenes en grados
    width_shift_range=0.2, # Rango de desplazamiento horizontal aleatorio de las imágenes como fraccion de la anchura total
    height_shift_range=0.2, # Rango de desplazamiento vertical aleatorio de las imagenes como fraccion de la altura total
    shear_range=0.2, # Rango de deformación de cizallamiento aleatoria
    zoom_range=0.2, # Rango de zoom aleatorio de las imágenes
    horizontal_flip=True, # Voltear horizontalmente las imágenes de forma aleatoria
    fill_mode='nearest') #Estrategia de relleno para los pixeles que se crean durante las transformaciones

validation_datagen = ImageDataGenerator(rescale=1.0/255)
```

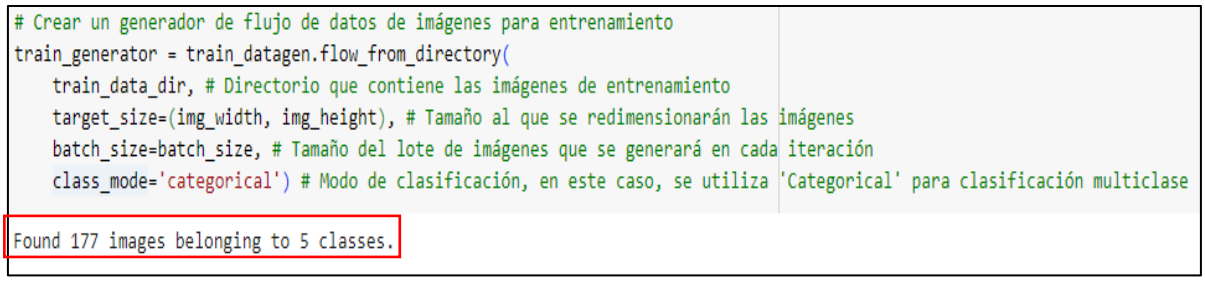
Figura 4.25 Creación de generadores de datos de imágenes

Fuente: Elaboración propia

- *train_datagen* Es un generador de datos de imágenes para el conjunto de entrenamiento. Se configura con varias transformaciones de aumento de datos, como la escala de los valores de pixeles en el rango de [0,1], la rotación aleatoria de imágenes en un rango de 30 grados, el desplazamiento horizontal y vertical aleatorio de las imágenes, la deformación de cizallamiento aleatorio, el zoom aleatorio de las imágenes, el volteo horizontal aleatorio de las imágenes y la estrategia de relleno para los pixeles que se crean durante las transformaciones.
- *validation_datagen* Es un generador de datos de imágenes para el conjunto de validación. Se configura simplemente con la escala de los valores de pixeles en el rango de [0,1].

Estos generadores de datos de imágenes son utilizados posteriormente para cargar y preprocesar las imágenes durante el entrenamiento y la validación del modelo de clasificación de imágenes.

En la Figura 4.26 se muestra la creación de un generador de flujos de datos de imágenes para entrenamiento para el conjunto de entrenamiento. Este generador se utiliza para cargar y procesar las imágenes de entrenamiento de forma eficiente durante el entrenamiento de modelo de clasificación.



```
# Crear un generador de flujo de datos de imágenes para entrenamiento
train_generator = train_datagen.flow_from_directory(
    train_data_dir, # Directorio que contiene las imágenes de entrenamiento
    target_size=(img_width, img_height), # Tamaño al que se redimensionarán las imágenes
    batch_size=batch_size, # Tamaño del lote de imágenes que se generará en cada iteración
    class_mode='categorical') # Modo de clasificación, en este caso, se utiliza 'Categorical' para clasificación multiclase

Found 177 images belonging to 5 classes.
```

Figura 4.26 Creación de un generador de datos de imágenes para el entrenamiento.

Fuente: Elaboración propia

- *train_generador* es generador de flujo de datos de imágenes para el conjunto de entrenamiento. Se crea utilizando el método *flow_from_directory* del generador de datos de imágenes *train_datagen*. Este método carga las imágenes del directorio especificando en *train_data_dir* y las procesadas según las configuraciones establecidas en *train_datagen*.
- *train_data_dir* es el directorio que contiene las imágenes de entrenamiento. Especifique la ubicación de las imágenes que se utilizan para entrenar el modelo.
- *target_size* especifica el tamaño al que se redimensionarán las imágenes. En este caso, se establece (*img_width, img_height*) para asegurar que todas las imágenes tengan el mismo tamaño antes de ser alimentadas al modelo de clasificación.
- *batch_size* especifica el tamaño del lote de imágenes que se generará en cada iteración. En este caso, se establece en *batch_size*, que es el valor definido anteriormente.
- *class_mode* especifica el modo de clasificación. En este caso, se utiliza '*categorical*' para clasificación multiclase, lo que significa que las etiquetas de las clases se generarán en formato de codificación.

En la siguiente Figura 4.27 se visualiza la creación de un generador de datos de imágenes para la validación.

```
# Crear un generador de flujo de datos de imágenes para validación
validation_generator = validation_datagen.flow_from_directory(
    validation_data_dir, # Directorio que contiene las imágenes de validación
    target_size=(img_width, img_height), # Tamaño al que se redimensionarán las imágenes
    batch_size=batch_size, # Tamaño del lote de imágenes que se generara en cada iteración
    class_mode='categorical') #Modo de clasificación, en este caso, se utiliza 'Categorical' para clasificacion multiclase

Found 69 images belonging to 5 classes.
```

Figura 4.27 Creación de un generador de datos de imágenes para la validación

Fuente: Elaboración propia

En estas líneas de código, se esta creando un generador de flujo de datos de imágenes para el conjunto de validación. Este generador se utiliza para cargar y preprocesar las imágenes de validación de forma eficiente durante la evaluación del modelo de clasificación.

- *validation_generatores* el generador de flujo de datos de imágenes para el conjunto de validación. Se crea utilizando el método *flow_from_directory* del generador de datos de imágenes *validation_datagen*. Este método carga las imágenes del directorio especificado en *validation_data_diry* las preprocesadas según las configuraciones establecidas en *validation_datagen*.
- *validation_data_dires* el directorio que contiene las imágenes de validación. Especifique la ubicación de las imágenes que se utilizarán para evaluar el modelo.
- *target_size* especifica el tamaño al que se redimensionarán las imágenes. En este caso, se establece (*img_width, img_height*) para asegurar que todas las imágenes tengan el mismo tamaño antes de ser alimentadas al modelo de clasificación.
- *batch_size* especifica el tamaño del lote de imágenes que se generará en cada iteración. En este caso, se establece en *batch_size*, que es el valor definido anteriormente.
- *class_mode* especifica el modo de clasificación. En este caso, se utiliza '*categorica*l' para clasificación multiclase, lo que significa que las etiquetas de las clases se generarán en formato de codificación.

Como resultado del generador de datos para el entrenamiento encontró 177 imágenes pertenecientes a 5 clases y del generador de datos de imágenes para la validación encontró 69 imágenes pertenecientes a 5 clases. Como se puede visualizar en la Figura 4.28 se utilizo el 72% de datos para el entrenamiento y el 28% para la validación.

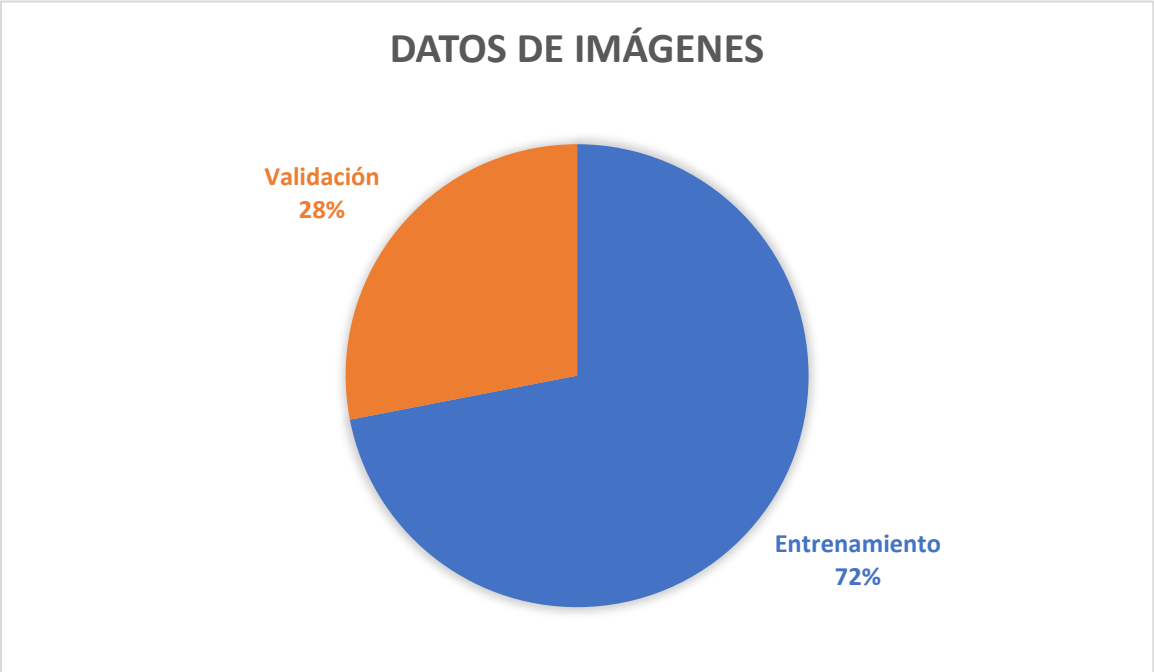


Figura 4.28 Datos encontrados por los generadores de datos.

Fuente: Elaboración propia

Se crea un modelo secuencial de redes neuronales convolucionales para la clasificación de imágenes. Como se muestra en la Figura 4.29.

```
model = Sequential() # Crear un modelo secuencial

model.add(Conv2D(32, (3, 3), input_shape=(img_width, img_height, 3))) # Agregar una capa de convolución con 32 filtros de tamaño 3x3 y esp. la forma de entrada
model.add(Activation('relu')) #Agregar una capa de activación ReLu
model.add(MaxPooling2D(pool_size=(2, 2))) # Agregar una capa de agrupación maxima con un tamaño de agrupación de 2x2

model.add(Conv2D(64, (3, 3))) # Agregar otra capa de convolución con 64 filtros de tamaño 3x3
model.add(Activation('relu')) #Agregar una capa de activación ReLu
model.add(MaxPooling2D(pool_size=(2, 2))) # Agregar otra capa de agrupación maxima con un tamaño de agrupación de 2x2

model.add(Conv2D(128, (3, 3))) # Agregar otra capa de convolución con 128 filtros de tamaño 3x3
model.add(Activation('relu')) #Agregar una capa de activación ReLu
model.add(MaxPooling2D(pool_size=(2, 2))) # Agregar otra capa de agrupación maxima con un tamaño de agrupación de 2x2

model.add(Flatten()) # Aplanar la salida de las capas anteriores
model.add(Dense(64)) # Agregar una capa totalmente conectada con 64 unidades
model.add(Activation('relu')) #Agregar una capa de activación ReLu
model.add(Dropout(0.5)) # Agregar una capa de dropout con una tasa de dropout del 50%
model.add(Dense(5)) # Agregar otra capa completamente conectada con 5 unidades (correspondiente al número de clase)
model.add(Activation('softmax'))# Agregar una capa de activación softmax
```

Figura 4.29 Creación de un modelo secuencial de redes neuronales convolucionales

Fuente: Elaboración propia

- model = Sequential() crea un modelo secuencial vacío donde se pueden ir agregando capas una tras otra.
- model.add(Conv2D(32, (3, 3), input_shape=(img_width, img_height, 3))) Agrega una capa de convolución con 32 filtros de tamaño 3x3 y una forma de entrada especificada por input_shape. Esta capa se encarga de extraer características de las imágenes.

- model.add(Activation('relu')) agregue una capa de activación ReLU (Rectified Linear Unit) que introduce la no linealidad en el modelo.
- model.add(MaxPooling2D(pool_size=(2, 2))) agregue una capa de agrupación máxima con un tamaño de agrupación de 2x2. Esta capa reduce la dimensionalidad de las características extraídas.

Estos pasos se repiten dos veces más con capas adicionales de convolución, activación ReLU y agrupación para extraer características más complejas.

- model.add(Flatten()) aplanar la salida de las capas anteriores para prepararla para la capa de clasificación final.
- model.add(Dense(64)) agregue una capa totalmente conectada con 64 unidades. Esta capa realiza la clasificación basada en las características extraídas.
- model.add(Activation('relu')) se agrega una capa de activación ReLU para introducir la no linealidad en la capa de clasificación.
- model.add(Dropout(0.5)) se agrega una capa de deserción con una tasa de deserción del 50%. Esto ayuda a prevenir el sobreajuste al apagar aleatoriamente algunas neuronas durante el entrenamiento.
- model.add(Dense(5)) agregue otra capa completamente conectada con 5 unidades, que corresponde al número de clases en la clasificación.
- model.add(Activation('softmax')) agregue una capa de activación softmax para obtener las probabilidades de cada clase en la salida final del modelo.

En las siguientes líneas de código que se muestra en la Figura 4.30 se esta compilando el modelo antes de entrenarlo. La compilación del modelo implica la configuración de la función de perdida, el optimizador y las métricas que se utilizaron durante el entrenamiento y la evaluación del modelo.

```
model.compile(loss='categorical_crossentropy', # Compilar el modelo con una función de pérdida de entropía cruzada categórica
              optimizer='adam', # Utilizar el optimizador Adam para ajustar los pesos del modelo durante el entrenamiento
              metrics=['accuracy']) # Calcular la metrica de precisión durante el entrenamiento y la evaluación del modelo
```

Figura 4.30 Compilación del modelo

Fuente: Elaboración propia

- loss='categorical_crossentropy'establece la función de pérdida del modelo en la entropía cruzada categórica. Esta función de pérdida es utilizada en

problemas de clasificación multiclase, donde las etiquetas de las clases se codifican.

- optimizer='adam'configure el optimizador del modelo en Adam. Adam es un algoritmo de optimización popular y eficiente que ajusta los pesos del modelo durante el entrenamiento para minimizar la función de pérdida.
- metrics=['accuracy']especifica que se calculará la métrica de precisión durante el entrenamiento y la evaluación del modelo. La precisión es una métrica utilizada para evaluar el rendimiento de los modelos de clasificación.

Como podemos ver en la Figura 4.31 se está entrenando el modelo utilizando los generadores de flujo de datos de imágenes de entrenamiento y validación.

```
history = model.fit(train_generator, # Entrenar el modelo utilizando el generador de flujo de datos de imágenes de entrenamiento
                    steps_per_epoch=train_generator.samples // batch_size, #Especificar el número de pasos por época basado en el tamaño del lote y el número tot
                    epochs=epochs, # Especificar el número de épocas de entrenamiento
                    validation_data=validation_generator, # Utilizar el generador de flujo de datos de imágenes de validación para la validación durante el entre
                    validation_steps=validation_generator.samples // batch_size) # Especificar el número de pasos de validación basado en el tamaño del lote y el

model.save('image_model_v1.h5') # Guardar el modelo entrenado en un archivo h5
```

Figura 4.31 Entrenamiento del modelo

Fuente: Elaboración propia

- history = model.fit(train_generator, ...) realice el entrenamiento del modelo utilizando el generador de flujo de datos de imágenes de entrenamiento train_generator. El método fit ajusta los pesos del modelo utilizando los datos de entrenamiento y realiza un número especificado de épocas de entrenamiento.
- steps_per_epoch=train_generator.samples // batch_sizeespecifica el número de pasos por época de entrenamiento. Se calcula dividiendo el número total de muestras de entrenamiento (train_generator.samples) por el tamaño del lote (batch_size). Esto asegura que todas las muestras de entrenamiento se utilicen en cada época.
- epochs=epochsespecifica el número de épocas de entrenamiento. Epochs es una variable que contiene el número de épocas definido anteriormente.
- validation_data=validation_generatorutilice el generador de flujo de datos de imágenes de validación validation_generatorpara realizar la validación durante el entrenamiento. Esto permite evaluar el rendimiento del modelo en datos no vistos durante el entrenamiento.

- validation_steps=validation_generator.samples // batch_size especifica el número de pasos de validación. Se calcula dividiendo el número total de muestras de validación (validation_generator.samples) por el tamaño del lote (batch_size). Esto asegura que todas las muestras de validación se utilizan en cada paso de validación.

Despues de entrenar el modelo, se guarda en un archivo h5 utilizando la siguiente línea de código: model.save(‘image_model_v1.h5’), esto guarda el modelo entrenado en un archivo llamado “image_model_v1.h5” en el sistema de archivos.

Con la siguiente Figura 4.32 se detalla el análisis del entrenamiento de la red neuronal.

```
acc = history.history['accuracy'] # Obtener el historial de precisión de entrenamiento
val_acc = history.history['val_accuracy'] # Obtener el historial de precisión de validación
loss = history.history['loss'] # Obtener el historial de pérdida de entrenamiento
val_loss = history.history['val_loss'] # Obtener el historial de pérdida de validación

epochs_range = range(1, epochs + 1) # Crear un rango de épocas para el eje x del gráfico

plt.figure(figsize=(12, 4)) # Crear una figura de tamaño 12x4
plt.subplot(1, 2, 1) # Crear un subplot en la posición 1 de una fila y dos columnas
plt.plot(epochs_range, acc, label='Accuracy') # Graficar la precisión de entrenamiento en función de las epocas
plt.plot(epochs_range, val_acc, label='Validation Accuracy') # Graficar la precisión de validación en función de las epocas
plt.legend(loc='lower right') # Mostrar una leyenda en la esquina inferior derecha del gráfico
plt.title('Accuracy vs. Validation Accuracy') #Establecer el título del gráfico
plt.xlabel('Epoch') # Etiquetar el eje x del gráfico como 'Epoch'
plt.ylabel('Accuracy') # Etiquetar el eje y del gráfico como 'Accuracy'

plt.subplot(1, 2, 2) # Crear un subplot en la precisión 2 de una fila y dos columnas
plt.plot(epochs_range, loss, label='Loss') # Graficar la pérdida de entrenamiento en función de las épocas
plt.plot(epochs_range, val_loss, label='Validation Loss') # Graficar la pérdida de validación en función de las épocas
plt.legend(loc='upper right') # Mostrar una leyenda en la esquina superior derecha del gráfico
plt.title('Loss vs. Validation Loss') # Establecer el título del gráfico
plt.xlabel('Epoch') # Etiquetar el eje x del gráfico como 'Epoch'
plt.ylabel('Loss') # Etiquetar el eje y del gráfico como 'Loss'

plt.show() # Mostrar el gráfico
```

Figura 4.31 Análisis del entrenamiento de la red neuronal

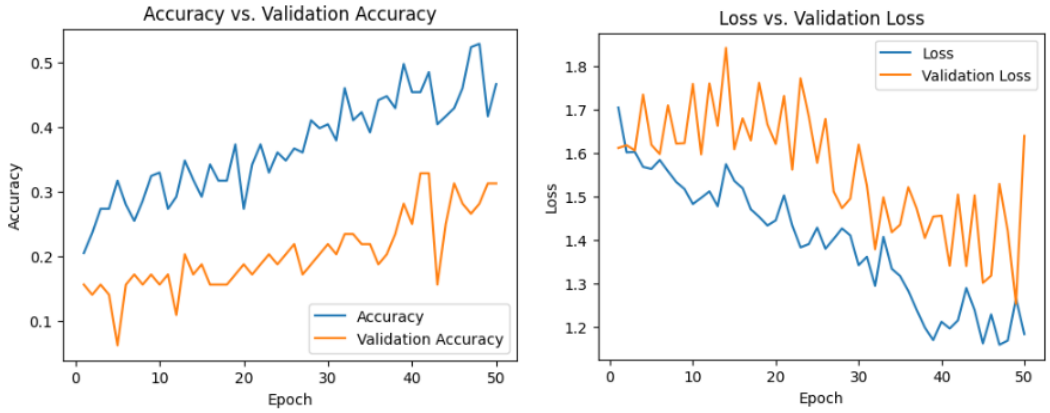
Fuente: Elaboración propia

El código entrena una red neuronal convolucional para la clasificación de imágenes durante 50 épocas y muestra métricas como la pérdida (loss) y la precisión (accuracy) para cada época tanto para el conjunto de entrenamiento como para el de validación. Cada época implica entrenar la red con todo el conjunto de entrenamiento y evaluarla en el conjunto de validación. Después de cada época se muestran las métricas. En general vemos que a medida que aumentan las épocas:

- La pérdida en el conjunto de entrenamiento va disminuyendo, esto significa que el modelo está aprendiendo y reduciendo su error.
- La precisión en el conjunto de entrenamiento va aumentando, esto significa que el modelo está clasificando mejor las imágenes de entrenamiento.

- La pérdida y precisión en el conjunto de validación muestran fluctuaciones, lo que significa que el modelo no está sobreajustado. Un sobreajuste se vería como una disminución continua de la pérdida de validación, pero un estancamiento o disminución en la precisión de validación.
- Los resultados muestran que el modelo logra alrededor de un 50% de precisión de validación al final de las 50 épocas, lo que indica que está aprendiendo a clasificar razonablemente bien las imágenes, pero aún queda margen de mejora. Se podrían probar más épocas de entrenamiento, usar lotes más grandes, agregar más capas o abandonos para mejorar el rendimiento.

Como resultado en la Figura 4.32, se puede observar que el modelo de aprendizaje profundo puede mejorar su eficiencia con una información mucho más extensa, en la gráfica a) se trata sobre la precisión o exactitud que debería reflejar en un 0,8 o 0,9, esto se debe a que no se dispone de mucha información, ya que los modelos de aprendizaje profundo necesitan una data mucho más extensa. Mientras la gráfica b) es muy importante ya que la perdida de la validación se mantiene en 1,4 debido a su poco data, no es un resultado positivo, con más información permitirá obtener un mejor modelo dando como resultado una pérdida de validación menor a 0,5.



a) **b)**

Figura 4.32 a) Gráfico de la precisión del entrenamiento en función de las épocas, b) Gráfico de la precisión de validación en función de las épocas.

Fuente: Elaboración propia

Este modelo realiza predicciones, en la Figura 4.33 se detalla el código que realiza algunas operaciones de evaluación y cálculo de métricas utilizando un modelo entrenado y un conjunto de validad.

```
validation_predictions = model.predict(validation_generator) #
Realizar predicciones en el conjunto de validación utilizando el
modelo entrenado
true_labels = validation_generator.classes # Obtener las etiquetas
verdaderas del conjunto de validación

rmse = np.sqrt(np.mean((validation_predictions.argmax(axis=1) -
true_labels) ** 2))  # Calcular la raíz del error cuadrático medio
(RMSE) entre las predicciones y las etiquetas verdaderas
print(f'RMSE: {rmse:.2f}') # Imprimir el valor de RMSE formateado con
dos decimales

mae = np.mean(np.abs(validation_predictions.argmax(axis=1) -
true_labels)) # Calcular el error absoluto medio (MAE) entre las
predicciones y las etiquetas verdaderas
print(f'MAE: {mae:.2f}') # Imprimir el valor del MAE formateado con
dos decimales
```

Figura 4.33 Creación del modelo de predicciones.

Fuente: Elaboración propia

- *validation_predictions* se utiliza en el modelo de entrenamiento y el generador de validación para realizar predicciones en el conjunto de validación.
- *true_labels* se obtiene las etiquetas verdaderas del conjunto de validación.
- Se calcula la raíz del error cuadrático medio (RMSE) entre las predicciones y las etiquetas verdaderas. Se utiliza la función *np.sqrt* de la biblioteca NumPy para calcular la raíz cuadrada y *np.mean* para obtener el promedio de los errores cuadráticos. El resultado se guarda en la variable *rmse.*
- Se imprime el valor de RMSE formateado con dos decimales utilizando la función *print.*
- Se calcula el error absoluto medio (MAE) entre las predicciones y las etiquetas verdaderas. Se utiliza la función de *np.abs* de NumPy para obtener el valor absoluto de la diferencia entre las predicciones y las etiquetas verdaderas.
- Se imprime el calor de MAE formateado con dos decimales utilizando la función *print.*
- El valor de RMSE y de MAE se muestra en la Figura 4.34.

```
5/5 [==============================] - 1s 70ms/step
RMSE: 1.57
MAE: 1.35
```

Figura 4.34 Valor de error cuadrático medio y el error absoluto

Fuente: Elaboración propia

Se realiza la clasificación de imágenes utilizando un modelo previamente entrenado. El objetivo de este es determinar a qué clase de severidad de degradación térmica pertenece la imagen. A continuación, se detalla en la Figura 4.35.

```
new_image_path =
'drive/MyDrive/clasificacion_imagenes/test/2.Bajo/06965.PNG'  # Ruta
de la imagen

img = image.load_img(new_image_path, target_size=(150, 150)) #Cargar
la imagen utilizando la función load_img de keras y redimensionarla a
una tamaño de 150x150 píxeles
img_array = image.img_to_array(img) # Convertir la imagen en un
arreglo de numpy
img_array = np.expand_dims(img_array, axis=0) # Agregar una dimensión
adicional al arreglo para que coincida con el formato de entrada del
modelo
img_array /= 255.0 # Normalizar los valores de los pixeles de la
imagen dividiendolos por 255.0 para que estén en el rango de [0,1]

predictions = model.predict(img_array) # Realizar predicciones en la
imagen utilizando el modelo cargado

class_index = np.argmax(predictions) # Obtener el índice de la clase
con mayor probabilidad de predicción
classes = ['buen estado', 'bajo', 'moderada', 'grave', 'extremo'] #
Definir las clases posibles de predicción
predicted_class = classes[class_index] # Obtener la clase predicha
correspondiente al índice

print(f'La imagen se clasifica como: {predicted_class}') # Imprimir la
clase predicha de la imagen
```

Figura 4.35 Modelo entrenado de predicción

Fuente: Elaboración propia

4.3 PRUEBAS Y RESULTADOS

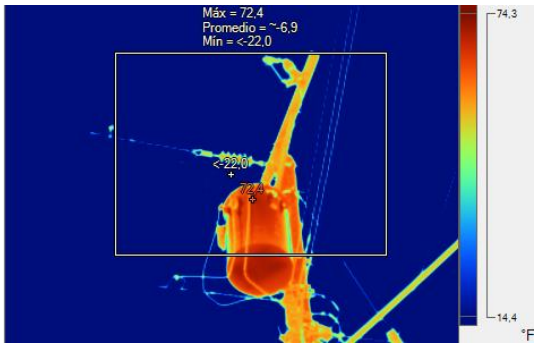
En el transcurso del desarrollo de esta tesis, se desarrolla el análisis termográfico y un modelo de predicción. En una primera etapa, se elaboró un análisis termográfico de los transformadores de distribución, una vez que se confirmó la adecuada base de datos, se realizó el modelo de predicción como se detalló anteriormente.

4.3.1 Evaluación de la severidad de la degradación de los aislamientos.

El objetivo de la evaluación es determinar la gravedad de la degradación de los aislamientos y tomar medidas correctivas adecuadas. Esto puede implicar la reparación o reemplazo de los aislamientos dañados, así como la implementación de medidas preventivas para evitar futuros problemas.

4.3.1.1 CIRCUITOS EN BUEN ESTADO:

Se realiza pruebas en transformadores de distribución, por medio del cual se ingresa la información a través del modelo para que brinde buena eficiencia em la clasificación sobre niveles de severidad en los circuitos del transformador, tal como se observa en la siguiente Figura 4.36.



a)

```
new_image_path = 'drive/MyDrive/clasificacion_imagenes/validation/1_Buen_estado/06918.PNG'  # Ruta de la imagen

img = image.load_img(new_image_path, target_size=(150, 150))
img_array = image.img_to_array(img)
img_array = np.expand_dims(img_array, axis=0)
img_array /= 255.0

predictions = model.predict(img_array)

class_index = np.argmax(predictions)
classes = ['buen estado', 'bajo', 'moderada', 'grave', 'extremo']
predicted_class = classes[class_index]

print(f'La imagen se clasifica como: {predicted_class}')
```

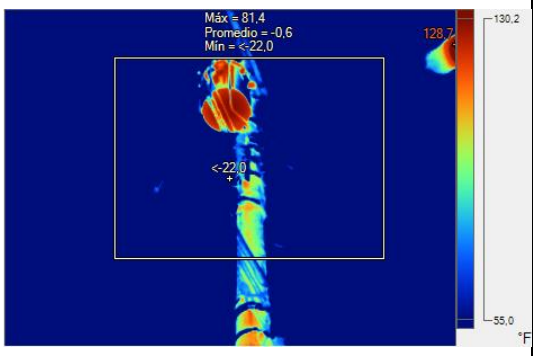
b)

```
1/1 [==============================] - 0s 21ms/step
La imagen se clasifica como: buen estado
```

c)

Figura 4.36 Prueba IR 06918 a) análisis termográfico, b) predicción en el modelo realizado en Python, c) Resultado de la predicción

Fuente: Elaboración propia



a)

```python
new_image_path = 'drive/MyDrive/clasificacion_imagenes/validation/1_Buen_estado/06921.PNG'  # Ruta de la imagen

img = image.load_img(new_image_path, target_size=(150, 150))
img_array = image.img_to_array(img)
img_array = np.expand_dims(img_array, axis=0)
img_array /= 255.0

predictions = model.predict(img_array)

class_index = np.argmax(predictions)
classes = ['buen estado', 'bajo', 'moderada', 'grave', 'extremo']
predicted_class = classes[class_index]

print(f'La imagen se clasifica como: {predicted_class}')
```

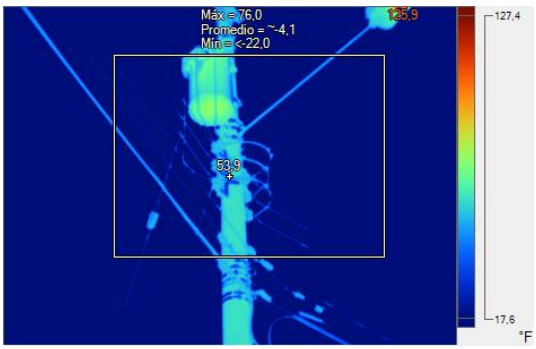
b)

```
1/1 [==============================] - 0s 18ms/step
La imagen se clasifica como: buen estado
```

c)

Figura 4.37 Prueba IR 06921 a) análisis termográfico, b) predicción en el modelo realizado en Python, c) Resultado de la predicción

Fuente: Elaboración propia



a)

```python
new_image_path = 'drive/MyDrive/clasificacion_imagenes/test/1_Buen_estado/06932.PNG'  # Ruta de la imagen

img = image.load_img(new_image_path, target_size=(150, 150)) #Cargar la imagen utilizando la función load_img de keras y redimensionarla a una tamaño de 150x150 pixeles
img_array = image.img_to_array(img) # Convertir la imagen en un arreglo de numpy
img_array = np.expand_dims(img_array, axis=0) # Agregar una dimensión adicional al arreglo para que coincida con el formato de entrada del modelo
img_array /= 255.0 # Normalizar los valores de los pixeles de la imagen dividiéndolos por 255.0 para que estén en el rango de [0,1]

predictions = model.predict(img_array) # Realizar predicciones en la imagen utilizando el modelo cargado

class_index = np.argmax(predictions) # Obtener el índice de la clase con mayor probabilidad de predicción
classes = ['buen estado', 'bajo', 'moderada', 'grave', 'extremo'] # Definir las clases posibles de predicción
predicted_class = classes[class_index] # Obtener la clase predicha correspondiente al índice

print(f'La imagen se clasifica como: {predicted_class}') # Imprimir la clase predicha de la imagen
```

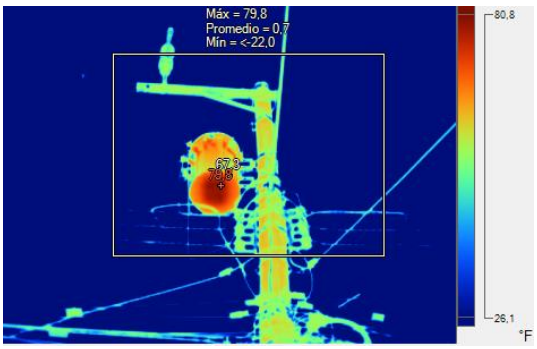
b)

```
1/1 [==============================] - 0s 17ms/step
La imagen se clasifica como: buen estado
```

c)

Figura 4.38 Prueba IR 06932 a) análisis termográfico, b) predicción en el modelo realizado en Python, c) Resultado de la predicción

Fuente: Elaboración propia



a)

```
new_image_path = 'drive/MyDrive/clasificacion_imagenes/test/1_Buen_estado/06939.PNG'  # Ruta de la imagen

img = image.load_img(new_image_path, target_size=(150, 150)) #Cargar la imagen utilizando la función load_img de keras y redimensionarla a una tamaño de 150x150 pixeles
img_array = image.img_to_array(img) # Convertir la imagen en un arreglo de numpy
img_array = np.expand_dims(img_array, axis=0) # Agregar una dimensión adicional al arreglo para que coincida con el formato de entrada del modelo
img_array /= 255.0 # Normalizar los valores de los pixeles de la imagen dividiendolos por 255.0 para que estén en el rango de [0,1]

predictions = model.predict(img_array) # Realizar predicciones en la imagen utilizando el modelo cargado

class_index = np.argmax(predictions) # Obtener el índice de la clase con mayor probabilidad de predicción
classes = ['buen estado', 'bajo', 'moderada', 'grave', 'extremo'] # Definir las clases posibles de predicción
predicted_class = classes[class_index] # Obtener la clase predicha correspondiente al índice

print(f'La imagen se clasifica como: {predicted_class}') # Imprimir la clase predicha de la imagen
```

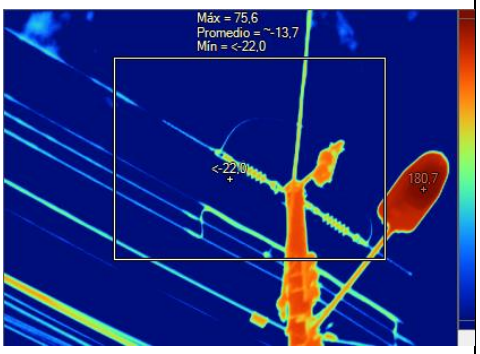
b)

```
1/1 [==============================] - 0s 19ms/step
La imagen se clasifica como: buen estado
```

c)

Figura 4.39 Prueba IR 06939 a) análisis termográfico, b) predicción en el modelo realizado en Python, c) Resultado de la predicción

Fuente: Elaboración propia



a)

```
new_image_path = 'drive/MyDrive/clasificacion_imagenes/test/1_Buen_estado/06946.PNG'  # Ruta de la imagen

img = image.load_img(new_image_path, target_size=(150, 150)) #Cargar la imagen utilizando la función load_img de keras y redimensionarla a una tamaño
img_array = image.img_to_array(img) # Convertir la imagen en un arreglo de numpy
img_array = np.expand_dims(img_array, axis=0) # Agregar una dimensión adicional al arreglo para que coincida con el formato de entrada del modelo
img_array /= 255.0 # Normalizar los valores de los pixeles de la imagen dividiendolos por 255.0 para que estén en el rango de [0,1]

predictions = model.predict(img_array) # Realizar predicciones en la imagen utilizando el modelo cargado

class_index = np.argmax(predictions) # Obtener el índice de la clase con mayor probabilidad de predicción
classes = ['buen estado', 'bajo', 'moderada', 'grave', 'extremo'] # Definir las clases posibles de predicción
predicted_class = classes[class_index] # Obtener la clase predicha correspondiente al índice

print(f'La imagen se clasifica como: {predicted_class}') # Imprimir la clase predicha de la imagen
```

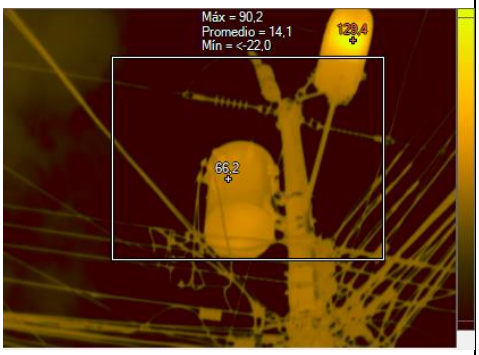
b)

```
1/1 [==============================] - 0s 18ms/step
La imagen se clasifica como: buen estado
```

c)

Figura 4.40 Prueba IR 06946 a) análisis termográfico, b) predicción en el modelo realizado en Python, c) Resultado de la predicción

Fuente: Elaboración propia



a)

```
new_image_path = 'drive/MyDrive/clasificacion_imagenes/test/1_Buen_estado/06948.PNG'  # Ruta de la imagen

img = image.load_img(new_image_path, target_size=(150, 150)) #Cargar la imagen utilizando la función load_img de keras y redimensionarla a una tamaño
img_array = image.img_to_array(img) # Convertir la imagen en un arreglo de numpy
img_array = np.expand_dims(img_array, axis=0) # Agregar una dimensión adicional al arreglo para que coincida con el formato de entrada del modelo
img_array /= 255.0 # Normalizar los valores de los pixeles de la imagen dividiendolos por 255.0 para que estén en el rango de [0,1]

predictions = model.predict(img_array) # Realizar predicciones en la imagen utilizando el modelo cargado

class_index = np.argmax(predictions) # Obtener el índice de la clase con mayor probabilidad de predicción
classes = ['buen estado', 'bajo', 'moderada', 'grave', 'extremo'] # Definir las clases posibles de predicción
predicted_class = classes[class_index] # Obtener la clase predicha correspondiente al índice

print(f'La imagen se clasifica como: {predicted_class}') # Imprimir la clase predicha de la imagen
```

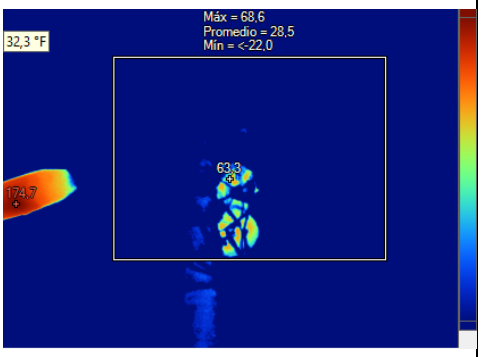
b)

```
1/1 [==============================] - 0s 30ms/step
La imagen se clasifica como: buen estado
```

c)

Figura 4.41 Prueba IR 06948 a) análisis termográfico, b) predicción en el modelo realizado en Python, c) Resultado de la predicción

Fuente: Elaboración propia



a)

```
new_image_path = 'drive/MyDrive/clasificacion_imagenes/test/1_Buen_estado/06952.PNG'  # Ruta de la imagen

img = image.load_img(new_image_path, target_size=(150, 150)) #Cargar la imagen utilizando la función load_img de keras y redimensionarla a una tamaño
img_array = image.img_to_array(img) # Convertir la imagen en un arreglo de numpy
img_array = np.expand_dims(img_array, axis=0) # Agregar una dimensión adicional al arreglo para que coincida con el formato de entrada del modelo
img_array /= 255.0 # Normalizar los valores de los pixeles de la imagen dividiendolos por 255.0 para que estén en el rango de [0,1]

predictions = model.predict(img_array) # Realizar predicciones en la imagen utilizando el modelo cargado

class_index = np.argmax(predictions) # Obtener el índice de la clase con mayor probabilidad de predicción
classes = ['buen estado', 'bajo', 'moderada', 'grave', 'extremo'] # Definir las clases posibles de predicción
predicted_class = classes[class_index] # Obtener la clase predicha correspondiente al índice

print(f'La imagen se clasifica como: {predicted_class}') # Imprimir la clase predicha de la imagen
```

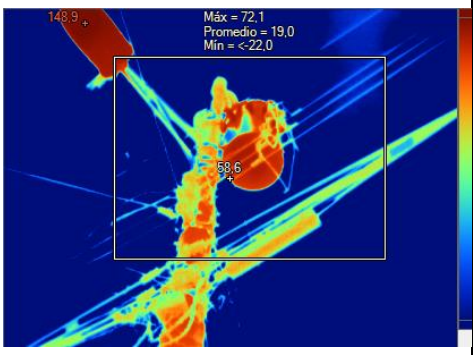
b)

```
1/1 [==============================] - 0s 45ms/step
La imagen se clasifica como: buen estado
```

c)

Figura 4.42 Prueba IR 06952 a) análisis termográfico, b) predicción en el modelo realizado en Python, c) Resultado de la predicción

Fuente: Elaboración propia



a)

```
new_image_path = 'drive/MyDrive/clasificacion_imagenes/test/1_Buen_estado/06958.PNG'  # Ruta de la imagen

img = image.load_img(new_image_path, target_size=(150, 150)) #Cargar la imagen utilizando la función load_img de keras y redimensionarla a una tamañ
img_array = image.img_to_array(img) # Convertir la imagen en un arreglo de numpy
img_array = np.expand_dims(img_array, axis=0) # Agregar una dimensión adicional al arreglo para que coincida con el formato de entrada del modelo
img_array /= 255.0 # Normalizar los valores de los pixeles de la imagen dividiendolos por 255.0 para que estén en el rango de [0,1]

predictions = model.predict(img_array) # Realizar predicciones en la imagen utilizando el modelo cargado

class_index = np.argmax(predictions) # Obtener el índice de la clase con mayor probabilidad de predicción
classes = ['buen estado', 'bajo', 'moderada', 'grave', 'extremo'] # Definir las clases posibles de predicción
predicted_class = classes[class_index] # Obtener la clase predicha correspondiente al índice

print(f'La imagen se clasifica como: {predicted_class}') # Imprimir la clase predicha de la imagen
```

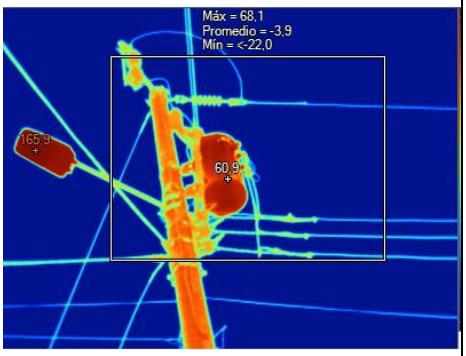
b)

```
1/1 [==============================] - 0s 75ms/step
La imagen se clasifica como: buen estado
```

c)

Figura 4.43 Prueba IR 06958 a) análisis termográfico, b) predicción en el modelo realizado en Python, c) Resultado de la predicción

Fuente: Elaboración propia



a)

```
new_image_path = 'drive/MyDrive/clasificacion_imagenes/test/1_Buen_estado/06961.PNG'  # Ruta de la imagen

img = image.load_img(new_image_path, target_size=(150, 150)) #Cargar la imagen utilizando la función load_img de keras y redimensionarla a una t
img_array = image.img_to_array(img) # Convertir la imagen en un arreglo de numpy
img_array = np.expand_dims(img_array, axis=0) # Agregar una dimensión adicional al arreglo para que coincida con el formato de entrada del model
img_array /= 255.0 # Normalizar los valores de los pixeles de la imagen dividiendolos por 255.0 para que estén en el rango de [0,1]

predictions = model.predict(img_array) # Realizar predicciones en la imagen utilizando el modelo cargado

class_index = np.argmax(predictions) # Obtener el índice de la clase con mayor probabilidad de predicción
classes = ['buen estado', 'bajo', 'moderada', 'grave', 'extremo'] # Definir las clases posibles de predicción
predicted_class = classes[class_index] # Obtener la clase predicha correspondiente al índice

print(f'La imagen se clasifica como: {predicted_class}') # Imprimir la clase predicha de la imagen
```

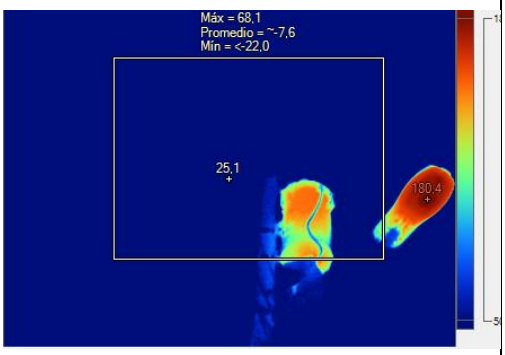
b)

```
1/1 [==============================] - 0s 17ms/step
La imagen se clasifica como: buen estado
```

c)

Figura 4.44 Prueba IR 06961 a) análisis termográfico, b) predicción en el modelo realizado en Python, c) Resultado de la predicción

Fuente: Elaboración propia



a)

```
new_image_path = 'drive/MyDrive/clasificacion_imagenes/test/1_Buen_estado/06966.PNG'  # Ruta de la imagen

img = image.load_img(new_image_path, target_size=(150, 150)) #Cargar la imagen utilizando la función load_img de keras y redimensionarla a una tamaño de 150x1
img_array = image.img_to_array(img) # Convertir la imagen en un arreglo de numpy
img_array = np.expand_dims(img_array, axis=0) # Agregar una dimensión adicional al arreglo para que coincida con el formato de entrada del modelo
img_array /= 255.0 # Normalizar los valores de los pixeles de la imagen dividiendolos por 255.0 para que estén en el rango de [0,1]

predictions = model.predict(img_array) # Realizar predicciones en la imagen utilizando el modelo cargado

class_index = np.argmax(predictions) # Obtener el índice de la clase con mayor probabilidad de predicción
classes = ['buen estado', 'bajo', 'moderada', 'grave', 'extremo'] # Definir las clases posibles de predicción
predicted_class = classes[class_index] # Obtener la clase predicha correspondiente al índice

print(f'La imagen se clasifica como: {predicted_class}') # Imprimir la clase predicha de la imagen
```

b)

```
1/1 [==============================] - 0s 72ms/step
La imagen se clasifica como: buen estado
```

c)

Figura 4.45 Prueba IR 06966 a) análisis termográfico, b) predicción en el modelo realizado en Python, c) Resultado de la predicción

Fuente: Elaboración propia

Por medio de la Tabla 8 se observa que el modelo presenta buena precisión, puesto que presenta un criterio de severidad nula.

Tabla 8. Pruebas de clasificación en buen estado.

Fotografía IR	Temperatura |°C|	Informe termográfico	Red neuronal Convolucional (CNN)
06918	15	En buen estado	En buen estado
06921	14.4	En buen estado	En buen estado
06932	16	En buen estado	En buen estado
06939	13.8	En buen estado	En buen estado
06946	17	En buen estado	En buen estado
06948	15.1	En buen estado	En buen estado
06952	13.5	En buen estado	En buen estado
06958	14.5	En buen estado	En buen estado

06961	15.1	En buen estado	En buen estado
06966	15.5	En buen estado	En buen estado

Fuente: Elaboración propia

Las pruebas de clasificación han demostrado una precisión y eficiencia cuando se trata de severidad nula. Los resultados obtenidos han demostrado que se puede llegar a clasificar en menos tiempo transformadores en los cuales no es necesario un mantenimiento de urgencia.

4.3.2 CIRCUITOS EN SEVERIDAD BAJA:

Se realiza pruebas sobre 3 transformadores de distribución, por medio del cual se ingresa la información a través del modelo para que brinde buena eficiencia em la clasificación sobre niveles de severidad.

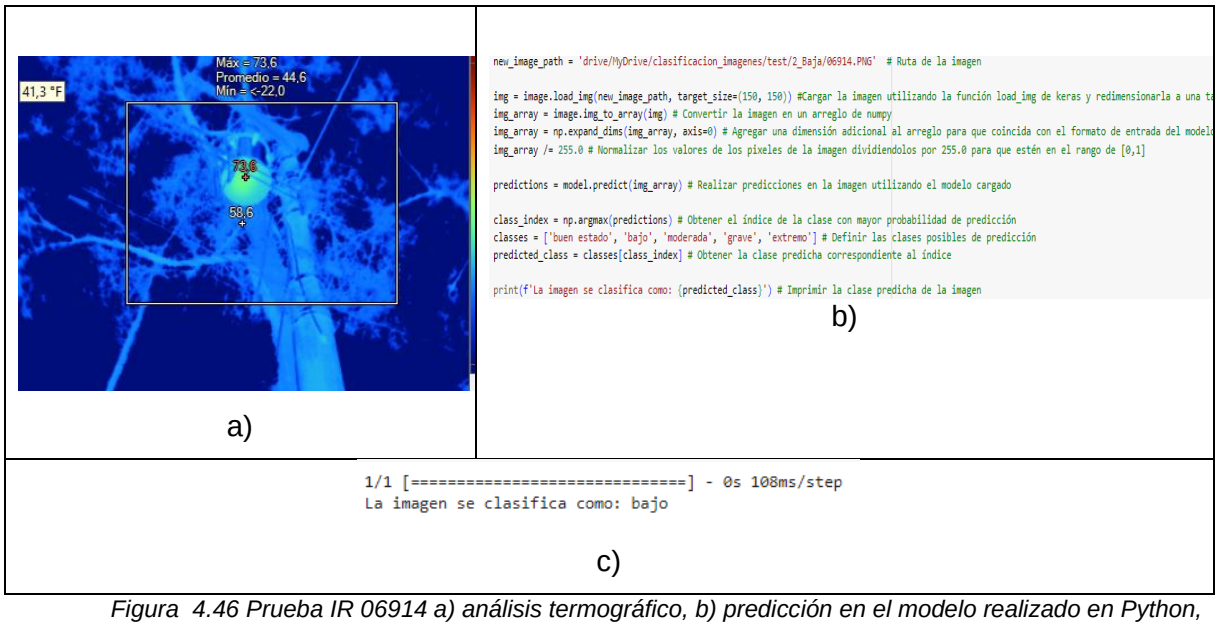
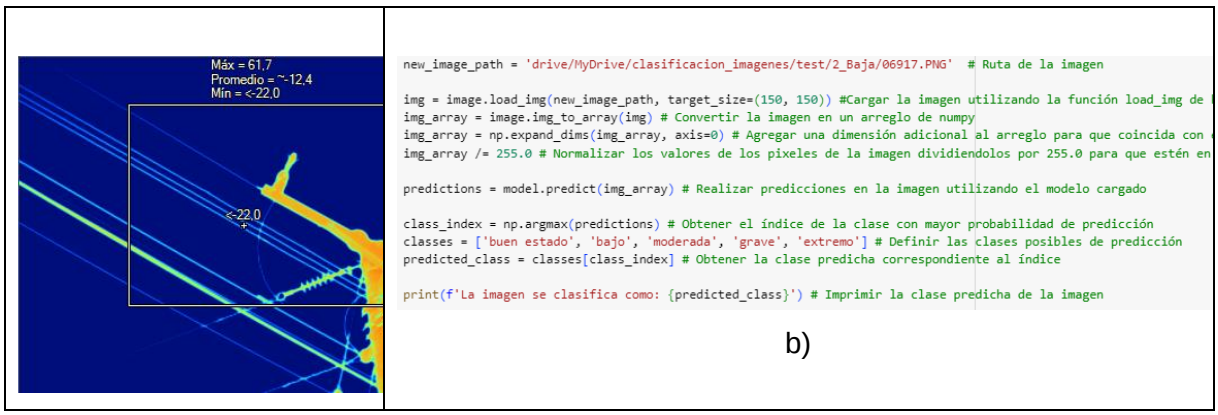


Figura 4.46 Prueba IR 06914 a) análisis termográfico, b) predicción en el modelo realizado en Python, c) Resultado de la predicción

Fuente: Elaboración propia



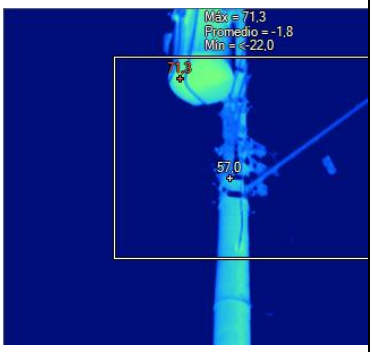
a)

```
1/1 [==============================] - 0s 39ms/step
La imagen se clasifica como: bajo
```

c)

Figura 4.47 Prueba IR 06917 a) análisis termográfico, b) predicción en el modelo realizado en Python, c) Resultado de la predicción

Fuente: Elaboración propia



a)

```python
new_image_path = 'drive/MyDrive/clasificacion_imagenes/test/2_Baja/06919.PNG'  # Ruta de la imagen

img = image.load_img(new_image_path, target_size=(150, 150)) #Cargar la imagen utilizando la función load_img de
img_array = image.img_to_array(img) # Convertir la imagen en un arreglo de numpy
img_array = np.expand_dims(img_array, axis=0) # Agregar una dimensión adicional al arreglo para que coincida con
img_array /= 255.0 # Normalizar los valores de los pixeles de la imagen dividiendolos por 255.0 para que estén en

predictions = model.predict(img_array) # Realizar predicciones en la imagen utilizando el modelo cargado

class_index = np.argmax(predictions) # Obtener el índice de la clase con mayor probabilidad de predicción
classes = ['buen estado', 'bajo', 'moderada', 'grave', 'extremo'] # Definir las clases posibles de predicción
predicted_class = classes[class_index] # Obtener la clase predicha correspondiente al índice

print(f'La imagen se clasifica como: {predicted_class}') # Imprimir la clase predicha de la imagen
```

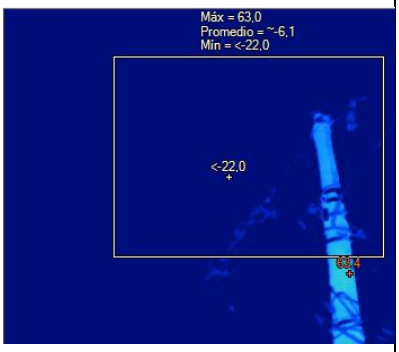
b)

```
1/1 [==============================] - 0s 35ms/step
La imagen se clasifica como: bajo
```

c)

Figura 4.48 Prueba IR 06919 a) análisis termográfico, b) predicción en el modelo realizado en Python, c) Resultado de la predicción

Fuente: Elaboración propia



a)

```python
new_image_path = 'drive/MyDrive/clasificacion_imagenes/test/2_Baja/08570.PNG'  # Ruta de la imagen

img = image.load_img(new_image_path, target_size=(150, 150)) #Cargar la imagen utilizando la función load_img de keras y r
img_array = image.img_to_array(img) # Convertir la imagen en un arreglo de numpy
img_array = np.expand_dims(img_array, axis=0) # Agregar una dimensión adicional al arreglo para que coincida con el format
img_array /= 255.0 # Normalizar los valores de los pixeles de la imagen dividiendolos por 255.0 para que estén en el rango

predictions = model.predict(img_array) # Realizar predicciones en la imagen utilizando el modelo cargado

class_index = np.argmax(predictions) # Obtener el índice de la clase con mayor probabilidad de predicción
classes = ['buen estado', 'bajo', 'moderada', 'grave', 'extremo'] # Definir las clases posibles de predicción
predicted_class = classes[class_index] # Obtener la clase predicha correspondiente al índice

print(f'La imagen se clasifica como: {predicted_class}') # Imprimir la clase predicha de la imagen
```

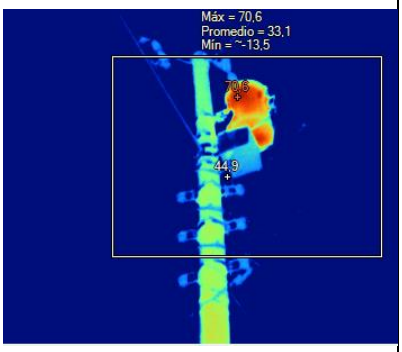
b)

```
1/1 [==============================] - 0s 34ms/step
La imagen se clasifica como: bajo
```

c)

Figura 4.49 Prueba IR 08570 a) análisis termográfico, b) predicción en el modelo realizado en Python, c) Resultado de la predicción

Fuente: Elaboración propia



a)

```
new_image_path = 'drive/MyDrive/clasificacion_imagenes/test/2_Baja/08571.PNG'  # Ruta de la imagen

img = image.load_img(new_image_path, target_size=(150, 150)) #Cargar la imagen utilizando la función load_img de keras y re
img_array = image.img_to_array(img) # Convertir la imagen en un arreglo de numpy
img_array = np.expand_dims(img_array, axis=0) # Agregar una dimensión adicional al arreglo para que coincida con el formato
img_array /= 255.0 # Normalizar los valores de los pixeles de la imagen dividiendolos por 255.0 para que estén en el rango

predictions = model.predict(img_array) # Realizar predicciones en la imagen utilizando el modelo cargado

class_index = np.argmax(predictions) # Obtener el índice de la clase con mayor probabilidad de predicción
classes = ['buen estado', 'bajo', 'moderada', 'grave', 'extremo'] # Definir las clases posibles de predicción
predicted_class = classes[class_index] # Obtener la clase predicha correspondiente al índice

print(f'La imagen se clasifica como: {predicted_class}') # Imprimir la clase predicha de la imagen
```

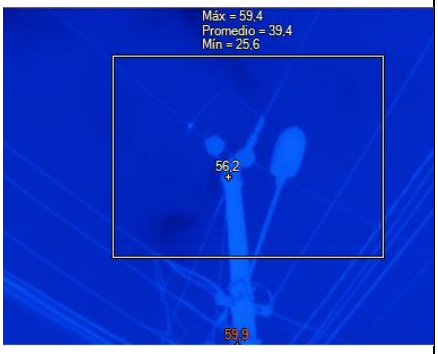
b)

```
1/1 [==============================] - 0s 34ms/step
La imagen se clasifica como: bajo
```

c)

Figura 4.50 Prueba IR 08571 a) análisis termográfico, b) predicción en el modelo realizado en Python, c) Resultado de la predicción

Fuente: Elaboración propia



a)

```
new_image_path = 'drive/MyDrive/clasificacion_imagenes/test/2_Baja/08576.PNG'  # Ruta de la imagen

img = image.load_img(new_image_path, target_size=(150, 150)) #Cargar la imagen utilizando la función load_img de keras y redimensionarla
img_array = image.img_to_array(img) # Convertir la imagen en un arreglo de numpy
img_array = np.expand_dims(img_array, axis=0) # Agregar una dimensión adicional al arreglo para que coincida con el formato de entrada d
img_array /= 255.0 # Normalizar los valores de los pixeles de la imagen dividiendolos por 255.0 para que estén en el rango de [0,1]

predictions = model.predict(img_array) # Realizar predicciones en la imagen utilizando el modelo cargado

class_index = np.argmax(predictions) # Obtener el índice de la clase con mayor probabilidad de predicción
classes = ['buen estado', 'bajo', 'moderada', 'grave', 'extremo'] # Definir las clases posibles de predicción
predicted_class = classes[class_index] # Obtener la clase predicha correspondiente al índice

print(f'La imagen se clasifica como: {predicted_class}') # Imprimir la clase predicha de la imagen
```

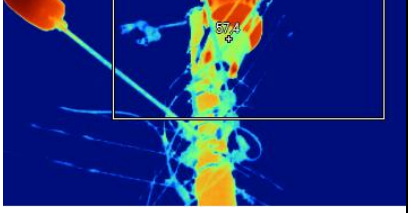
b)

```
1/1 [==============================] - 0s 130ms/step
La imagen se clasifica como: bajo
```

c)

Figura 4.51 Prueba IR 08576 a) análisis termográfico, b) predicción en el modelo realizado en Python, c) Resultado de la predicción

Fuente: Elaboración propia



a)

```
new_image_path = 'drive/MyDrive/clasificacion_imagenes/test/2_Baja/08577.PNG'  # Ruta de la imagen

img = image.load_img(new_image_path, target_size=(150, 150)) #Cargar la imagen utilizando la función load_img de keras y redim
img_array = image.img_to_array(img) # Convertir la imagen en un arreglo de numpy
img_array = np.expand_dims(img_array, axis=0) # Agregar una dimensión adicional al arreglo para que coincida con el formato de
img_array /= 255.0 # Normalizar los valores de los píxeles de la imagen dividiendolos por 255.0 para que estén en el rango de

predictions = model.predict(img_array) # Realizar predicciones en la imagen utilizando el modelo cargado

class_index = np.argmax(predictions) # Obtener el índice de la clase con mayor probabilidad de predicción
classes = ['buen estado', 'bajo', 'moderada', 'grave', 'extremo'] # Definir las clases posibles de predicción
predicted_class = classes[class_index] # Obtener la clase predicha correspondiente al índice

print(f'La imagen se clasifica como: {predicted_class}') # Imprimir la clase predicha de la imagen
```

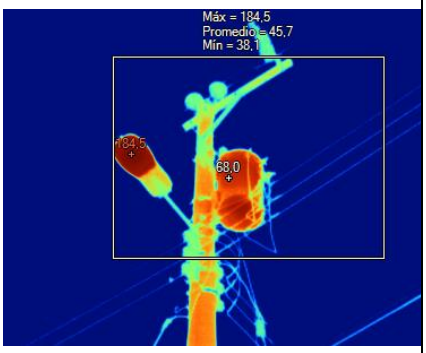
b)

```
1/1 [==============================] - 0s 130ms/step
La imagen se clasifica como: bajo
```

c)

Figura 4.52 Prueba IR 08577 a) análisis termográfico, b) predicción en el modelo realizado en Python, c) Resultado de la predicción

Fuente: Elaboración propia



a)

```
new_image_path = 'drive/MyDrive/clasificacion_imagenes/test/2_Baja/08578.PNG'  # Ruta de la imagen

img = image.load_img(new_image_path, target_size=(150, 150)) #Cargar la imagen utilizando la función load_img de keras y redimensio
img_array = image.img_to_array(img) # Convertir la imagen en un arreglo de numpy
img_array = np.expand_dims(img_array, axis=0) # Agregar una dimensión adicional al arreglo para que coincida con el formato de entr
img_array /= 255.0 # Normalizar los valores de los píxeles de la imagen dividiendolos por 255.0 para que estén en el rango de [0,1

predictions = model.predict(img_array) # Realizar predicciones en la imagen utilizando el modelo cargado

class_index = np.argmax(predictions) # Obtener el índice de la clase con mayor probabilidad de predicción
classes = ['buen estado', 'bajo', 'moderada', 'grave', 'extremo'] # Definir las clases posibles de predicción
predicted_class = classes[class_index] # Obtener la clase predicha correspondiente al índice

print(f'La imagen se clasifica como: {predicted_class}') # Imprimir la clase predicha de la imagen
```

b)

```
1/1 [==============================] - 0s 60ms/step
La imagen se clasifica como: bajo
```

c)

Figura 4.53 Prueba IR 08578 a) análisis termográfico, b) predicción en el modelo realizado en Python, c) Resultado de la predicción

Fuente: Elaboración propia

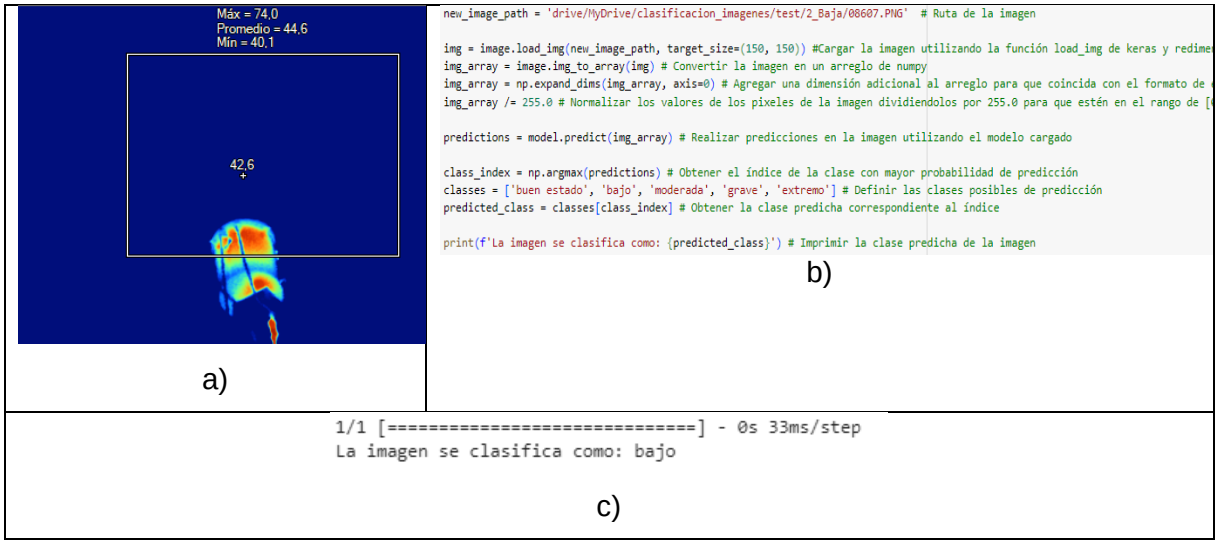


Figura 4.54 Prueba IR 08607 a) análisis termográfico, b) predicción en el modelo realizado en Python, c) Resultado de la predicción

Fuente: Elaboración propia

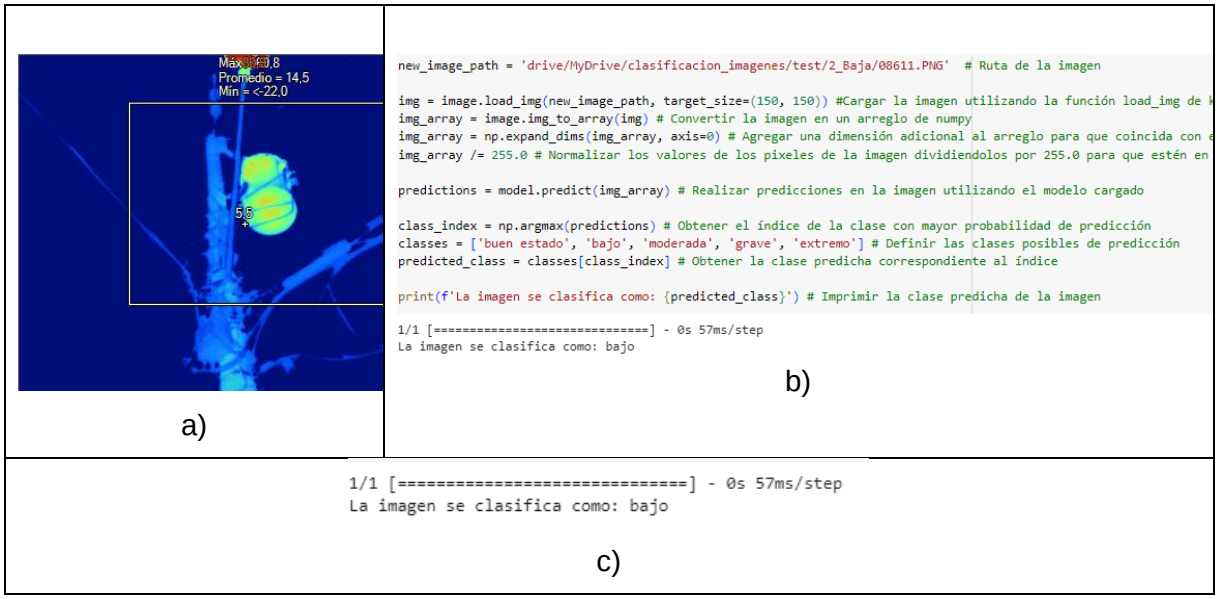


Figura 4.55 Prueba IR 08611 a) análisis termográfico, b) predicción en el modelo realizado en Python, c) Resultado de la predicción

Fuente: Elaboración propia

Por medio de la Tabla 9 se observa que el modelo presenta buena precisión, puesto que presenta un criterio de severidad nula.

Tabla 9. Pruebas de clasificación en severidad baja.

Fotografía IR	Temperatura |°C|	Informe termográfico	Red neuronal Convolucional (CNN)
06914	22	Baja	Baja

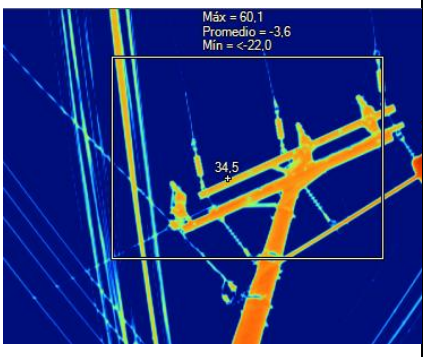
06917	21	Baja	Baja
06919	22	Baja	Baja
08570	23	Baja	Baja
08571	21	Baja	Baja
08576	23.4	Baja	Baja
08577	22.1	Baja	Baja
08578	22	Baja	Baja
08607	20	Baja	Baja
08611	18.6	Baja	Baja

Fuente: Elaboración propia

Los resultados favorables de nuestras predicciones de severidad baja de degradación térmica son alentadores. Gracias al modelo desarrollado, se ha logrado predecir con éxito la mínima degradación causada por la temperatura en diferentes partes del transformador de distribución. Estos hallazgos respaldan la eficacia y precisión de nuestro modelo, lo que nos permite tomar decisiones informadas para garantizar integridad y durabilidad de los materiales en condiciones térmicas desafiantes.

4.3.3 CIRCUITOS EN SEVERIDAD MODERADA:

Se debe realizar pruebas de transformadores de distribución, por medio del cual se ingresa la información a través del modelo para que brinde buena eficiencia em la clasificación sobre niveles de severidad.



a)

```
new_image_path = 'drive/MyDrive/clasificacion_imagenes/test/3_Moderada/09974.PNG'  # Ruta de la imagen

img = image.load_img(new_image_path, target_size=(150, 150)) #Cargar la imagen utilizando la función load_img de keras y redimensio
img_array = image.img_to_array(img) # Convertir la imagen en un arreglo de numpy
img_array = np.expand_dims(img_array, axis=0) # Agregar una dimensión adicional al arreglo para que coincida con el formato de entr
img_array /= 255.0 # Normalizar los valores de los pixeles de la imagen dividiendolos por 255.0 para que estén en el rango de [0,1]

predictions = model.predict(img_array) # Realizar predicciones en la imagen utilizando el modelo cargado

class_index = np.argmax(predictions) # Obtener el índice de la clase con mayor probabilidad de predicción
classes = ['buen estado', 'bajo', 'moderada', 'grave', 'extremo'] # Definir las clases posibles de predicción
predicted_class = classes[class_index] # Obtener la clase predicha correspondiente al índice

print(f'La imagen se clasifica como: {predicted_class}') # Imprimir la clase predicha de la imagen
```

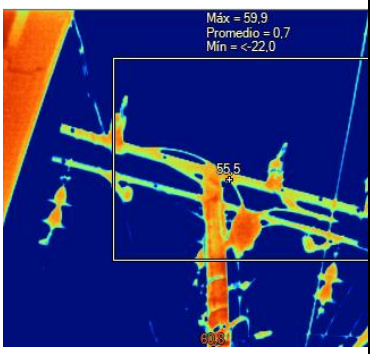
b)

```
1/1 [==============================] - 0s 34ms/step
La imagen se clasifica como: moderada
```

c)

Figura 4.56 Prueba IR 09974 a) análisis termográfico, b) predicción en el modelo realizado en Python, c) Resultado de la predicción

Fuente: Elaboración propia



a)

```
new_image_path = 'drive/MyDrive/clasificacion_imagenes/test/3_Moderada/09975.PNG'  # Ruta de la imagen

img = image.load_img(new_image_path, target_size=(150, 150)) #Cargar la imagen utilizando la función load_img de k
img_array = image.img_to_array(img) # Convertir la imagen en un arreglo de numpy
img_array = np.expand_dims(img_array, axis=0) # Agregar una dimensión adicional al arreglo para que coincida con e
img_array /= 255.0 # Normalizar los valores de los pixeles de la imagen dividiendolos por 255.0 para que estén en

predictions = model.predict(img_array) # Realizar predicciones en la imagen utilizando el modelo cargado

class_index = np.argmax(predictions) # Obtener el índice de la clase con mayor probabilidad de predicción
classes = ['buen estado', 'bajo', 'moderada', 'grave', 'extremo'] # Definir las clases posibles de predicción
predicted_class = classes[class_index] # Obtener la clase predicha correspondiente al índice

print(f'La imagen se clasifica como: {predicted_class}') # Imprimir la clase predicha de la imagen
```

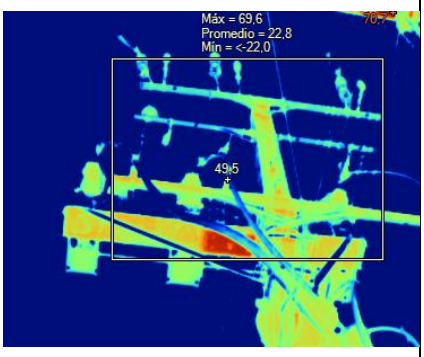
b)

```
1/1 [==============================] - 0s 33ms/step
La imagen se clasifica como: moderada
```

c)

Figura 4.57 Prueba IR 09975 a) análisis termográfico, b) predicción en el modelo realizado en Python, c) Resultado de la predicción

Fuente: Elaboración propia



a)

```
new_image_path = 'drive/MyDrive/clasificacion_imagenes/test/3_Moderada/09972.PNG'  # Ruta de la imagen

img = image.load_img(new_image_path, target_size=(150, 150)) #Cargar la imagen utilizando la función load_img de keras y redimensi
img_array = image.img_to_array(img) # Convertir la imagen en un arreglo de numpy
img_array = np.expand_dims(img_array, axis=0) # Agregar una dimensión adicional al arreglo para que coincida con el formato de ent
img_array /= 255.0 # Normalizar los valores de los pixeles de la imagen dividiendolos por 255.0 para que estén en el rango de [0,1

predictions = model.predict(img_array) # Realizar predicciones en la imagen utilizando el modelo cargado

class_index = np.argmax(predictions) # Obtener el índice de la clase con mayor probabilidad de predicción
classes = ['buen estado', 'bajo', 'moderada', 'grave', 'extremo'] # Definir las clases posibles de predicción
predicted_class = classes[class_index] # Obtener la clase predicha correspondiente al índice

print(f'La imagen se clasifica como: {predicted_class}') # Imprimir la clase predicha de la imagen
```

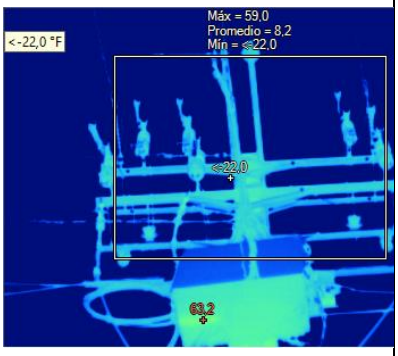
b)

```
1/1 [==============================] - 0s 34ms/step
La imagen se clasifica como: moderada
```

c)

Figura 4.58 Prueba IR 09972 a) análisis termográfico, b) predicción en el modelo realizado en Python, c) Resultado de la predicción

Fuente: Elaboración propia



a)

```
new_image_path = 'drive/MyDrive/clasificacion_imagenes/test/3_Moderada/09963.PNG'  # Ruta de la imagen

img = image.load_img(new_image_path, target_size=(150, 150)) #Cargar la imagen utilizando la función load_img de keras y r
img_array = image.img_to_array(img) # Convertir la imagen en un arreglo de numpy
img_array = np.expand_dims(img_array, axis=0) # Agregar una dimensión adicional al arreglo para que coincida con el format
img_array /= 255.0 # Normalizar los valores de los pixeles de la imagen dividiendolos por 255.0 para que estén en el rango

predictions = model.predict(img_array) # Realizar predicciones en la imagen utilizando el modelo cargado

class_index = np.argmax(predictions) # Obtener el índice de la clase con mayor probabilidad de predicción
classes = ['buen estado', 'bajo', 'moderada', 'grave', 'extremo'] # Definir las clases posibles de predicción
predicted_class = classes[class_index] # Obtener la clase predicha correspondiente al índice

print(f'La imagen se clasifica como: {predicted_class}') # Imprimir la clase predicha de la imagen
```

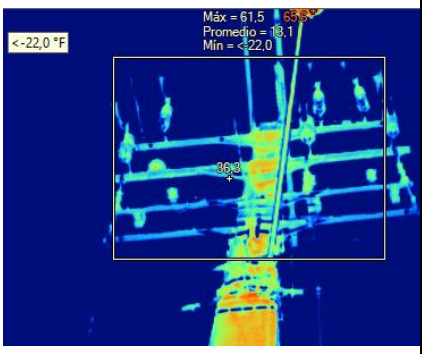
b)

```
1/1 [==============================] - 0s 97ms/step
La imagen se clasifica como: moderada
```

c)

Figura 4.58 Prueba IR 09963 a) análisis termográfico, b) predicción en el modelo realizado en Python, c) Resultado de la predicción

Fuente: Elaboración propia



a)

```
new_image_path = 'drive/MyDrive/clasificacion_imagenes/test/3_Moderada/09964.PNG'  # Ruta de la imagen

img = image.load_img(new_image_path, target_size=(150, 150)) #Cargar la imagen utilizando la función load_img de keras y redimensi
img_array = image.img_to_array(img) # Convertir la imagen en un arreglo de numpy
img_array = np.expand_dims(img_array, axis=0) # Agregar una dimensión adicional al arreglo para que coincida con el formato de ent
img_array /= 255.0 # Normalizar los valores de los pixeles de la imagen dividiendolos por 255.0 para que estén en el rango de [0,1

predictions = model.predict(img_array) # Realizar predicciones en la imagen utilizando el modelo cargado

class_index = np.argmax(predictions) # Obtener el índice de la clase con mayor probabilidad de predicción
classes = ['buen estado', 'bajo', 'moderada', 'grave', 'extremo'] # Definir las clases posibles de predicción
predicted_class = classes[class_index] # Obtener la clase predicha correspondiente al índice

print(f'La imagen se clasifica como: {predicted_class}') # Imprimir la clase predicha de la imagen
```

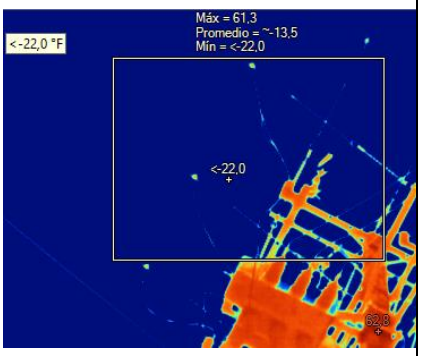
b)

```
1/1 [==============================] - 0s 60ms/step
La imagen se clasifica como: moderada
```

c)

Figura 4.59 Prueba IR 09964 a) análisis termográfico, b) predicción en el modelo realizado en Python, c) Resultado de la predicción

Fuente: Elaboración propia



a)

```
new_image_path = 'drive/MyDrive/clasificacion_imagenes/test/3_Moderada/09966.PNG'  # Ruta de la imagen

img = image.load_img(new_image_path, target_size=(150, 150)) #Cargar la imagen utilizando la función load_img de keras y redimens
img_array = image.img_to_array(img) # Convertir la imagen en un arreglo de numpy
img_array = np.expand_dims(img_array, axis=0) # Agregar una dimensión adicional al arreglo para que coincida con el formato de en
img_array /= 255.0 # Normalizar los valores de los pixeles de la imagen dividiendolos por 255.0 para que estén en el rango de [0,

predictions = model.predict(img_array) # Realizar predicciones en la imagen utilizando el modelo cargado

class_index = np.argmax(predictions) # Obtener el índice de la clase con mayor probabilidad de predicción
classes = ['buen estado', 'bajo', 'moderada', 'grave', 'extremo'] # Definir las clases posibles de predicción
predicted_class = classes[class_index] # Obtener la clase predicha correspondiente al índice

print(f'La imagen se clasifica como: {predicted_class}') # Imprimir la clase predicha de la imagen
```

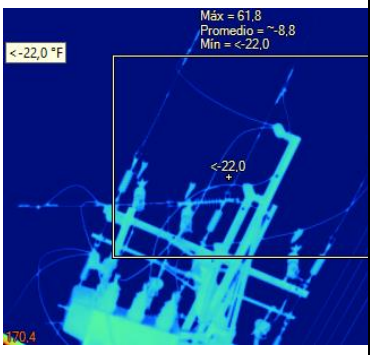
b)

```
1/1 [==============================] - 0s 38ms/step
La imagen se clasifica como: moderada
```

c)

Figura 4.60 Prueba IR 09966 a) análisis termográfico, b) predicción en el modelo realizado en Python, c) Resultado de la predicción

Fuente: Elaboración propia



a)

```
new_image_path = 'drive/MyDrive/clasificacion_imagenes/test/3_Moderada/09968.PNG'  # Ruta de la imagen

img = image.load_img(new_image_path, target_size=(150, 150)) #Cargar la imagen utilizando la función load_img de k
img_array = image.img_to_array(img) # Convertir la imagen en un arreglo de numpy
img_array = np.expand_dims(img_array, axis=0) # Agregar una dimensión adicional al arreglo para que coincida con e
img_array /= 255.0 # Normalizar los valores de los pixeles de la imagen dividiendolos por 255.0 para que estén en

predictions = model.predict(img_array) # Realizar predicciones en la imagen utilizando el modelo cargado

class_index = np.argmax(predictions) # Obtener el índice de la clase con mayor probabilidad de predicción
classes = ['buen estado', 'bajo', 'moderada', 'grave', 'extremo'] # Definir las clases posibles de predicción
predicted_class = classes[class_index] # Obtener la clase predicha correspondiente al índice

print(f'La imagen se clasifica como: {predicted_class}') # Imprimir la clase predicha de la imagen
```

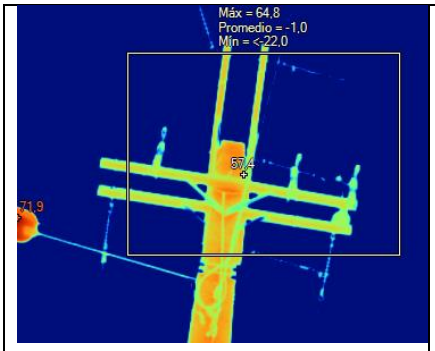
b)

```
1/1 [==============================] - 0s 34ms/step
La imagen se clasifica como: moderada
```

c)

Figura 4.61 Prueba IR 09968 a) análisis termográfico, b) predicción en el modelo realizado en Python, c) Resultado de la predicción

Fuente: Elaboración propia



a)

```
new_image_path = 'drive/MyDrive/clasificacion_imagenes/test/3_Moderada/09970.PNG'  # Ruta de la imagen

img = image.load_img(new_image_path, target_size=(150, 150)) #Cargar la imagen utilizando la función load_img de keras y redimens
img_array = image.img_to_array(img) # Convertir la imagen en un arreglo de numpy
img_array = np.expand_dims(img_array, axis=0) # Agregar una dimensión adicional al arreglo para que coincida con el formato de en
img_array /= 255.0 # Normalizar los valores de los pixeles de la imagen dividiendolos por 255.0 para que estén en el rango de [0,

predictions = model.predict(img_array) # Realizar predicciones en la imagen utilizando el modelo cargado

class_index = np.argmax(predictions) # Obtener el índice de la clase con mayor probabilidad de predicción
classes = ['buen estado', 'bajo', 'moderada', 'grave', 'extremo'] # Definir las clases posibles de predicción
predicted_class = classes[class_index] # Obtener la clase predicha correspondiente al índice

print(f'La imagen se clasifica como: {predicted_class}') # Imprimir la clase predicha de la imagen
```

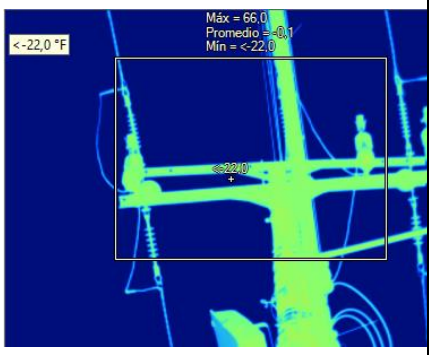
b)

```
1/1 [==============================] - 0s 53ms/step
La imagen se clasifica como: moderada
```

c)

Figura 4.62 Prueba IR 09970 a) análisis termográfico, b) predicción en el modelo realizado en Python, c) Resultado de la predicción

Fuente: Elaboración propia



a)

```
new_image_path = 'drive/MyDrive/clasificacion_imagenes/test/3_Moderada/09995.PNG'  # Ruta de la imagen

img = image.load_img(new_image_path, target_size=(150, 150)) #Cargar la imagen utilizando la función load_img de keras y redimensionar
img_array = image.img_to_array(img) # Convertir la imagen en un arreglo de numpy
img_array = np.expand_dims(img_array, axis=0) # Agregar una dimensión adicional al arreglo para que coincida con el formato de entrada
img_array /= 255.0 # Normalizar los valores de los pixeles de la imagen dividiendolos por 255.0 para que estén en el rango de [0,1]

predictions = model.predict(img_array) # Realizar predicciones en la imagen utilizando el modelo cargado

class_index = np.argmax(predictions) # Obtener el índice de la clase con mayor probabilidad de predicción
classes = ['buen estado', 'bajo', 'moderada', 'grave', 'extremo'] # Definir las clases posibles de predicción
predicted_class = classes[class_index] # Obtener la clase predicha correspondiente al índice

print(f'La imagen se clasifica como: {predicted_class}') # Imprimir la clase predicha de la imagen
```

b)

```
1/1 [==============================] - 0s 53ms/step
La imagen se clasifica como: moderada
```

c)

Figura 4.63 Prueba IR 09951 a) análisis termográfico, b) predicción en el modelo realizado en Python, c) Resultado de la predicción

Fuente: Elaboración propia

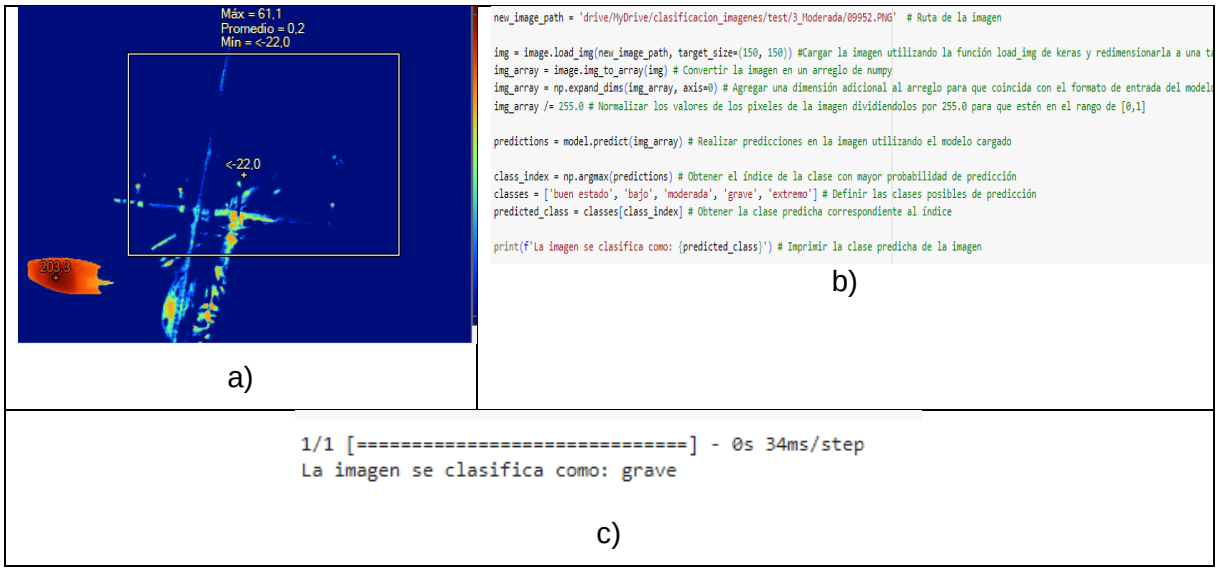


Figura 4.64 Prueba IR 09952 a) análisis termográfico, b) predicción en el modelo realizado en Python, c) Resultado de la predicción

Fuente: Elaboración propia

Por medio de la Tabla 10 se observa que el modelo presenta una buena precisión puesto que presenta un criterio de severidad grave en una imagen térmica analizada en severidad moderada.

Tabla 10. Comparación de clasificación mediante un informe termográfico y una red neuronal convolucional

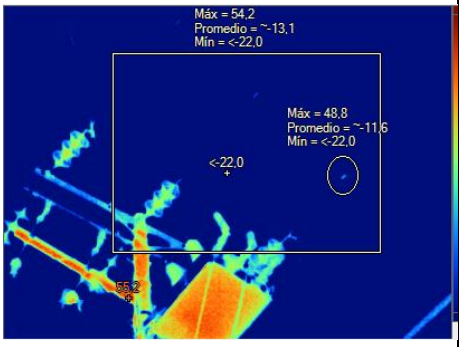
Fotografía IR	Temperatura |°C|	Informe termográfico	Red neuronal Convolucional (CNN)
09974	25	Moderada	Moderada
09975	26	Moderada	Moderada
09972	26	Moderada	Moderada
09963	27	Moderada	Moderada
09964	28	Moderada	Moderada
09966	25.5	Moderada	Moderada
09968	27.6	Moderada	Moderada
09970	30	Moderada	Moderada
09951	30	Moderada	Moderada
09952	32.1	Moderada	grave

Fuente: Elaboración propia

Los resultados de nuestras predicciones de severidad moderada en degradación térmica no son los esperados. A pesar de los esfuerzos realizados utilizando nuestro modelo, hemos identificado una cierta cantidad de deterioro causada por la temperatura en diferentes materiales. Estos hallazgos nos hacen que instantáneamente sean revisadas y mejore nuestro enfoque para lograr una mayor precisión en futuras predicciones.

4.3.4 CIRCUITOS EN SEVERIDAD GRAVE:

Se realiza pruebas en transformadores de distribución, por medio del cual se ingresa la información a través del modelo para que brinde buena eficiencia em la clasificación sobre niveles de severidad.



a)

```
new_image_path = 'drive/MyDrive/clasificacion_imagenes/test/4_Grave/00007.PNG'  # Ruta de la imagen

img = image.load_img(new_image_path, target_size=(150, 150)) #Cargar la imagen utilizando la función load_img de keras y redimensionarla a una
img_array = image.img_to_array(img) # Convertir la imagen en un arreglo de numpy
img_array = np.expand_dims(img_array, axis=0) # Agregar una dimensión adicional al arreglo para que coincida con el formato de entrada del mode
img_array /= 255.0 # Normalizar los valores de los pixeles de la imagen dividiendolos por 255.0 para que estén en el rango de [0,1]

predictions = model.predict(img_array) # Realizar predicciones en la imagen utilizando el modelo cargado

class_index = np.argmax(predictions) # Obtener el índice de la clase con mayor probabilidad de predicción
classes = ['buen estado', 'bajo', 'moderada', 'grave', 'extremo'] # Definir las clases posibles de predicción
predicted_class = classes[class_index] # Obtener la clase predicha correspondiente al índice

print(f'La imagen se clasifica como: {predicted_class}') # Imprimir la clase predicha de la imagen
```

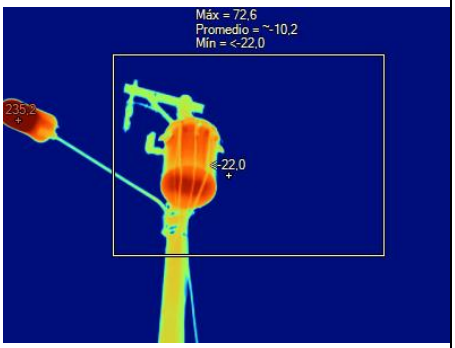
b)

```
1/1 [==============================] - 0s 42ms/step
La imagen se clasifica como: grave
```

c)

Figura 4.65 Prueba IR 00007 a) análisis termográfico, b) predicción en el modelo realizado en Python, c) Resultado de la predicción

Fuente: Elaboración propia



a)

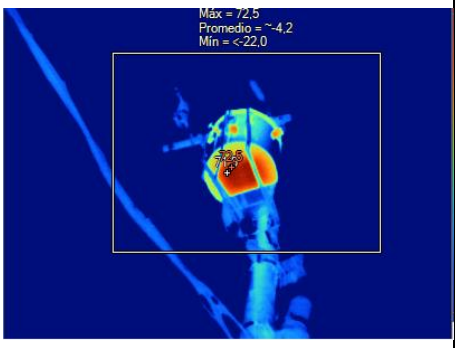
```
new_image_path = 'drive/MyDrive/clasificacion_imagenes/test/4_Grave/00168.PNG'  # Ruta de la imagen

img = image.load_img(new_image_path, target_size=(150, 150)) #Cargar la imagen utilizando la función load_img de keras y redimensionarla a u
img_array = image.img_to_array(img) # Convertir la imagen en un arreglo de numpy
img_array = np.expand_dims(img_array, axis=0) # Agregar una dimensión adicional al arreglo para que coincida con el formato de entrada del m
img_array /= 255.0 # Normalizar los valores de los pixeles de la imagen dividiendolos por 255.0 para que estén en el rango de [0,1]

predictions = model.predict(img_array) # Realizar predicciones en la imagen utilizando el modelo cargado

class_index = np.argmax(predictions) # Obtener el índice de la clase con mayor probabilidad de predicción
classes = ['buen estado', 'bajo', 'moderada', 'grave', 'extremo'] # Definir las clases posibles de predicción
predicted_class = classes[class_index] # Obtener la clase predicha correspondiente al índice

print(f'La imagen se clasifica como: {predicted_class}') # Imprimir la clase predicha de la imagen
```

b)

```
1/1 [==============================] - 0s 38ms/step
La imagen se clasifica como: moderada
```

c)

Figura 4.66 Prueba IR 00168 a) análisis termográfico, b) predicción en el modelo realizado en Python, c) Resultado de la predicción



a)

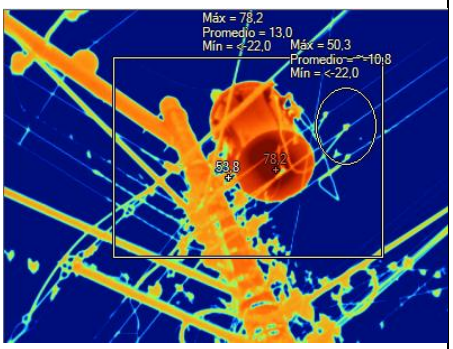
```
new_image_path = 'drive/MyDrive/clasificacion_imagenes/test/4_Grave/00180.PNG'  # Ruta de la imagen

img = image.load_img(new_image_path, target_size=(150, 150)) #Cargar la imagen utilizando la función load_img de keras y redimensionarla a un
img_array = image.img_to_array(img) # Convertir la imagen en un arreglo de numpy
img_array = np.expand_dims(img_array, axis=0) # Agregar una dimensión adicional al arreglo para que coincida con el formato de entrada del mo
img_array /= 255.0 # Normalizar los valores de los pixeles de la imagen dividiendolos por 255.0 para que estén en el rango de [0,1]

predictions = model.predict(img_array) # Realizar predicciones en la imagen utilizando el modelo cargado

class_index = np.argmax(predictions) # Obtener el índice de la clase con mayor probabilidad de predicción
classes = ['buen estado', 'bajo', 'moderada', 'grave', 'extremo'] # Definir las clases posibles de predicción
predicted_class = classes[class_index] # Obtener la clase predicha correspondiente al índice

print(f'La imagen se clasifica como: {predicted_class}') # Imprimir la clase predicha de la imagen
```

b)

```
1/1 [==============================] - 0s 33ms/step
La imagen se clasifica como: grave
```

c)

Figura 4.67 Prueba IR 00180 a) análisis termográfico, b) predicción en el modelo realizado en Python, c) Resultado de la predicción



a)

```
new_image_path = 'drive/MyDrive/clasificacion_imagenes/test/4_Grave/07471.PNG'  # Ruta de la imagen

img = image.load_img(new_image_path, target_size=(150, 150)) #Cargar la imagen utilizando la función load_img de keras y redimensionarla a
img_array = image.img_to_array(img) # Convertir la imagen en un arreglo de numpy
img_array = np.expand_dims(img_array, axis=0) # Agregar una dimensión adicional al arreglo para que coincida con el formato de entrada del
img_array /= 255.0 # Normalizar los valores de los pixeles de la imagen dividiendolos por 255.0 para que estén en el rango de [0,1]

predictions = model.predict(img_array) # Realizar predicciones en la imagen utilizando el modelo cargado

class_index = np.argmax(predictions) # Obtener el índice de la clase con mayor probabilidad de predicción
classes = ['buen estado', 'bajo', 'moderada', 'grave', 'extremo'] # Definir las clases posibles de predicción
predicted_class = classes[class_index] # Obtener la clase predicha correspondiente al índice

print(f'La imagen se clasifica como: {predicted_class}') # Imprimir la clase predicha de la imagen
```

b)

```
1/1 [==============================] - 0s 35ms/step
La imagen se clasifica como: grave
```

c)

Figura 4.68 Prueba IR 07471 a) análisis termográfico, b) predicción en el modelo realizado en Python, c) Resultado de la predicción

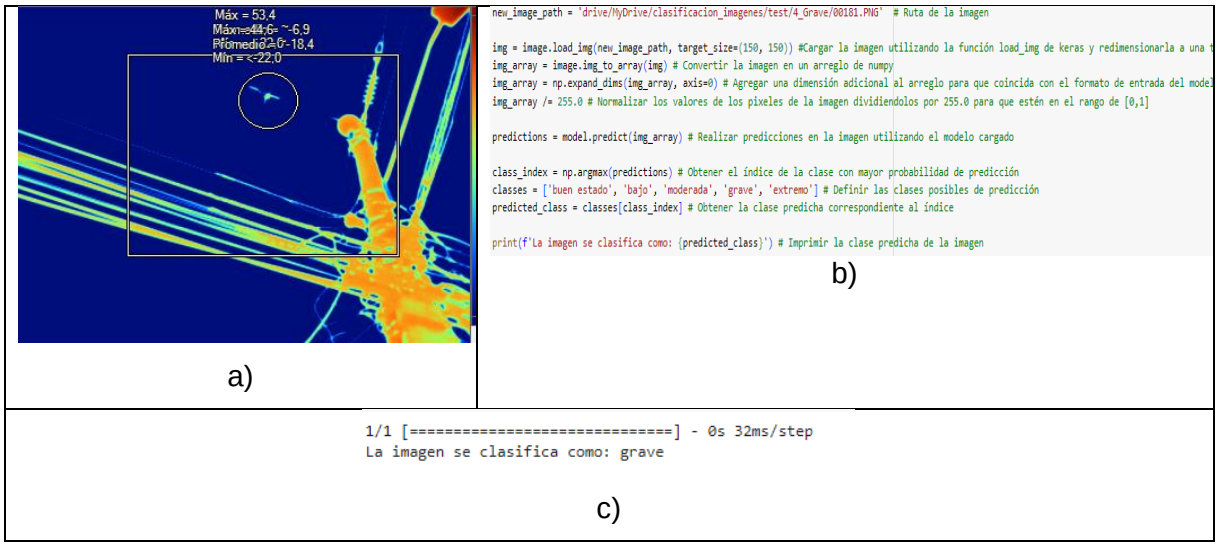


Figura 4.69 Prueba IR 00007 a) análisis termográfico, b) predicción en el modelo realizado en Python, c) Resultado de la predicción

Por medio de la Tabla 11 se observa que el modelo no presenta una buena precisión, puesto que presenta un criterio de severidad moderada en imágenes térmicas analizada en severidad moderada.

Tabla 11. Comparación de clasificación mediante un informe termográfico y una red neuronal convolucional

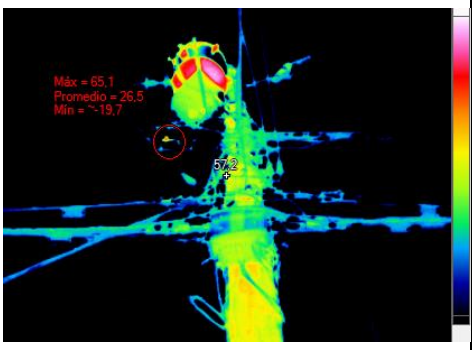
Fotografía IR	Temperatura |°C|	Informe termográfico	Red neuronal Convolucional (CNN)
00007	27.3	Grave	Moderada
00168	44	Grave	Grave
00180	42	Grave	Grave
07471	38	Grave	Grave
00181	45.5	Grave	Grave

Fuente: Elaboración propia

Los resultados de nuestras predicciones de severidad grave de degradacion termica pueden ser confusos al mostrar una tendencia hacia la severidad moderada. Aunque nuestro modelo desarrollado ha brindado cierta informacion sobre la degradacion térmica, es importante tener en cuenta la posibilidad de una subestimación de la severidad real.

4.3.5 CIRCUITOS EN SEVERIDAD EXTREMA:

Se realiza pruebas de clasificación de severidad extrema de degradación térmica en circuitos de transformadores de distribución, por medio del cual se ingresa la información a través del modelo para que brinde buena eficiencia en la clasificación sobre niveles de severidad. Desde la Figura 4.70 a la Figura 4.74 se puede observar las pruebas realizadas.



a)

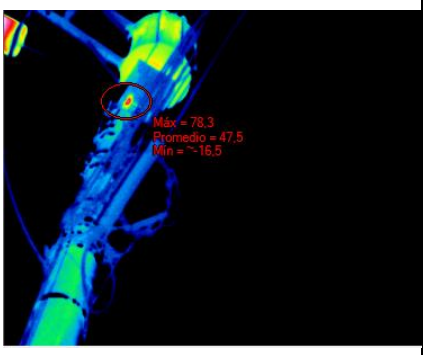
```
new_image_path = 'drive/MyDrive/clasificacion_imagenes/test/5_Extrema/00147 Extrema.PNG'  # Ruta de la imagen

img = image.load_img(new_image_path, target_size=(150, 150)) #Cargar la imagen utilizando la función load_img de keras y redimensionarla a una tama
img_array = image.img_to_array(img) # Convertir la imagen en un arreglo de numpy
img_array = np.expand_dims(img_array, axis=0) # Agregar una dimensión adicional al arreglo para que coincida con el formato de entrada del modelo
img_array /= 255.0 # Normalizar los valores de los pixeles de la imagen dividiendolos por 255.0 para que estén en el rango de [0,1]

predictions = model.predict(img_array) # Realizar predicciones en la imagen utilizando el modelo cargado

class_index = np.argmax(predictions) # Obtener el índice de la clase con mayor probabilidad de predicción
classes = ['buen estado', 'bajo', 'moderada', 'grave', 'extremo'] # Definir las clases posibles de predicción
predicted_class = classes[class_index] # Obtener la clase predicha correspondiente al índice

print(f'La imagen se clasifica como: {predicted_class}') # Imprimir la clase predicha de la imagen
```

b)

```
1/1 [==============================] - 0s 34ms/step
La imagen se clasifica como: extremo
```

c)

Figura 4.70 Prueba IR 00147 a) análisis termográfico, b) predicción en el modelo realizado en Python, c) Resultado de la predicción



a)

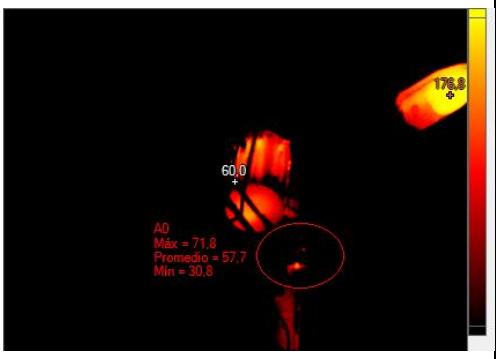
```
new_image_path = 'drive/MyDrive/clasificacion_imagenes/test/5_Extrema/00162 Extrema.PNG'  # Ruta de la imagen

img = image.load_img(new_image_path, target_size=(150, 150)) #Cargar la imagen utilizando la función load_img de keras y redimensio
img_array = image.img_to_array(img) # Convertir la imagen en un arreglo de numpy
img_array = np.expand_dims(img_array, axis=0) # Agregar una dimensión adicional al arreglo para que coincida con el formato de entr
img_array /= 255.0 # Normalizar los valores de los pixeles de la imagen dividiendolos por 255.0 para que estén en el rango de [0,1

predictions = model.predict(img_array) # Realizar predicciones en la imagen utilizando el modelo cargado

class_index = np.argmax(predictions) # Obtener el índice de la clase con mayor probabilidad de predicción
classes = ['buen estado', 'bajo', 'moderada', 'grave', 'extremo'] # Definir las clases posibles de predicción
predicted_class = classes[class_index] # Obtener la clase predicha correspondiente al índice

print(f'La imagen se clasifica como: {predicted_class}') # Imprimir la clase predicha de la imagen
```

b)

```
1/1 [==============================] - 0s 34ms/step
La imagen se clasifica como: extremo
```

c)

Figura 4.71 Prueba IR 00162 a) análisis termográfico, b) predicción en el modelo realizado en Python, c) Resultado de la predicción



a)

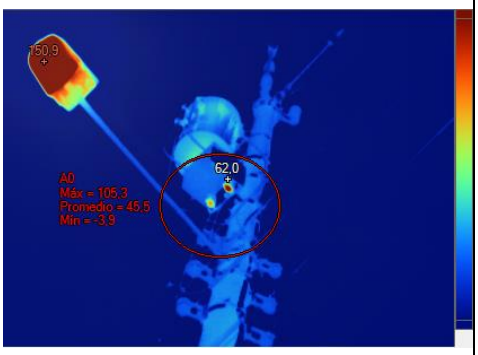
```
new_image_path = 'drive/MyDrive/clasificacion_imagenes/test/5_Extrema/06944 Extrema.PNG'  # Ruta de la imagen

img = image.load_img(new_image_path, target_size=(150, 150)) #Cargar la imagen utilizando la función load_img de keras y redimensionarla a una tamaño de 15
img_array = image.img_to_array(img) # Convertir la imagen en un arreglo de numpy
img_array = np.expand_dims(img_array, axis=0) # Agregar una dimensión adicional al arreglo para que coincida con el formato de entrada del modelo
img_array /= 255.0 # Normalizar los valores de los pixeles de la imagen dividiendolos por 255.0 para que estén en el rango de [0,1]

predictions = model.predict(img_array) # Realizar predicciones en la imagen utilizando el modelo cargado

class_index = np.argmax(predictions) # Obtener el índice de la clase con mayor probabilidad de predicción
classes = ['buen estado', 'bajo', 'moderada', 'grave', 'extremo'] # Definir las clases posibles de predicción
predicted_class = classes[class_index] # Obtener la clase predicha correspondiente al índice

print(f'La imagen se clasifica como: {predicted_class}') # Imprimir la clase predicha de la imagen
```

b)

```
1/1 [==============================] - 0s 32ms/step
La imagen se clasifica como: extremo
```

c)

Figura 4.72 Prueba IR 06944 a) análisis termográfico, b) predicción en el modelo realizado en Python, c) Resultado de la predicción



a)

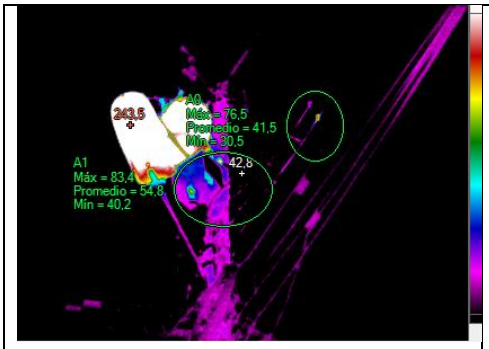
```
new_image_path = 'drive/MyDrive/clasificacion_imagenes/test/5_Extrema/06949 Extrema.PNG'  # Ruta de la imagen

img = image.load_img(new_image_path, target_size=(150, 150)) #Cargar la imagen utilizando la función load_img de keras y redimensionarla a una tamañ
img_array = image.img_to_array(img) # Convertir la imagen en un arreglo de numpy
img_array = np.expand_dims(img_array, axis=0) # Agregar una dimensión adicional al arreglo para que coincida con el formato de entrada del modelo
img_array /= 255.0 # Normalizar los valores de los pixeles de la imagen dividiendolos por 255.0 para que estén en el rango de [0,1]

predictions = model.predict(img_array) # Realizar predicciones en la imagen utilizando el modelo cargado

class_index = np.argmax(predictions) # Obtener el índice de la clase con mayor probabilidad de predicción
classes = ['buen estado', 'bajo', 'moderada', 'grave', 'extremo'] # Definir las clases posibles de predicción
predicted_class = classes[class_index] # Obtener la clase predicha correspondiente al índice

print(f'La imagen se clasifica como: {predicted_class}') # Imprimir la clase predicha de la imagen
```

b)

```
1/1 [==============================] - 0s 34ms/step
La imagen se clasifica como: extremo
```

c)

Figura 4.73 Prueba IR 06949 a) análisis termográfico, b) predicción en el modelo realizado en Python, c) Resultado de la predicción



a)

```
new_image_path = 'drive/MyDrive/clasificacion_imagenes/test/5_Extrema/07548 Extrema.PNG'  # Ruta de la imagen

img = image.load_img(new_image_path, target_size=(150, 150)) #Cargar la imagen utilizando la función load_img de keras y redimensionarla a una tama
img_array = image.img_to_array(img) # Convertir la imagen en un arreglo de numpy
img_array = np.expand_dims(img_array, axis=0) # Agregar una dimensión adicional al arreglo para que coincida con el formato de entrada del modelo
img_array /= 255.0 # Normalizar los valores de los pixeles de la imagen dividiendolos por 255.0 para que estén en el rango de [0,1]

predictions = model.predict(img_array) # Realizar predicciones en la imagen utilizando el modelo cargado

class_index = np.argmax(predictions) # Obtener el índice de la clase con mayor probabilidad de predicción
classes = ['buen estado', 'bajo', 'moderada', 'grave', 'extremo'] # Definir las clases posibles de predicción
predicted_class = classes[class_index] # Obtener la clase predicha correspondiente al índice

print(f'La imagen se clasifica como: {predicted_class}') # Imprimir la clase predicha de la imagen
```

b)

```
1/1 [==============================] - 0s 51ms/step
La imagen se clasifica como: extremo
```

c)

Figura 4.74 Prueba IR 07548 a) análisis termográfico, b) predicción en el modelo realizado en Python, c) Resultado de la predicción

Por medio de la Tabla 12. se observa que el modelo presenta una buena precisión, puesto que presenta un criterio de severidad extrema en imágenes térmicas.

Tabla 12. Comparación de clasificación de severidad extrema mediante un informe termográfico y una red neuronal convolucional

Fotografía IR	Temperatura |°C|	Informe termográfico	Red neuronal Convolucional (CNN)
00147	63.1	Extrema	Extrema
00162	78	Extrema	Extrema
06944	77	Extrema	Extrema
06949	62.5	Extrema	Extrema
07548	75	Extrema	Extrema

Fuente: Elaboración propia

En los resultados de nuestras predicciones de severidad extrema de degradación térmica son alarmantes. Utilizando nuestro modelo, se ha logrado identificar con precisión las condiciones en las que los materiales experimentarán una degradación térmica severa. Estos hallazgos nos brindan una visión crítica sobre los límites de temperatura que los materiales pueden soportar, lo que nos permite tomar medidas inmediatas para evitar daños graves.

CAPÍTULO V
CONCLUSIONES

En primer lugar, se han logrado importantes avances en el desarrollo de un modelo de clasificación de imágenes térmicas según su clase. Se ha demostrado que el modelo es capaz de realizar predicciones con una precisión aceptable en la categorización de las imágenes en las diferentes clases establecidas. Esto indica que el modelo tiene potencial, aunque para ser utilizado en la identificación y clasificación de objetos o fenómenos térmicos se debe agregar una base de datos con mucha más información.

Sin embargo, durante el desarrollo del modelo, se han identificado ciertas limitaciones que han afectado el desempeño del modelo. Se ha observado que el modelo presenta dificultades para distinguir con precisión entre clases que presentan características térmicas y visuales similares. Eso se debe a la falta de información y datos específicos para entrenar el modelo en estas situaciones particulares.

La ayuda de inteligencia artificial en la evaluación de la termografía de los transformadores de distribución puede proporcionar una detección más rápida y precisa de los posibles fallos y degradaciones en los aislamientos. La combinación de la termografía y la clasificación de severidad según su degradación térmica en los circuitos de transformadores de distribución pueden ayudar a prevenir daños mayores y a tomar medidas correctivas de manera oportuna, lo que resulta en una mayor eficiencia y confiabilidad del sistema.

Ahora bien, se utilizó un conjunto de 226 imágenes termográficas de los transformadores de distribución de la Empresa Eléctrica Azogues como referencia. A partir de esto, se construyó una matriz de vectores que contiene la base de datos para el entrenamiento y validación de la red mediante el aprendizaje profundo. Se llevó a cabo una clasificación de acuerdo con los diferentes niveles de severidad bajo la norma ANSI/NETA ATS.2021 para los circuitos de transformadores de distribución. Por otro lado, la Empresa Eléctrica Azogues determina el grado de gravedad basándose en la diferencia de temperatura entre los valores máximos y los de un cuerpo similar. De ahí que, el diseño consista en clasificar las imágenes termográficas en cinco niveles: Buen estado, Baja, Moderada, Grave y Extrema.

Finalmente, se diseñó un modelo estructural que se implementa en el software Python. El diseño contiene capas de convolución, activación (ReLU), normalización, agrupamiento, abandono (*dropout*) y está totalmente conectada para determinar la salida según la clasificación de cinco niveles de severidad. Asimismo, se estableció que el 72 % y 28 % de los datos se utilizaron para el entrenamiento y la validación de las redes neuronales convolucionales; además de que el modelo de predicción tiene una eficiencia del 70 %. Esto se debe a que se deberían obtener al menos 10 000 imágenes para el aprendizaje profundo,

en vista de que, si este modelo se enfrenta con una cantidad inferior a 500 imágenes, podría dar un resultado erróneo. Aunque se han logrado avances significativos en el modelo de clasificación de imágenes térmicas existen limitaciones que deben abordarse. La recopilación y la exploración de técnicas de preprocesamiento, es posible mejorar la eficacia y la precisión del modelo en futuras investigaciones.

RECOMENDACIONES

Para mejorar la eficacia del modelo, se recomienda recopilar un conjunto de datos más amplio y diverso. Esto permitirá una mejor comprensión de las variaciones en las condiciones de iluminación, ángulos de visión y posibles inferencias térmicas. Además, se sugiere explorar técnicas de preprocesamiento de imágenes térmicas, como la mejora de contrastes y la normalización, para mejorar la calidad y la discriminación de las características relevantes para la clasificación.

Otro aspecto a considerar es la necesidad de continuar investigando y actualizando el modelo con nuevos datos y técnicas a medida que estén disponibles. La tecnología en el campo de la clasificación de imágenes térmicas está en constante evolución, y es importante mantenerse actualizada para mejorar el rendimiento y la precisión del modelo.

Es importante realizar una investigación exhaustiva sobre los transformadores de distribución, la termografía y los métodos de clasificación de imágenes. El cual nos ayudará a comprender mejor el contexto y los desafíos asociados.

Antes de imponer datos a un modelo de clasificación, es necesario analizar correctamente cada imagen termográfica y dar una etiqueta. Existen varios algoritmos y arquitecturas de modelos de aprendizaje automático que podemos utilizar para la clasificación, debemos seleccionar el modelo más adecuado para el problema entrenarlo utilizando los datos termográficos etiquetados. Una vez entrenado el algoritmo, debemos evaluar su rendimiento utilizando datos de prueba. Podemos evaluar la capacidad de nuestro modelo con el que clasifica el grado de severidad de los transformadores.

Aguilar, D. A. (2022). *Diseño de un clasificador para el mantenimiento predictivo de una red de distribucion eléctrica utilizando Deep Learning.* TESIS PARA OPTAR GRADO DE DOCTOR EN INGENIERÍA , UNIVERSIDAD DE PIURA, Piura. Obtenido de https://pirhua.udep.edu.pe/bitstream/handle/11042/5995/DOC_ING_AUT_2204.pdf?sequence=7&isAllowed=y

Alava, J. A. (2022). *Diagnóstico Visual-Térmico En Sistemas Eléctricos De Subtransmisión Y Distribución Con El Uso De Drones Para Efectuar Mantenimientos.* UNIVERSIDAD POLITÉCNICA SALESIANA, Guayaquil. Obtenido de https://dspace.ups.edu.ec/bitstream/123456789/22503/1/UPS-GT003710.pdf

ANSI/NETA-ATS. (2021). *Standard for acceptance testing specifications for electrical power equipment and systems.* InterNational Electrical Testing Association.

Ávila-Tomas, J., Mayer, M., & Quesada, V. (2021). La inteligencia artificial y sus aplicaciones en medicina II: importancia actual y aplicaciones prácticas. *Atención Primaria, 53(1)*, 81-88. https://doi.org/10.1016/j.aprim.2020.04.014.

Bosch, A., Casas, J., & Lozano, T. (2019). *Deep Learning y Fundamentos.* Reverté-Aguilar.

Cadme, A. C. (2022). *DETECCIÓN DE FALLAS EN BAJANTES DE TRANSFORMADORES DE DISTRIBUCIÓN, MEDIANTE EL ANÁLISIS DEEP LEARNING, EN IMÁGENES TERMOGRÁFICAS.* UNIVERSIDAD POLITÉCNICA SALESIANA, CUENCA. Obtenido de https://dspace.ups.edu.ec/bitstream/123456789/21911/1/UPS-CT009579.pdf

Calle, P. (2023). *Alimentador 123 Mediciones Termográficas.* Empresa Eléctrica Azogues C.A.

Cañada, M.; Royo, R. (2016). *Termografía Infrarroja Nivel II.* Fundacion Confemetal.

Chapman, S. J. (s.f.). *Máquina Eléctricas.* Colombia: Mc Graw Hill.

Chauvin Arnoux. (2010). *Guía de la medición de aislamiento.* Obtenido de https://www.chauvin-arnoux.com/sites/default/files/documents/cat_guia_de_medicion_de_aislamiento.pdf

Cuauhtli, D., & Sánchez, F. (2009). *Estudio del ajuste y operación de la aplicación del relevador SEL-300G Aplicado a un generador síncrono.* Mexico D.F.: s.e.

Denio, H. (2012). *Aerial solar thermography and condition monitoring of photovoltaic systems.* IEEE Xplore. doi:10.1109/PVSC.2012.6317686

Empresa Eléctrica Quito S.A. (2014). *Normas para Sistemasde Distribucion Parte B.* Obtenido de

https://www.academia.edu/34775647/Normaspara_Sistemasde_Distribucion_Parte_B

Fluke. (2023). *Fluke Corporation*. Obtenido de https://www.fluke.com/es-ec

Gálvez C., A. d., & Cobián R., O. (2018). *Mantenimiento predictivo, preventivo y/o correctivo a transformadores de distribución.* Instituto Tecnológico de Tuxtla Gutiérrez, México. Obtenido de http://repositoriodigital.tuxtla.tecnm.mx/xmlui/bitstream/handle/123456789/1275/MDRPEICA2018016.pdf?sequence=1&isAllowed=y

Grainger , J., & Stevenson, W. (2001). *Analisis de sitemas de potencia .* Mexico: McGRAW-HILL.

Haya, J. C. (2015). *Aportaciones al estudio del envejecimiento de componentes dielectricos en transformadores de portencia.* Universidad de Cantabria, Santander. Obtenido de http://hdl.handle.net/10902/8112

Jafrouni, H., Elbreki, A., Almaktar, M., Zakariya, R., & Mohamed, F. A. (29-30 de Junio de 2023). Optimal Placement and Sizing of Static Var Compensators in Radial Distribution Networks Using Artificial Intelligence Techniques. *IEEE Xplore*(15), 01-06. doi:10.1109/ECAI58194.2023.10194155

Jiménez León , N. A., & Maza Pinza, J. G. (2008). *Implementación de un sistema basado en inteligencia artificial para la deteccion de regímenes anormales en transformadores de portencia.* Universidad Nacional de Loja, Loja. Obtenido de http://dspace.unl.edu.ec/jspui/handle/123456789/16716

Kicheriavenkov, A. A. (2021). *USING ARTIFICIAL INTELLIGENCE TO PREVENT DISTRIBUTION NETWORKS' ACCIDENTS.* Rusia: CIRED. Obtenido de USING_ARTIFICIAL_INTELLIGENCE_TO_PREVENT_DISTRIBUTION_NETWORKS_ACCIDENTS

Llanos, J. S. (2023). *Estado del arte de la inteligencia artificial en el sector ferroviario poniendo en común fabricante, operadora e infraestructura.* UNIVERSIDAD POLITÉCNICA DE MADRID, Madrid. Obtenido de https://oa.upm.es/76157/3/TFG_JUAN_SANTOS_GONZALEZ-LLANOS.pdf

Mathworks. (23 de 08 de 2023). Obtenido de https://la.mathworks.com/discovery/convolution.html

Mathworks. (23 de 08 de 2023). *Mathworks*. Obtenido de https://la.mathworks.com/discovery/convolution.html

Matich, D. J. (2001). *Redes Neuronales. Conceptos Básicos y Aplicaciones.* Universidad Tecnológica Nacional , Rosario. Obtenido de https://d1wqtxts1xzle7.cloudfront.net/36957218/redesneuronales-libre.pdf?1426217818=&response-content-disposition=inline%3B+filename%3DRedes_Neuronales_Conceptos_Basicos_y_Apl.pdf&Expires=1698256901&Signature=Q7x~FDQsUWSF5iNWCk1xfH3FJ6358I~n6kany1lO6r~Ojm

Mengyao, L. (22-24 de noviembre de 2021). Application of Artificial Intelligence Technology in Automatic Control of Electrical Engineering. *IEEE Xplore*. doi: 10.1109/ICESIT53460.2021.9696536

Minero, R. (2010). *PRUEBAS EN MAQUINAS SINCRONAS CONFORME A LA NORMA IEC 34 Y LA IEEE STD. 115.* Costa Rica: s.e.

Moreno M., C. D. (2013). *Optimización de recursos hardware para la operación de convolucion utilizada en el procesamiento digital de señales.* Instituto de Estudios de Posgrado, Departamento de Arquitectura de Computadores, Eléctronica y Tecnología Eléctronica. Córdoba: Instituto de Estudios de Posgrado. Obtenido de https://helvia.uco.es/bitstream/handle/10396/11518/2013000000885.pdf?sequence=1&isAllowed=y

Moya, R. (1990). *Curso de captacitación para operadores Central Paute Fase AB Generador Principal.* Cuenca: DONSI-INECEL.

Saldívar, J. (2018). *Estudio de niveles de eficiencia en transformadores de distribución en función del perfil de carga.* Obtenido de Instituto Tecnológico y de Estudios Superiores de Monterrey: https://repositorio.tec.mx/bitstream/handle/11285/632565/Tesis%20Eficiencia%20en%20Transformadores%20%28V0%29.pdf?sequence=1

Satellites. (2023). *Mapa de Azogues Ecuador.* Recuperado el 15 de julio de 2021, de https://satellites.pro/mapa_de_Azogues.Ecuador#B-2.746945,-78.853598,14

Torresí, A. (2020). *Ensayo de transformadores.* Científica Universitaria.

Weatherspark. (2023). *El clima promedio en Azogues Ecuador.* Obtenido de https://es.weatherspark.com/y/20017/Clima-promedio-en-Azogues-Ecuador-durante-todo-el-a%C3%B1o

UNIR. (2021). *Deep Learning: en qué consiste, ejemplos y aplicaciones*. Unir.Net. https://www.unir.net/ingenieria/revista/deep-learning/

Zamora, Manuel Carranza. “TERMO I: UN ESTUDIO DE LOS SISTEMAS TERMODINÁMICOS.” Universidad de Sevilla,1999.

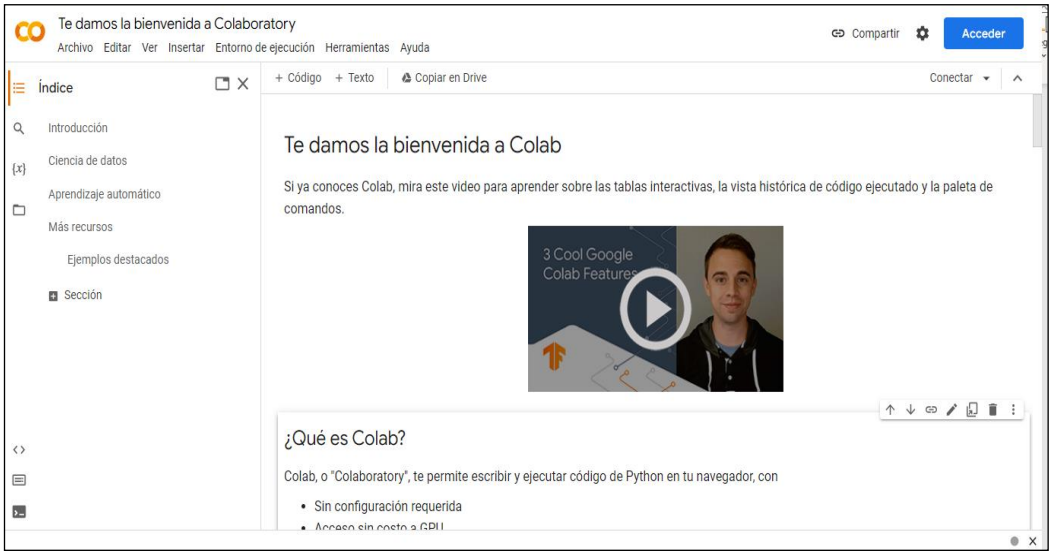
González, A., Antón, M., Fuertes, S., Blázquez, E., Alonso, A. "Implicación de la termografía en el diagnóstico de la..." (2007).

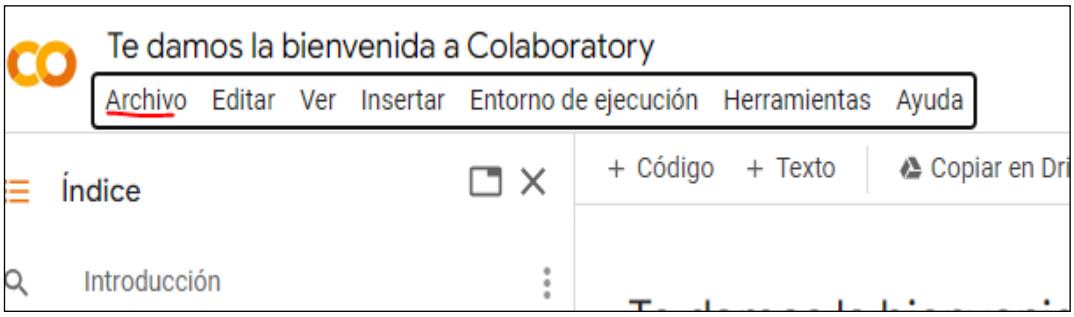
Neita, L., Peña, E. "Principios básicos de la termografía infrarroja y su utilización como técnica para..." (2011).

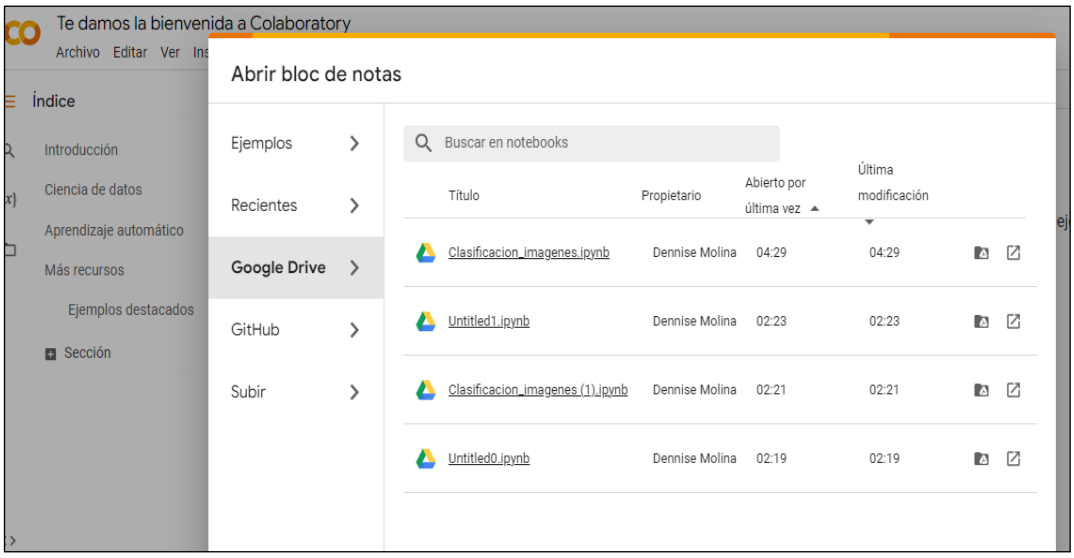
ANEXOS

Anexo 1. *Creación de cuenta en Google Colab.*

- Debido a que no se pudo instalar el programa de Python en el ordenadorse procedio a crear una cuenta en Google Colab. Google Colab es una plataforma basada en la nube que permite ejecutar y colaborar en proyectos de Python sin necesidad de instalar ningún software en el equipo local. Al utilizar Google Colab, se puede acceder a un entorno de desarrollo en Python completo y utilizar bibliotecas y herramientas populares para el análisis de datos y la programación en Python.



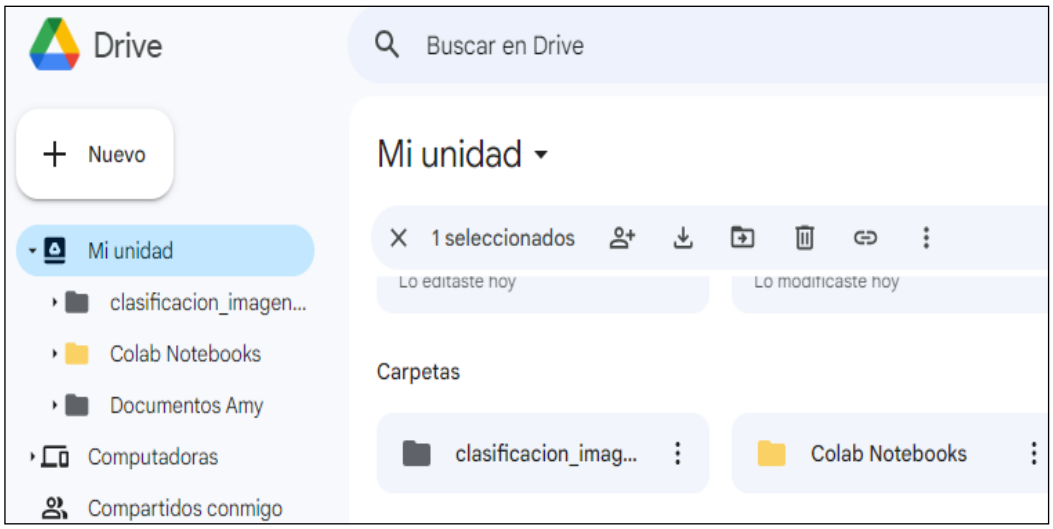




- Esta opción brinda una solución conveniente y accesible para aquellos que enfrentan dificultades con la instalación del programa de Python en los dispositivos. Con Google Colab, se pudo programar y trabajar en el modelo de predicción de manera rápida.

Anexo 2. *Base de datos en el Drive.*

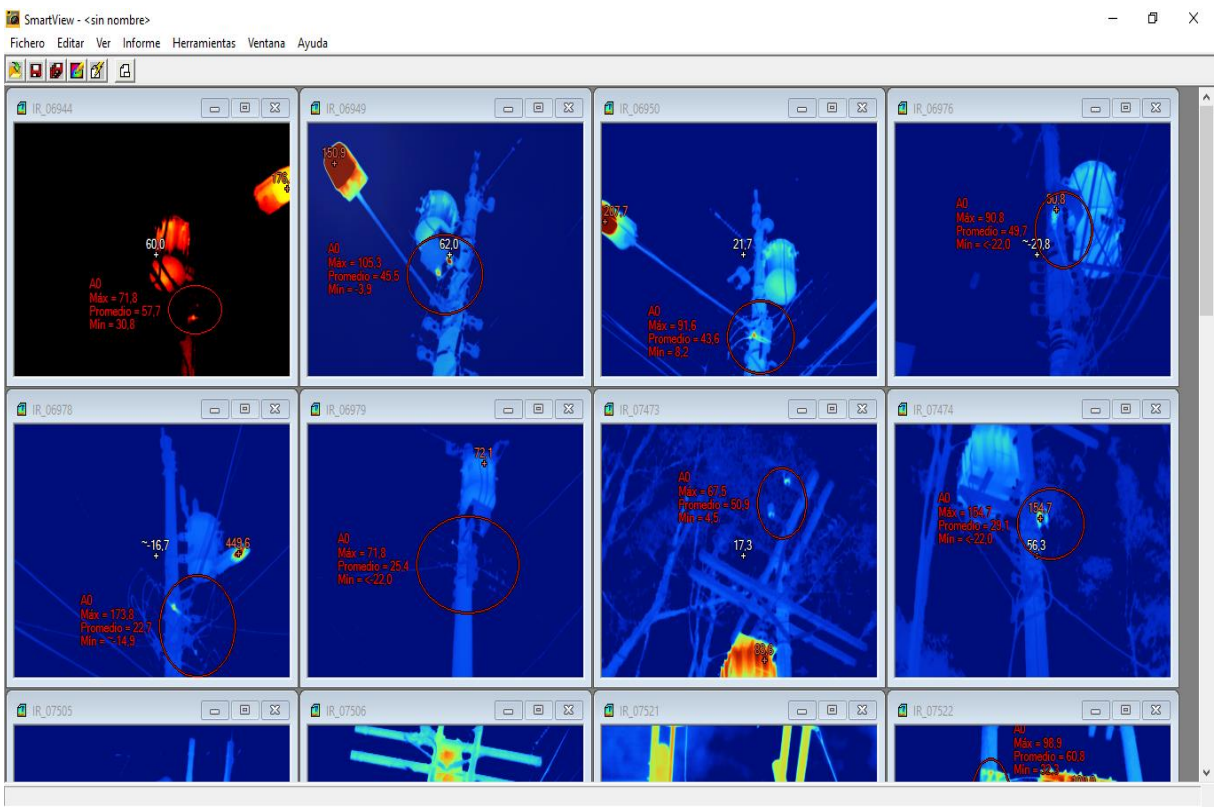
Para facilitar la colaboración y el acceso a datos en proyectos de Python, se ha optado por crear una base de datos en Google Drive y utilizarla en conjunto con Google Colab. Google Drive ofrece un espacio de almacenamiento en la nube que permite guardar y compartir archivos de manera segura. Al crear una base de datos en Google Drive, se puede acceder a ella desde cualquier dispositivo con conexión a internet y trabajar de forma colaborativa con otros miembros del equipo. Esta solución proporciona flexibilidad y conveniencia al permitir el acceso y la edición de la base de datos de manera remota, sin la necesidad de transferir archivos manualmente.

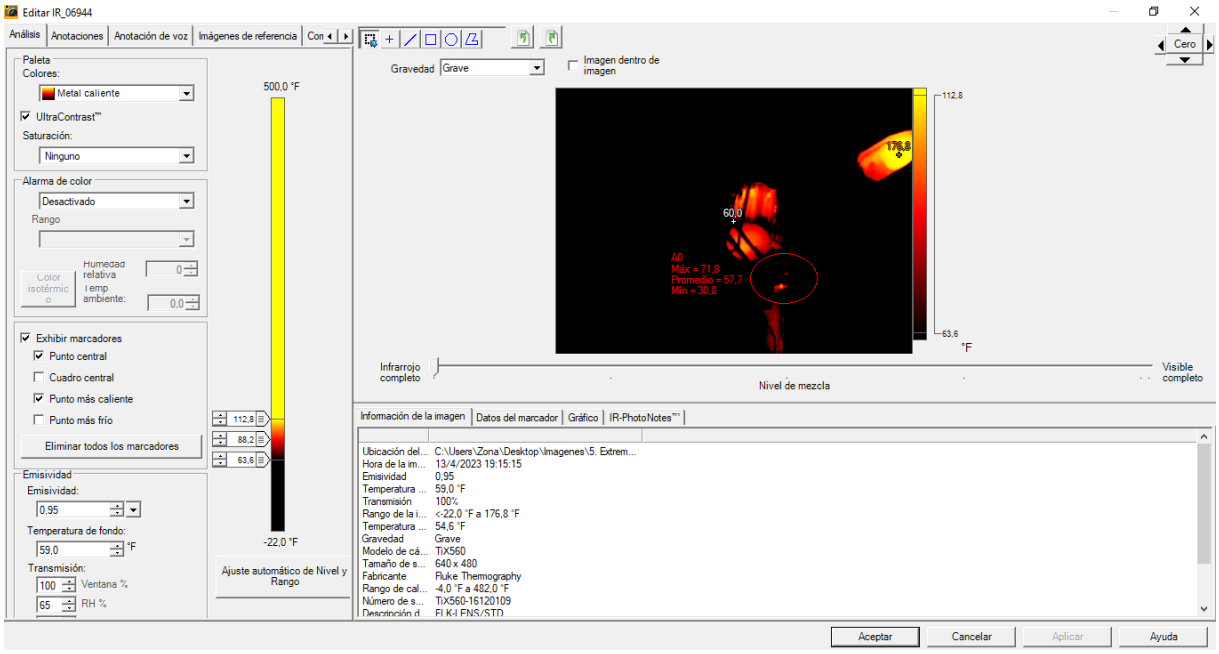


La creación de una base de datos en Google Drive y su uso en conjunto con Google Colab brinda una solución eficiente y práctica para trabajar en proyectos de Python de manera colaborativa. Al aprovechar la capacidad de almacenamiento en la nube de Google Drive y la potencia de procesamiento de Google Colab, se puede acceder y manipular la base de datos de forma remota, lo que facilita la colaboración y el intercambio de información entre los miembros del equipo.

Anexo 3. Análisis termográfico en la aplicación de Fluke SmartView.

El análisis termográfico es una herramienta fundamental en el campo de la termografía, y la aplicación Fluke SmartView ofrece una solución completa y eficiente para llevar a cabo este tipo de análisis. Fluke SmartView es una aplicación de software diseñada para visualizar, analizar y reportar datos termográficos obtenidos con cámaras de infrarrojos de Fluke. Con esta aplicación, los profesionales pueden realizar un análisis detallado de las imágenes térmicas capturadas, identificar patrones de temperatura anormales y tomar decisiones informadas para el mantenimiento predictivo y la resolución de problemas. , se presentan los beneficios y características clave de la aplicación Fluke SmartView para el análisis termográfico





El análisis termográfico desempeña un papel crucial en la detección temprana de problemas y en la toma de decisiones informadas en una amplia gama de aplicaciones. La aplicación Fluke SmartView proporciona una plataforma intuitiva y poderosa para realizar este análisis de manera eficiente y Preciso. Con su conjunto de herramientas avanzadas y capacidades de visualización, los profesionales pueden aprovechar al máximo los datos termográficos y obtener información valiosa para el mantenimiento preventivo y predictivo. Fluke SmartView se ha convertido en una solución confiable y ampliamente utilizada en la industria, brindando a los usuarios la capacidad de realizar análisis termográficos de manera efectiva y mejorar la eficiencia y confiabilidad de sus operaciones.

Anexo 4. *Clasificación según su grado de severidad.*

La creación de un modelo de clasificación según el grado de severidad de degradación térmica en Python es un proceso fundamental para el análisis y la predicción de problemas relacionados con la temperatura en diversos materiales. Este modelo permite categorizar y evaluar la gravedad de la degradación térmica en función de diferentes parámetros y características. Al utilizar Python o Google Colab como lenguaje de programación, se pueden aplicar algoritmos y técnicas de aprendizaje automático para entrenar y validar el modelo, lo que proporciona resultados precisos y confiables. A continuación, se presentan los pasos y consideraciones claves para la creación de este modelo de clasificación.

```
from google.colab import drive #Importamos la biblioteca necesaria para ejecutar Google Drive
drive.mount('/content/drive') #Conectamos Google Drive en la ubicación '/content/drive'

#A partir de este punto, podemos acceder a los archivos de Google Drive
# y trabajar con ellos en nuestro entorno de colab
```

```
import tensorflow as tf # Importamos das bibliotecas necesarias
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D
from tensorflow.keras.layers import Activation, Dropout, Flatten, Dense
import matplotlib.pyplot as plt
```

```
train_data_dir = 'drive/MyDrive/clasificacion_imagenes/training' # Directorio de datos de entrenamiento
validation_data_dir = 'drive/MyDrive/clasificacion_imagenes/validation' # Directorio de datos de validación
```

```
epochs = 50 # Número de épocas de entrenamiento
batch_size = 16 # Tamaño del lote (batch size)
img_width, img_height = 150, 150 # Ancho y alto de las imágenes
```

```
#Crear un generador de datos de imágenes
train_datagen = ImageDataGenerator(
    rescale=1.0/255, # Escalar los valores de pixeles en el rango de [0,1]
    rotation_range=30, # Rango de rotación aleatoria de las imágenes en grados
    width_shift_range=0.2, # Rango de desplazamiento horizontal aleatorio de las imágenes como fraccion de la anchura total
    height_shift_range=0.2, # Rango de desplazamiento vertical aleatorio de las imagenes como fraccion de la altura total
    shear_range=0.2, # Rango de deformación de cizallamiento aleatoria
    zoom_range=0.2, # Rango de zoom aleatorio de las imágenes
    horizontal_flip=True, # Voltear horizontalmente las imágenes de forma aleatoria
    fill_mode='nearest') #Estrategia de relleno para los pixeles que se crean durante las transformaciones

validation_datagen = ImageDataGenerator(rescale=1.0/255)
```

```
# Crear un generador de flujo de datos de imágenes para entrenamiento
train_generator = train_datagen.flow_from_directory(
    train_data_dir, # Directorio que contiene las imágenes de entrenamiento
    target_size=(img_width, img_height), # Tamaño al que se redimensionarán las imágenes
    batch_size=batch_size, # Tamaño del lote de imágenes que se generará en cada iteración
    class_mode='categorical') # Modo de clasificación, en este caso, se utiliza 'Categorical' para clasificación multiclase
```

```
Found 177 images belonging to 5 classes.
```

```
model = Sequential() # Crear un modelo secuencial

model.add(Conv2D(32, (3, 3), input_shape=(img_width, img_height, 3))) # Agregar una capa de convolución con 32 filtros de tamaño 3x3 y esp. la forma de entrada
model.add(Activation('relu')) #Agregar una capa de activación ReLu
model.add(MaxPooling2D(pool_size=(2, 2))) # Agregar una capa de agrupación maxima con un tamaño de agrupación de 2x2

model.add(Conv2D(64, (3, 3))) # Agregar otra capa de convolución con 64 filtros de tamaño 3x3
model.add(Activation('relu')) #Agregar una capa de activación ReLu
model.add(MaxPooling2D(pool_size=(2, 2))) # Agregar otra capa de agrupación maxima con un tamaño de agrupación de 2x2

model.add(Conv2D(128, (3, 3))) # Agregar otra capa de convolución con 128 filtros de tamaño 3x3
model.add(Activation('relu')) #Agregar una capa de activación ReLu
model.add(MaxPooling2D(pool_size=(2, 2))) # Agregar otra capa de agrupación maxima con un tamaño de agrupación de 2x2

model.add(Flatten()) # Aplanar la salida de las capas anteriores
model.add(Dense(64)) # Agregar una capa totalmente conectada con 64 unidades
model.add(Activation('relu')) #Agregar una capa de activación ReLu
model.add(Dropout(0.5)) # Agregar una capa de dropout con una tasa de dropout del 50%
model.add(Dense(5)) # Agregar otra capa completamente conectada con 5 unidades (correspondiente al número de clase)
model.add(Activation('softmax'))# Agregar una capa de activación softmax
```

```
model.compile(loss='categorical_crossentropy', # Compilar el modelo con una función de pérdida de entropía cruzada categórica
              optimizer='adam', # Utilizar el optimizador Adam para ajustar los pesos del modelo durante el entrenamiento
              metrics=['accuracy']) # Calcular la metrica de precisión durante el entrenamiento y la evaluación del modelo
```

```
history = model.fit(train_generator, # Entrenar el modelo utilizando el generador de flujo de datos de imágenes de entrenamiento
                    steps_per_epoch=train_generator.samples // batch_size, #Especificar el número de pasos por época basado en el tamaño del lote y el número total de muestras de
                    epochs=epochs, # Especificar el número de épocas de entrenamiento
                    validation_data=validation_generator, # Utilizar el generador de flujo de datos de imágenes de validación para la validación durante el entrenamiento
                    validation_steps=validation_generator.samples // batch_size) # Especificar el número de pasos de validación basado en el tamaño del lote y el número total de m
```

```
model.save('image_model_v1.h5') # Guardar el modelo entrenado en un archivo h5
```

```
Epoch 1/50
11/11 [==============================] - 12s 959ms/step - loss: 1.6948 - accuracy: 0.2733 - val_loss: 1.5830 - val_accuracy: 0.1250
Epoch 2/50
11/11 [==============================] - 11s 979ms/step - loss: 1.6071 - accuracy: 0.2174 - val_loss: 1.5307 - val_accuracy: 0.1406
Epoch 3/50
11/11 [==============================] - 10s 927ms/step - loss: 1.5628 - accuracy: 0.2981 - val_loss: 1.5150 - val_accuracy: 0.1094
Epoch 4/50
11/11 [==============================] - 11s 952ms/step - loss: 1.5745 - accuracy: 0.2671 - val_loss: 1.5788 - val_accuracy: 0.1406
Epoch 5/50
11/11 [==============================] - 10s 1s/step - loss: 1.5800 - accuracy: 0.2671 - val_loss: 1.6694 - val_accuracy: 0.1562
Epoch 6/50
11/11 [==============================] - 10s 781ms/step - loss: 1.5411 - accuracy: 0.2671 - val_loss: 1.6081 - val_accuracy: 0.2031
Epoch 7/50
11/11 [==============================] - 11s 1000ms/step - loss: 1.5345 - accuracy: 0.3478 - val_loss: 1.5513 - val_accuracy: 0.1250
Epoch 8/50
11/11 [==============================] - 11s 961ms/step - loss: 1.6325 - accuracy: 0.2330 - val_loss: 1.5959 - val_accuracy: 0.1250
Epoch 9/50
11/11 [==============================] - 10s 938ms/step - loss: 1.5695 - accuracy: 0.2733 - val_loss: 1.5703 - val_accuracy: 0.1562
Epoch 10/50
11/11 [==============================] - 11s 997ms/step - loss: 1.5350 - accuracy: 0.3230 - val_loss: 1.4944 - val_accuracy: 0.1562
Epoch 11/50
11/11 [==============================] - 9s 770ms/step - loss: 1.5367 - accuracy: 0.2795 - val_loss: 1.5108 - val_accuracy: 0.1719
Epoch 12/50
11/11 [==============================] - 10s 844ms/step - loss: 1.4543 - accuracy: 0.3540 - val_loss: 1.6454 - val_accuracy: 0.1562
Epoch 13/50
```

```
acc = history.history['accuracy'] # Obtener el historial de precisión de entrenamiento
val_acc = history.history['val_accuracy'] # Obtener el historial de precisión de validación
loss = history.history['loss'] # Obtener el historial de pérdida de entrenamiento
val_loss = history.history['val_loss'] # Obtener el historial de pérdida de validación

epochs_range = range(1, epochs + 1) # Crear un rango de épocas para el eje x del gráfico

plt.figure(figsize=(12, 4)) # Crear una figura de tamaño 12x4
plt.subplot(1, 2, 1) # Crear un subplot en la posición 1 de una fila y dos columnas
plt.plot(epochs_range, acc, label='Accuracy') # Graficar la precisión de entrenamiento en función de las epocas
plt.plot(epochs_range, val_acc, label='Validation Accuracy') # Graficar la precisión de validación en función de las epocas
plt.legend(loc='lower right') # Mostrar una leyenda en la esquina inferior derecha del gráfico
plt.title('Accuracy vs. Validation Accuracy') #Establecer el título del gráfico
plt.xlabel('Epoch') # Etiquetar el eje x del gráfico como 'Epoch'
plt.ylabel('Accuracy') # Etiquetar el eje y del gráfico como 'Accuracy'

plt.subplot(1, 2, 2) # Crear un subplot en la precisión 2 de una fila y dos columnas
plt.plot(epochs_range, loss, label='Loss') # Graficar la pérdida de entrenamiento en función de las épocas
plt.plot(epochs_range, val_loss, label='Validation Loss') # Graficar la pérdida de validación en función de las épocas
plt.legend(loc='upper right') # Mostrar una leyenda en la esquina superior derecha del gráfico
plt.title('Loss vs. Validation Loss') # Establecer el título del gráfico
plt.xlabel('Epoch') # Etiquetar el eje x del gráfico como 'Epoch'
plt.ylabel('Loss') # Etiquetar el eje y del gráfico como 'Loss'

plt.show() # Mostrar el gráfico
```

```
validation_predictions = model.predict(validation_generator) # Realizar predicciones en el conjunto de validación utilizando el modelo entrenado
true_labels = validation_generator.classes # Obtener las etiquetas verdaderas del conjunto de validación

rmse = np.sqrt(np.mean((validation_predictions.argmax(axis=1) - true_labels) ** 2))  # Calcular la raíz del error cuadrático medio (RMSE) entre las predicciones y
print(f'RMSE: {rmse:.2f}') # Imprimir el valor de RMSE formateado con dos decimales

mae = np.mean(np.abs(validation_predictions.argmax(axis=1) - true_labels)) # Calcular el error absoluto medio (MAE) entre las predicciones y las etiquetas verdade
print(f'MAE: {mae:.2f}') # Imprimir el valor del MAE formateado con dos decimales
```

```
5/5 [==============================] - 1s 178ms/step
RMSE: 1.78
MAE: 1.48
```

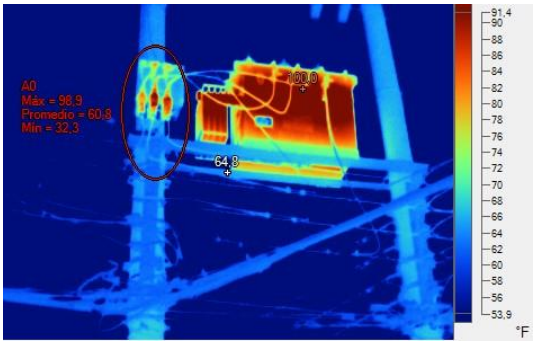
```
from tensorflow.keras.preprocessing import image # Importar el módulo de procesamiento de imágenes de keras
import numpy as np # Importar el módulo numpy para trabajar con matrices

model = tf.keras.models.load_model('image_model_v1.h5') # Cargar el modelo guardado desde el archivo 'image_model_v1.h5'
```

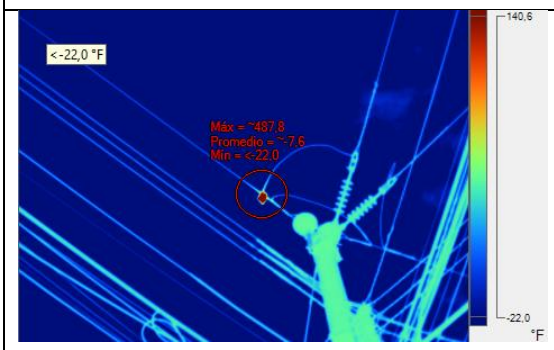
```
new_image_path = 'drive/MyDrive/clasificacion_imagenes/test.PNG'  # Ruta de la imagen

img = image.load_img(new_image_path, target_size=(150, 150)) #Cargar la imagen utilizando la función load_img de keras y redimensionarla a una tamaño de 150x150 píxeles
img_array = image.img_to_array(img) # Convertir la imagen en un arreglo de numpy
img_array = np.expand_dims(img_array, axis=0) # Agregar una dimensión adicional al arreglo para que coincida con el formato de entrada del modelo
img_array /= 255.0 # Normalizar los valores de los pixeles de la imagen dividiendolos por 255.0 para que estén en el rango de [0,1]

predictions = model.predict(img_array) # Realizar predicciones en la imagen utilizando el modelo cargado

class_index = np.argmax(predictions) # Obtener el índice de la clase con mayor probabilidad de predicción
classes = ['buen estado', 'bajo', 'moderada', 'grave', 'extremo'] # Definir las clases posibles de predicción
predicted_class = classes[class_index] # Obtener la clase predicha correspondiente al índice

print(f'La imagen se clasifica como: {predicted_class}') # Imprimir la clase predicha de la imagen
```

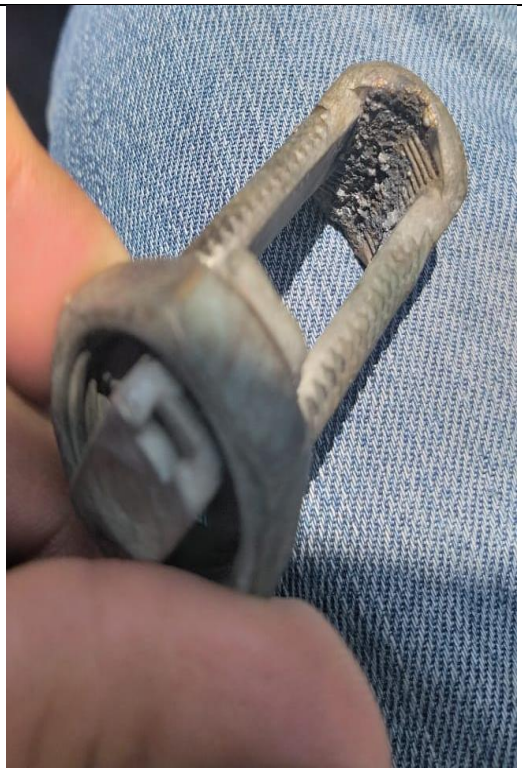
La creación de un modelo de clasificación según el grado de severidad de degradación térmica en Python es una herramienta valiosa para identificar y predecir problemas relacionados con la temperatura en diversos materiales. Este modelo permite una evaluación precisa y objetiva de la gravedad de la degradación térmica, lo que facilita la toma de decisiones informadas para el mantenimiento y la prevención de fallas. Al utilizar técnicas de aprendizaje automático y algoritmos en Python, se puede lograr un alto nivel de precisión en la clasificación de la severidad de la degradación térmica. La creación de este modelo representa un avance significativo en la detección temprana y la gestión eficiente de problemas relacionados con la temperatura en diversos sectores industriales.

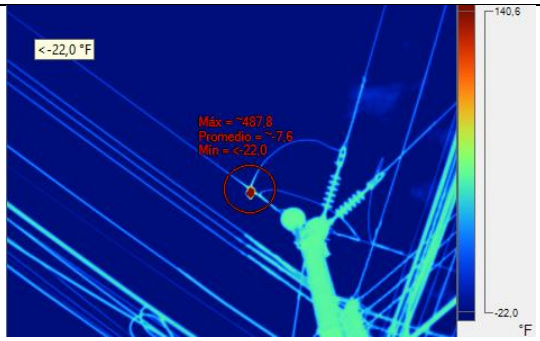
Anexo 5. *Piezas deterioradas por un grado de severidad extrema en los transformadores de distribución.*

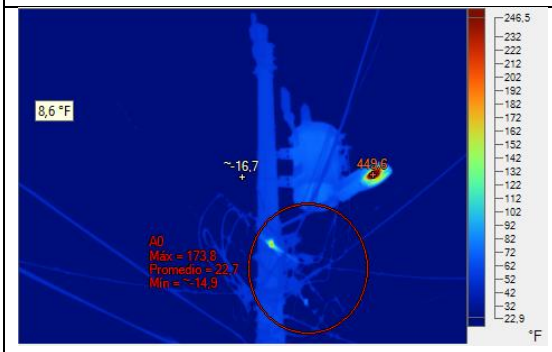
El grado de severidad extrema en los transformadores de distribución puede ocasionar daños significativos en las piezas deterioradas. Es crucial comprender y evaluar adecuadamente estos daños para tomar medidas correctivas y evitar posibles fallas en el sistema eléctrico. En este contexto, se ha realizado un análisis exhaustivo utilizando técnicas de evaluación de severidad extrema en Python. Este modelo de clasificación permite identificar y categorizar de manera precisa el grado de severidad de los daños en las piezas de los transformadores de distribución. A continuación, se presentan los resultados obtenidos y su importancia para el mantenimiento y la seguridad del sistema eléctrico.

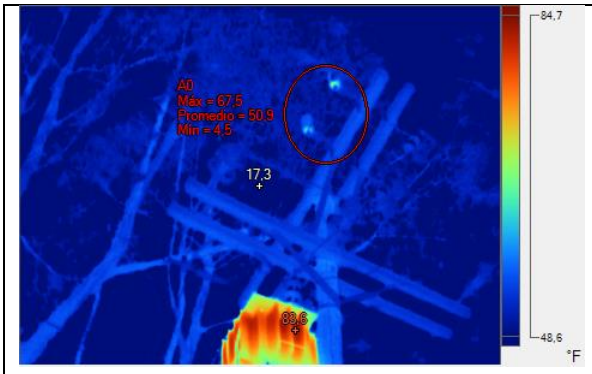

A0
Máx = 98,9
Promedio = 60,8
Mín = 32,3
100,0
64,8
91,4
90
88
86
84
82
80
78
76
74
72
70
68
66
64
62
60
58
56
53,9
°F


<-22,0 °F
Máx = ~487,8
Promedio = ~-7,6
Mín = <-22,0
140,6
-22,0
°F




<-22,0 °F
Máx = ~487.8
Promedio = ~-7.6
Mín = <-22.0
140,6
-22,0
°F


8,6 °F
~-16,7
449,6
A0
Máx = 173.8
Promedio = 22.7
Mín = ~-14.9
246,5
232
222
212
202
192
182
172
162
152
142
132
122
112
102
92
82
72
62
52
42
32
22,9
°F



El análisis de severidad extrema en los transformadores de distribución y los daños resultantes en las piezas deterioradas son aspectos críticos para garantizar la confiabilidad y seguridad del sistema eléctrico. Gracias al modelo de clasificación desarrollado en Python, se ha logrado evaluar y categorizar de manera precisa el grado de severidad de los daños.

Estos resultados proporcionan información valiosa para la toma de decisiones informadas en cuanto al mantenimiento preventivo y correctivo de los transformadores. Al comprender y abordar adecuadamente los daños en las piezas deterioradas, se pueden evitar posibles fallas y garantizar un funcionamiento óptimo del sistema eléctrico.

AUTORIZACION DE PUBLICACION EN EL REPOSITORIO INSTITUCIONAL

Yo, Diana Dennise Molina Farez portador de la cédula de ciudadanía N.º 1400770127. En calidad de autor/a y titular de los derechos patrimoniales del trabajo de titulación “Estudio para incorporación de inteligencia artificial en circuitos de transformadores de distribución para análisis de demanda, enlace y análisis de información” de conformidad a lo establecido en el artículo 114 Código Orgánico de la Economía Social de los Conocimientos, Creatividad e Innovación, reconozco a favor de la Universidad Católica de Cuenca una licencia gratuita, intransferible y no exclusiva para el uso no comercial de la obra, con fines estrictamente académicos, Así mismo; autorizo a la Universidad para que realice la publicación de este trabajo de titulación en el Repositorio Institucional de conformidad a lo dispuesto en el artículo 144 de la Ley Orgánica de Educación Superior.

Cuenca, 07 de noviembre de 2023

F: ……..

Diana Dennise Molina Farez

1400770127