Creación de una aplicación que permita el reconocimiento de los elementos del diagrama de un circuito eléctrico básico y sus interconexiones utilizando inteligencia artificial

Sambache Pelaez, Eddy Fernando

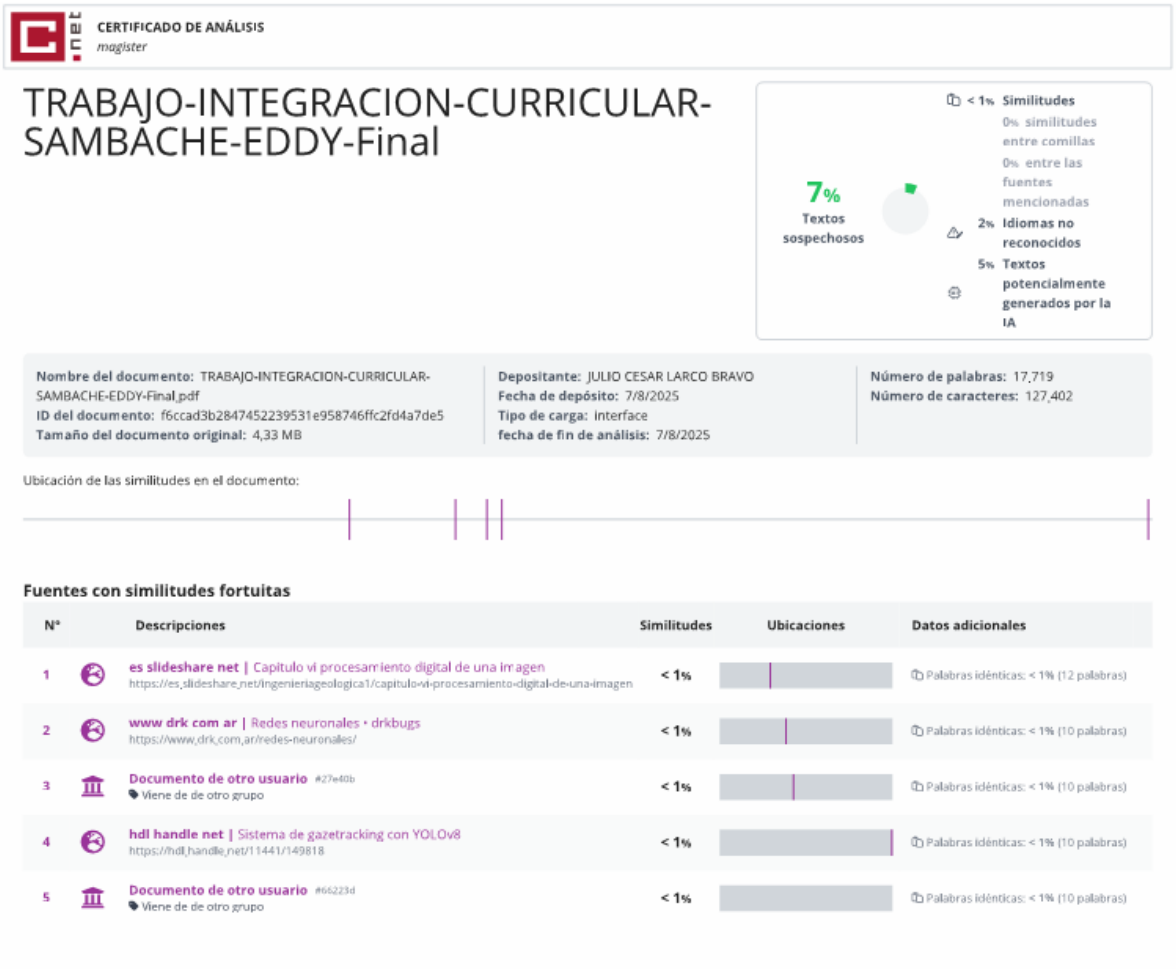
Departamento de Eléctrica, Electrónica y Telecomunicaciones

Carrera de Electrónica y Automatización

Trabajo de integración curricular, previo a la obtención del título de Ingeniero en Electrónica y Automatización

Ing. Larco Bravo, Julio César MSc.

7 de agosto de 2025



CERTIFICADO DE ANÁLISIS
magister

TRABAJO-INTEGRACION-CURRICULAR-SAMBACHE-EDDY-Final

7% Textos sospechosos

< 1% Similitudes
0% similitudes entre comillas
0% entre las fuentes mencionadas
2% Idiomas no reconocidos
5% Textos potencialmente generados por la IA

Nombre del documento: TRABAJO-INTEGRACION-CURRICULAR-SAMBACHE-EDDY-Final.pdf
ID del documento: f6ccad3b28474522395 31e958746ffc2fd4a7de5
Tamaño del documento original: 4,33 MB

Depositante: JULIO CESAR LARCO BRAVO
Fecha de depósito: 7/8/2025
Tipo de carga: Interface
fecha de fin de análisis: 7/8/2025

Número de palabras: 17,719
Número de caracteres: 127,402

Ubicación de las similitudes en el documento:

Fuentes con similitudes fortuitas

N°	Descripciones	Similitudes	Ubicaciones	Datos adicionales
1	es slideshare net | Capitulo vi procesamiento digital de una imagen https://es.slideshare.net/ingenieriageologica1/capitulo-vi-procesamiento-digital-de-una-imagen	< 1%		Palabras idénticas: < 1% (12 palabras)
2	www drk com ar | Redes neuronales • drkbugs https://www.drk.com.ar/redes-neuronales/	< 1%		Palabras idénticas: < 1% (10 palabras)
3	Documento de otro usuario #27e40b Viene de de otro grupo	< 1%		Palabras idénticas: < 1% (10 palabras)
4	hdl handle net | Sistema de gazetracking con YOLOv8 https://hdl.handle.net/11441/149818	< 1%		Palabras idénticas: < 1% (10 palabras)
5	Documento de otro usuario #66223d Viene de de otro grupo	< 1%		Palabras idénticas: < 1% (10 palabras)

Firmado electrónicamente por:
JULIO CESAR LARCO BRAVO
Validar únicamente con FirmaEC

Ing. Larco Bravo, Julio César MSc.

Director

Departamento de Eléctrica, Electrónica y Telecomunicaciones

Carrera de Electrónica y Automatización

Certificación

Certifico que el trabajo de integración curricular: "**Creación de una aplicación que permita el reconocimiento de los elementos del diagrama de un circuito eléctrico básico y sus interconexiones utilizando inteligencia artificial**" fue realizado por el señor **Sambache Pelaez, Eddy Fernando,** el mismo que cumple con los requisitos legales, teóricos, científicos, técnicos y metodológicos establecidos por la Universidad de las Fuerzas Armadas ESPE, además fue revisado y analizada en su totalidad por la herramienta de prevención y/o verificación de similitud de contenidos; razón por la cual me permito acreditar y autorizar para que se lo sustente públicamente.

Sangolquí, 7 de agosto de 2025

Firmado electrónicamente por:
JULIO CESAR LARCO BRAVO
Validar únicamente con FirmaEC

Ing. Larco Bravo, Julio César MSc.

Director

C.C.: 1710638808

Departamento de Eléctrica, Electrónica y Telecomunicaciones

Carrera de Electrónica y Automatización

Responsabilidad de Autoría

Yo, **Sambache Pelaez, Eddy Fernando**, con cédula de ciudadanía n°1751154996, declaro que el contenido, ideas y criterios del trabajo de integración curricular: **Creación de una aplicación que permita el reconocimiento de los elementos del diagrama de un circuito eléctrico básico y sus interconexiones utilizando inteligencia artificial** es de *mi autoría* y responsabilidad, cumpliendo con los requisitos legales, teóricos, científicos, técnicos, y metodológicos establecidos por la Universidad de las Fuerzas Armadas ESPE, respetando los derechos intelectuales de terceros y referenciando las citas bibliográficas.

Sangolquí, 7 de agosto de 2025

Sambache Pelaez, Eddy Fernando

C.C.: 1751154996

Departamento de Eléctrica, Electrónica y Telecomunicaciones

Carrera de Electrónica y Automatización

Autorización de Publicación

Yo, **Sambache Pelaez, Eddy Fernando**, con cédula de ciudadanía n°1751154996, autorizo a la Universidad de las Fuerzas Armadas ESPE publicar el trabajo de integración curricular: **Creación de una aplicación que permita el reconocimiento de los elementos del diagrama de un circuito eléctrico básico y sus interconexiones utilizando inteligencia artificial** *en el* Repositorio Institucional, cuyo contenido, ideas y criterios son de mi/nuestra responsabilidad.

Sangolquí, 7 de agosto de 2025

Sambache Pelaez, Eddy Fernando

C.C.: 1751154996

Dedicatoria

Este trabajo está dedicado a mi familia, quienes han sido un pilar importante en mi desarrollo como persona y profesional. Su amor incondicional me ha valido de fuerzas para vencer cualquier adversidad presente a lo largo de mi camino.

A mis padres Fernando y Margarita, quienes lo son todo para mí y han estado presentes en cada etapa de mi vida motivándome a siempre alcanzar mis metas, por más difíciles que estas sean.

A mis hermanos, quienes con su ejemplo me inspiran a superarme cada día. Y a mis queridos sobrinos, con la ilusión de que este logro los motive a nunca rendirse y crean en su capacidad para alcanzar sus metas.

Eddy Fernando Sambache Pelaez

Agradecimiento

En primer lugar, doy gracias a Dios por bendecirme con salud, cuidarme siempre y ser guía a lo largo de mi vida y por darme una familia estupenda que me brinda su apoyo de forma incondicional.

Agradezco a mis padres Fernando y Margarita, por su amor incondicional y ser mi sustento a lo largo de este camino, por estar conmigo en cada paso que doy apoyándome en bueno y malos momentos, motivándome a siempre salir adelante. A mis hermanos, por siempre estar pendientes de mí, agradezco su apoyo y palabras de aliento que me han motivado a seguir adelante.

Y agradezco al Ing. Julio Cesar Larco Bravo, quien ha sido un excelente guía en el desarrollo de este trabajo y gracias a su amplio conocimiento y experiencia me ha permitido llevarlo a cabo sin mayores complicaciones.

Eddy Fernando Sambache Pelaez

Índice de contenidos

Dedicatoria 6

Agradecimiento 7

Resumen 15

Capítulo I 17

Introducción 17

Antecedentes 17

Justificación 19

Alcance del Proyecto 20

Objetivos 23

Objetivo General 23

Objetivos Específicos 23

Capítulo II 24

Marco Teórico 24

Procesamiento Digital de Imágenes 24

Descripción General 24

Pasos para el procesamiento digital de imágenes 25

Morfología de Imágenes 28

Conceptos básicos 28

Operaciones fundamentales 29

Operaciones combinadas 30

Algoritmos morfológicos útiles 32

Redes Neuronales 35

Definición 35

Elementos Básicos 37

Aprendizaje y validación de la red neuronal 38

Topologías de redes neuronales 40

Aplicaciones 42

Software y Recursos Tecnológicos 44

Python 44

OpenCV 45

Métodos de Anotación y Etiquetado de Datos 45

Frameworks y Librerías para IA 47

Capítulo III 48

Diseño e Implementación 48

Selección de la arquitectura de detección de objetos 48

Metodología 50

Preparación de la base de datos 51

Etiquetado de datos 55

Detección de componentes eléctricos 62

Creación del proyecto 62

Entrenamiento de la red neuronal 63

Detección de nodos y conexiones 67

Detección de componentes eléctricos 68

Aplicación de filtros 69

Detección de nodos 70

Detección de terminales 72

Detección de conectividad 73

Detección e interpretación de texto 75

Detección de texto con PaddleOCR 76

Enlace texto – componentes eléctricos 77
Generación de archivos descriptivos 78
Simulación del circuito 80
Desarrollo de la aplicación 81
Área de visualización 83
Panel de operación 84
Acceso directo a la aplicación 84
Capítulo IV 86
Pruebas y Evaluación de Resultados 86
Evaluación del modelo de detección 86
Evaluación del modelo. 87
Pruebas de detección 89
Evaluación de la detección de texto 92
Evaluación de la funcionalidad de la aplicación 93
Pruebas funcionales de la interfaz gráfica 94
Simulación Netlist en Softwares con motor Spice 97
Capítulo V 100
Conclusiones y Recomendaciones 100
Conclusiones 100
Recomendaciones y Trabajos Futuros 101
Referencias 103
Apéndices 108

Índice de tablas

Tabla 1 Elementos de un circuito eléctrico básico 21

Tabla 2 Comparación de arquitecturas de inteligencia artificial para detección de objetos 49

Tabla 3 Etiquetas numéricas de las clases correspondiente a cada elemento 53

Tabla 4 Cantidad de imágenes por clase en la base de datos inicial 54

Tabla 5 Asignación de nombres acorde a la clase 75

Tabla 6 Funciones de la aplicación 84

Tabla 7 Parámetros de entrenamiento del modelo de detección 86

Tabla 8 Características del computador 87

Tabla 9 Métricas de evaluación del modelo de detección 88

Tabla 10 Comparación de resultados de la simulación del netlist 99

Índice de figuras

Figura 1 Procesamiento del diagrama de un circuito eléctrico 20

Figura 2 Imagen de 16 pixeles 25

Figura 3 Pasos fundamentales en el procesamiento digital de imágenes 26

Figura 4 Digitalización de objetos y elementos estructurantes 29

Figura 5 Ejemplo del proceso de erosión 29

Figura 6 Ejemplo del proceso de dilatación 30

Figura 7 Ejemplo del proceso de apertura 31

Figura 8 Ejemplo del proceso de cierre 32

Figura 9 Ejemplo del algoritmo de extracción de bordes 33

Figura 10 Ejemplo del algoritmo de relleno de huecos 33

Figura 11 Ejemplo del algoritmo de componentes conexos 34

Figura 12 Neurona biológica 36

Figura 13 Modelo de una neurona artificial 37

Figura 14 Etapas para la digitalización y simulación del diagrama de un circuito eléctrico 50

Figura 15 Ejemplo de imágenes de la base de datos 52

Figura 16 Captura de diagramas de circuitos eléctricos 54

Figura 17 Clases para el etiquetado de datos 56

Figura 18 Base de datos cargada en Roboflow 56

Figura 19 Clases para el etiquetado de datos 57

Figura 20 Preparación del dataset 58

Figura 21 Total de imágenes del dataset 59

Figura 22 Descarga de la base de datos 59

Figura 23 Anotaciones de las imágenes contenidas en los archivos de texto 60

Figura 24 Ejemplo de las anotaciones contenidas en los archivos de texto 61

Figura 25 Creación de un nuevo proyecto en el IDE PyCharm 62

Figura 26 Instalación de la librería ultralytics en el entorno virtual del proyecto 64

Figura 27 Estructura de la carpeta dataset para el entrenamiento..... 64

Figura 28 Fragmento de código utilizado para entrenar la red neuronal yolov11 66

Figura 29 Validación del modelo una vez finalizado su entrenamiento..... 67

Figura 30 Diagrama de flujo para la detección de nodos y conexiones 68

Figura 31 Inferencia del modelo para la detección de elementos un diagrama de circuito eléctrico..... 69

Figura 32 Ejemplo de la aplicación de filtros a un diagrama de circuito eléctrico 70

Figura 33 Visualización del componente conexo más grande 71

Figura 34 Eliminación de componentes eléctricos del circuito 71

Figura 35 Detección de nodos del circuito eléctrico..... 72

Figura 36 Creación de regiones de interés 72

Figura 37 Detección de terminales de cada componente eléctrico 73

Figura 38 Lista de conexiones..... 74

Figura 39 Filtrado para detección de texto..... 76

Figura 40 Detección de texto enmarcado por sus bounding boxes..... 77

Figura 41 Comparación distancias euclidianas entre los centros de cada texto y los componentes..... 78

Figura 42 Lista de detalle del diagrama de un circuito eléctrico..... 78

Figura 43 Archivos descriptivos del diagrama de un circuito eléctrico 80

Figura 44 Resultados de la simulación del archivo netlist 81

Figura 45 Diagrama de flujo de la funcionalidad de la aplicación..... 82

Figura 46 Archivos descriptivos del diagrama de un circuito eléctrico 83

Figura 47 Creación del archivo .bat..... 85

Figura 48 Acceso directo de la aplicación en el escritorio de Windows 85

Figura 49 Archivos descriptivos del diagrama de un circuito eléctrico 89

Figura 50 Detección de componentes del conjunto de prueba de la base de datos 90

Figura 51 Detección de componentes al segundo grupo de pruebas 91

Figura 52 Detección de textos con herramientas OCR..... 92

Figura 53 Vista general de la aplicación de reconocimiento inteligente de circuitos eléctricos. 94

Figura 54 Selección del diagrama de circuito eléctrico..... 95

Figura 55 Diagrama del circuito eléctrico cargado en la interfaz..... 95

Figura 56 Resultados del procesamiento del diagrama de circuito eléctrico 96

Figura 57 Diagrama del circuito eléctrico de prueba implementado en dos softwares de simulación 97

Figura 58 Comparación de netlist generada por softwares de simulación..... 98

Resumen

En la actualidad, existe un interés significativo en la digitalización de diagramas de ingeniería (circuitos eléctricos) debido a su complejidad y beneficios prácticos, además, existen pocos softwares que permiten realizar el reconocimiento de este tipo de diagramas. Debido a esto, ayudaría mucho tener una herramienta que identifique los elementos de un circuito, sus interconexiones y genere un netlist que permita la simulación. En consecuencia, se creó una aplicación basada en redes neuronales artificiales para el reconocimiento de los elementos del diagrama de un circuito eléctrico básico y sus interconexiones. Esta aplicación usa un modelo de detección de objetos con alta eficiencia (YOLOv11). La detección de nodos y conexiones se realizó utilizando técnicas clásicas del procesamiento de imágenes para el preprocesamiento, además del principio de componentes conexos. Se utilizó herramientas de reconocimiento óptico de caracteres (OCR) para hacer la identificación de textos mismos que se emplean para el desarrollo de archivos descriptivos, uno con detalles del circuito y otro con la lista de conexiones (netlist) que permite realizar una simulación automática en simuladores que se ejecutan bajo el motor de Spice. La aplicación se puso a prueba con varios circuitos compuestos por componentes activos como fuentes de voltaje y corriente, y componentes pasivos como resistencias, capacitores e inductores, en configuraciones de serie, paralelo y mixto. Además, la aplicación permite simular el netlist generado, presentando los resultados de la simulación de punto de operación, donde se presenta valores de voltajes de nodos y corrientes de cada elemento eléctrico que conforma el circuito. La aplicación presenta una alta eficiencia al detectar los elementos eléctricos de un circuito ya que logra una precisión promedio media (mAP50) del 95.7%. Para medir la eficacia de la aplicación, se comparó los resultados obtenidos, con los generados en otros softwares más desarrollados como PSpice o LTspice.

Palabras clave: Redes Neuronales Artificiales (RNA), Modelo de detección de objetos, Reconocimiento óptico de caracteres (OCR), Lista de conexiones (Netlist), Spice.

Abstract

Currently, there is significant interest in the digitization of engineering diagrams (electrical circuits) due to their complexity and practical benefits. Furthermore, there is little software available that allows for the recognition of these types of diagrams. Because of this, it would be very helpful to have a tool that identifies circuit elements and their interconnections and generates a netlist that allows for simulation. Consequently, an application based on artificial neural networks was created for the recognition of the elements of a basic electrical circuit diagram and their interconnections. This application uses a highly efficient object detection model (YOLOv11). Node and connection detection was performed using classic image processing techniques for preprocessing, in addition to the principle of connected components. Optical character recognition (OCR) tools were used to identify the texts, which are used to develop descriptive files: one with circuit details and another with the list of connections (netlist), which allows for automatic simulation in simulators running on the Spice engine. The application was tested with several circuits composed of active components such as voltage and current sources, and passive components such as resistors, capacitors, and inductors, in series, parallel, and mixed configurations. Additionally, the application allows the generated netlist to be simulated, presenting the results of the operating point simulation, which shows node voltages and currents for each electrical element that makes up the circuit. The application is highly efficient in detecting the electrical elements of a circuit, achieving a mean average accuracy (mAP50) of 95.7%. To measure the application's effectiveness, the results obtained were compared with those generated in other more advanced software programs such as PSpice or LTspice.

Keywords: Artificial Neural Networks (ANN), Object Detection Model, Optical Character Recognition (OCR), Netlist, Spice.

Capítulo I

Introducción

Antecedentes

En la actualidad, existe un interés importante en la digitalización de diagramas de ingeniería debido a su complejidad y beneficios prácticos (Jamieson et al., 2024), permitiendo a estudiantes de ingeniería interpretar diagramas de circuitos eléctricos correctamente, facilitando la comprensión de esquemas eléctricos, pudiendo realizar verificaciones automáticas de conexiones y elementos. Además, de poder generar archivos compatibles con simuladores que beneficien el aprendizaje de los circuitos eléctricos.

En los últimos años ha incrementado el interés del uso de inteligencia artificial para la detección de diagramas. Esto implica la creación de métodos de aprendizaje profundo para extraer todo el contenido importante de los diagramas eléctricos, como son los símbolos, líneas y texto, con el objetivo de analizar e identificar sus conexiones entre componentes (Bhanbhro et al., 2023).

Los investigadores han ideado diversos enfoques de aprendizaje profundo para la detección de elementos de diagramas eléctricos, basándose principalmente en detectores de objetos, como es la serie You Only Look One (YOLO) (Li et al., 2022) y Regions with Convolutional Neural Networks (R-CNN) (Ren et al., 2016). No obstante, un factor clave al utilizar técnicas de aprendizaje profundo para identificar símbolos es la necesidad de contar con un número considerable de datos de entrenamiento por cada tipo de símbolo.

En la actualidad, existen varias investigaciones relacionadas con el reconocimiento de símbolos de diagramas eléctricos, entre los más destacados se encuentran los siguientes. En (Rezaeva & Semendyaev, 2021) los autores desarrollan e implementan una arquitectura de red neuronal convolucional para el reconocimiento de objetos en el esquema de una red eléctrica aplicando un método de ventana flotante para poder determinar el lugar de cada elemento en la imagen. En el trabajo de (Bhanbhro et al., 2023) diseñan un sistema de reconocimiento de

símbolos para diagramas unifilares desarrollado mediante técnicas de aprendizaje profundo utilizando un conjunto de datos para evaluar la calidad general de un sistema visual. Sin embargo, estos estudios únicamente se enfocan en el reconocimiento de componentes eléctricos mas no en sus interconexiones.

En cuanto al reconocimiento de elementos de un diagrama de circuito eléctrico y sus conexiones, los proyectos que resaltan son los siguientes. En (Rachala & Panicker, 2022) los autores desarrollan un algoritmos para la detección de componentes de un circuito dibujado a mano empleando algoritmos de detección de objetos como YOLOv5, reconocimiento de nodos y terminales y una generación esquemática del circuito dibujado a mano, sin generar archivos compatibles para simulación. En (Bohara & Krishnamoorthy, 2024) se presenta un estudio basado en aprendizaje profundo para identificar esquemas dibujados a mano de circuitos de convertidores de potencia y realizar simulaciones automatizadas, haciendo uso de modelos de detección de objetos basados en visión artificial como YOLOv8, además, mediante el uso de técnicas clásicas de procesamiento de imágenes y enfoques de aprendizaje profundo logra identificar las conexiones entre elementos, sin embargo su enfoque es centrado únicamente en diseños de convertidores de potencia.

Dentro del ámbito local, ya existe un estudio similar en la ESPE. El trabajo de (Muñoz Chuquimarca, 2021), elabora una aplicación que detecta los elementos del diagrama de un circuito eléctrico básico y sus interconexiones, para generar archivos de texto que contienen una descripción de cómo están interconectados lo componentes del circuito, sin embargo, carece de la posibilidad de crear una Netlist compatible con simuladores.

Con lo expuesto se destaca la importancia de tener una herramienta de apoyo que ayude y facilite el uso de diagramas de circuitos eléctricos, permitiendo a los usuarios identificar los componentes de un esquema eléctrico y su conexiones.

Justificación

El presente proyecto de integración curricular pretende desarrollar una herramienta cuyos resultados permitan crear futuras aplicaciones para reducir las barreras de acceso a la información técnica mediante el uso de tecnologías avanzadas de procesamiento digital de señales, imágenes e inteligencia artificial. Orientado a estudiantes de carreras técnicas que necesiten accesibilidad a diagramas de circuitos eléctricos, fundamentales para el estudio y análisis en las áreas de eléctrica y electrónica.

Por esta razón, se propone desarrollar una aplicación que sirva como herramienta de apoyo para identificar componentes y sus interconexiones del diagramas de circuitos eléctricos. Con esta aplicación, los usuarios podrán conocer información detallada de cual es la disposición y organización de los elementos en un circuito eléctrico.

Por otro lado, el desarrollo de esta aplicación permitirá la integración con softwares de simulación, generando archivos compatibles con simuladores eléctricos para probar el funcionamiento de los diagramas de circuitos eléctricos y obtener valores de corrientes o voltajes de los elementos que constituyen el circuito.

Adicional, los resultados obtenidos pueden servir como base para futuras aplicaciones orientadas a la inclusión digital, dando facilidad de acceso a conocimientos técnicos a un público más amplio. Además, crea la posibilidad de integrar tecnologías de accesibilidad para personas que presenten discapacidad visual o dificultad en la aprendizaje de estos conceptos.

Finalmente, este trabajo de integración curricular fomenta el uso de tecnologías que pueden ayudar en la educación técnica en áreas de eléctrica y electrónica, ya que se alinea con el ODS número 4 el cual promueve “garantizar una educación inclusiva, equitativa y de calidad, y promover oportunidades de aprendizaje durante toda la vida para todos”, el desarrollo de esta aplicación se considera una herramienta innovadora que puede apoyar tanto a estudiantes como docentes en la formación técnica fomentando un aprendizaje accesible e interactivo.

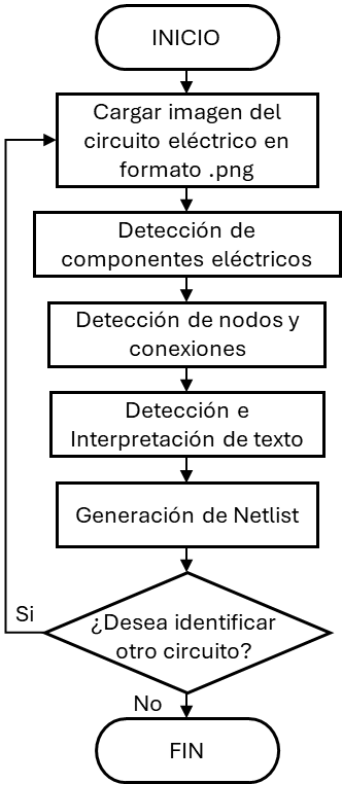
Alcance del Proyecto

El presente proyecto de integración curricular pretende desarrollar una herramienta haciendo uso del lenguaje de programación Python y OpenCV, que permita procesar el diagrama de un circuito eléctrico básico, y mediante el uso de inteligencia artificial identificar los componentes eléctricos que lo conforman (fuentes de corriente, fuentes de voltaje, resistencias, capacitores y bobinas) y sus interconexiones. La salida consistirá en dos archivos: uno que detalla los elementos del circuito, sus valores y conexiones, y otro en formato Netlist, el cual podrá ser utilizado en programas de simulación para analizar el funcionamiento del circuito.

Para lograr este objetivo, se diseñará una interfaz gráfica que permitirá al usuario ingresar el diagrama de un circuito eléctrico el cual va a ser procesado como se muestra en la Figura 1, y posterior a esto entregará a la salida dos archivos donde uno detallará la información de los elementos del circuito (.txt) y un segundo archivo en formato Netlist (.cir).

Figura 1

Procesamiento del diagrama de un circuito eléctrico



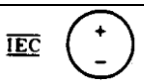
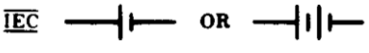
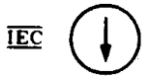
El proceso de reconocimiento de diagramas de circuitos eléctricos comprende varias etapas importantes como se observa en la Figura 1. En primera instancia, se debe ingresar la imagen del circuito a ser procesado, para nuestro caso estar en formato de extensión .png y esta deberá tener una resolución mínima de 640 pixeles de ancho. Además, de que el reconocimiento de elementos se hará únicamente de imágenes tomadas de documentos en formato pdf con el fin de mejorar la precisión del procesamiento y la detección de los elementos.

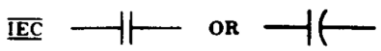
Posterior a esto, se hará la detección de componentes eléctricos donde se evalúa una gama de modelos de detección de objetos, para detectar varios componentes en las imágenes. Para esto se necesita trabajar con una base de datos, debido a que el modelo de detección debe ser entrenado para identificar los elementos de un circuito eléctrico. Una vez entrenado el modelo de detección se procederá a realizar las pruebas debidas para confirmar si es apto para identificar los elementos de un circuito.

Es importante mencionar que el modelo de detección será entrenado con una base de datos donde los diagramas de circuitos eléctricos sigan el estándar "*IEEE Standard 315-1975*" (Institute of Electrical and Electronics Engineers, 1975) y se considerará únicamente los componentes y simbología presentados en la Tabla 1.

Tabla 1

Elementos de un circuito eléctrico básico

N°	Elemento	Simbología
1	Fuente de voltaje	
2	Batería	
3	Fuente de corriente	

N°	Elemento	Simbología
4	Resistencia	
5	Capacitor	
6	Inductor	

Nota. Simbología tomada de (Institute of Electrical and Electronics Engineers, 1975)

Después se desarrollará la detección de nodos y conexiones, en esta etapa se identifica los terminales de cado componentes y se los asocia con los nodos detectados en el diagrama de circuito eléctrico.

Luego, se realizará la detección e interpretación de texto ya que los textos son la información vital que se muestra en un diagrama de circuito eléctrico a través de la cual los usuarios pueden transcribir los valores de los componentes como voltajes, corrientes, resistencias, etc. Por lo tanto, la identificación, interpretación y acoplamiento de los textos en los diagramas a los componentes respectivos son pasos cruciales en la generación automática de la Netlist. Para la detección e interpretación de texto de la escena, están disponibles varias herramientas de reconocimiento óptico de caracteres (OCR) de código abierto compatibles con Python.

Posteriormente, se genera un *netlist (lista de conexiones)*, este es un archivo con extensión `.cir` que describe textualmente a los componentes eléctricos de un circuito, conexiones entre estos y sus valores. La *netlist* se genera a partir de la información general del sistema, contemplando tipos de componentes, terminales y nodos involucrados. Generalmente, una *netlist* está organizada por tres columnas: 1) nombre del componente, 2) información sobre la conexión, y 3) valor del componente. Esta lista de conexiones permite visualizar de forma clara la configuración del circuito, así como su funcionamiento.

Finalmente, se evaluará la aplicación comparando la Netlist entregada con una generada con por un programa de simulación y se hará una comparación visual de las mismas. O también incorporando la Netlist generado por la aplicación a otro programa de simulación de circuitos y verificando sus resultados en simulación.

Objetivos

Objetivo General

Crear una aplicación que permita el reconocimiento de los elementos del diagrama de un circuito eléctrico básico y sus interconexiones utilizando inteligencia artificial.

Objetivos Específicos

- Seleccionar algoritmos de inteligencia artificial para reconocer componentes de un diagrama eléctrico.
- Seleccionar una base de datos para entrenamiento.
- Implementar el algoritmo de detección de los elementos de un diagrama eléctrico.
- Implementar el algoritmo de reconocimiento de conexiones.
- Generar archivos descriptivos y en formato Netlist.

Capítulo II

Marco Teórico

Procesamiento Digital de Imágenes

A partir de la década de 1960 la tecnología de procesamiento digital de señal ha crecido de manera considerable. En un principio el procesamiento de imágenes utilizaba circuitos analógicos los cuales presentaban características como lento procesamiento y baja precisión. En la década de 1970, el procesamiento digital de imágenes surge gradualmente debido a las mejoras de rendimiento informático. Para la década de 1980, con el surgimiento de nuevos algoritmos como el operador de detección de bordes Canny, el algoritmo de binarización de imágenes Otsu y la codificación de compresión por la transformada wavelet, entre otros, aportaron a la mejora continua de la teoría de procesamiento digital de imágenes.

Con el desarrollo de la tecnología multimedia y el Internet, en la década de 1990 el procesamiento digital de imágenes entra en una nueva etapa. Debido a la enorme cantidad de datos de imágenes, se sientan las bases para el desarrollo de aplicaciones de Deep Learning (Aprendizaje Profundo) y otras técnicas de inteligencia artificial en el capo de la imagen.

Gracias al buen rendimiento del Deep Learning, se ha extendido considerablemente el alcance de las aplicaciones de procesamiento de imágenes como es en el uso de reconocimiento óptico de caracteres (OCR), reconocimiento de rostro y detección de objetos, todas estas con un amplio panorama de aplicación. En la actualidad, el análisis de imágenes se ha integrado de forma significativa en diversos sectores como la industria, la medicina, la interacción entre humanos y computadoras, la seguridad y la conducción autónoma, contribuyendo notablemente al progreso económico y al bienestar social (Liu C. , 2024).

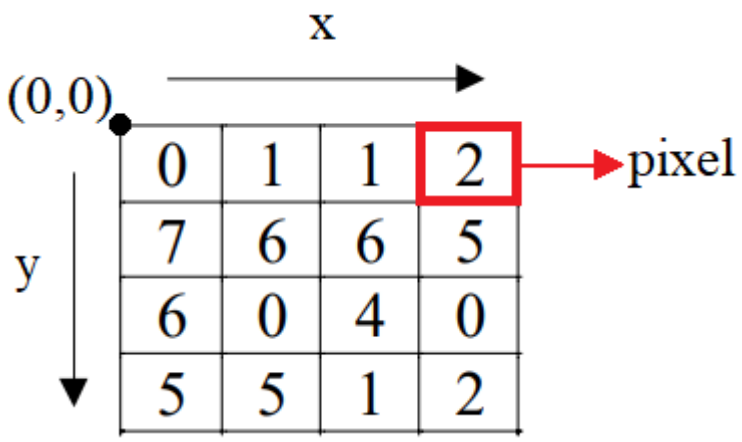
Descripción General

Una imagen puede conceptualizarse como una función matemática bidimensional f(x,y), donde las variables (x, y) representan coordenadas espaciales sobre un plano, y el valor de f en dichas coordenadas indica la intensidad o nivel de gris correspondiente como se muestra en

la Figura 2. Cuando tanto las coordenadas espaciales como los niveles de intensidad toman valores discretos y finitos, se define una imagen digital.

Figura 2

Imagen de 16 pixeles



Nota. Tomado de (Elizondo & Maestre, 2005)

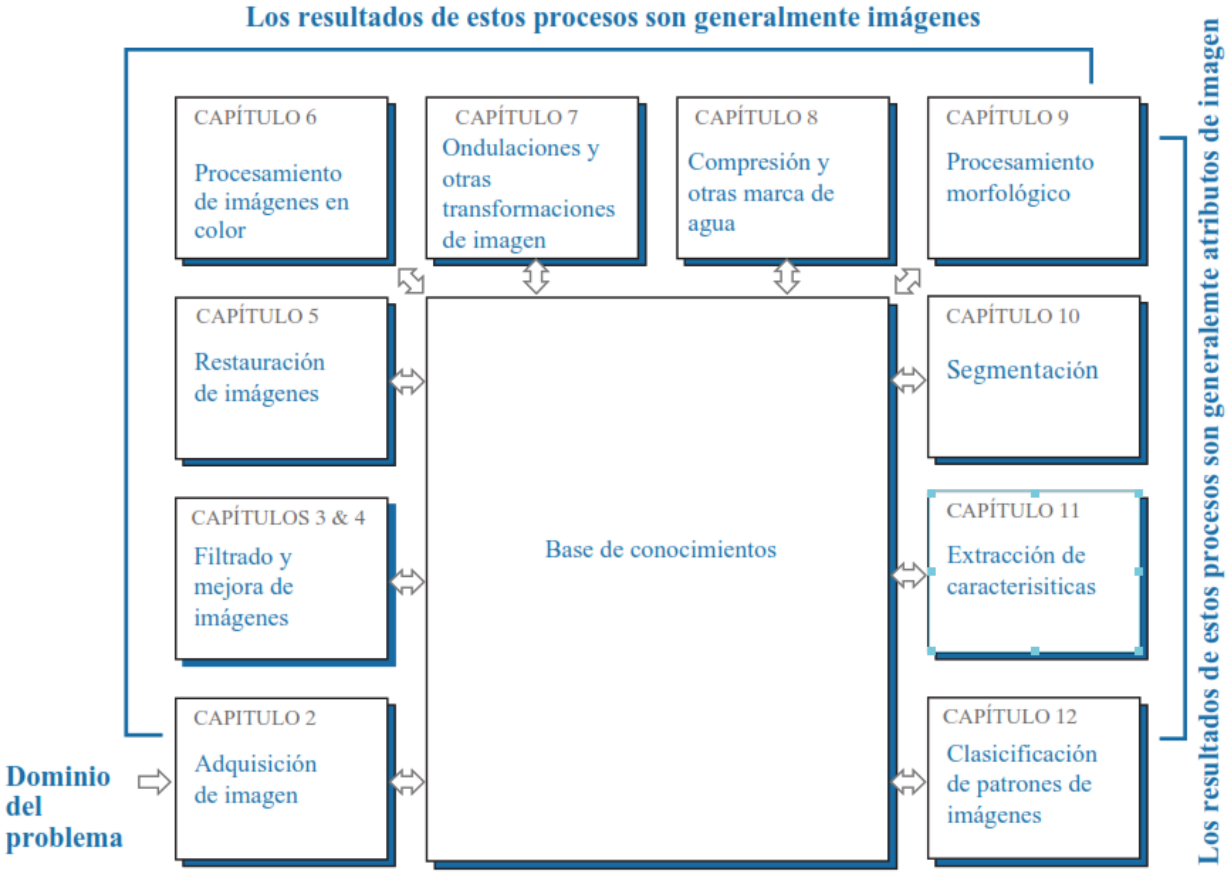
El procesamiento digital de imágenes engloba un conjunto de técnicas computacionales enfocadas en el análisis y manipulación de imágenes mediante sistemas digitales. Una imagen digital está compuesta por un número finito de unidades básicas, cada una con una ubicación y valor especifico. Estas unidades reciben el nombre de elementos de imagen, pels o píxeles, siendo este último el término más utilizado (Gonzales & Woods, 2018).

Pasos para el procesamiento digital de imágenes

En el procesamiento digital de imágenes, es fundamental diferenciar dos tipos de procesos: el primero que consiste en recibir una imagen y entregar otra como resultado, y el segundo que parte de una imagen a la cual se extrae información específica. Estos procesos generales se pueden visualizar en la Figura 3, donde se estructura distintas metodologías aplicables a imágenes, sin que todas sean necesariamente aplicables o en un orden especifico.

Figura 3

Pasos fundamentales en el procesamiento digital de imágenes



Nota. Tomado de (Gonzales & Woods, 2018)

En un sistema de procesamiento de imágenes uno de los primeros pasos es la adquisición. Esto conlleva a únicamente recibir una imagen ya existente o aplicar ciertas acciones de preprocesamiento como escalarla. Una vez adquirida la imagen, puede ser necesario arreglarla visualmente para que sea más eficiente en una tarea específica. Esta mejora, se la conoce como realce de imagen, lo que la adapta dependiendo el tipo de aplicación. Por ejemplo, no se aplican las mismas técnicas para imágenes médicas que para imágenes satelitales. A diferencia de otras áreas, no existe una teoría general para el realce, ya que su efectividad depende de la percepción visual humana y del propósito al que está destinado.

Por esta razón, generalmente se emplean ejemplos de mejora de imagen en las primeras etapas del aprendizaje, ya que son visualmente claros, fáciles de interpretar y permiten introducir conceptos clave. Sin embargo, muchas de estas técnicas también se aplican en áreas más amplias dentro del procesamiento digital.

Con el objetivo de buscar una mejora en la calidad visual de imágenes, un campo relacionado es la restauración de imágenes, este usa un enfoque más objetivo y matemático, debido a que repara defectos conocidos mediante modelos probabilísticos. Por otro lado, el realce depende más de la percepción humana.

El procesamiento de imágenes a color ha tenido mayor relevancia debido al crecimiento del uso de imágenes digitales, debido al crecimiento del internet. Para esto, se estudian modelos de color y cómo manipularlos digitalmente, lo que permite extraer información relevante en función del color.

Por otro lado, las transformadas de *Fourier* como las *Wavelets*, son utilizadas para representar diferentes niveles de detalle en las imágenes, resultando útil al momento de interpretar datos. Por este motivo, es importante la compresión de imágenes puesto a que ahorrar espacio de almacenamiento o en transmisión de datos ayuda a reducir el ancho de banda. Aunque hoy en día los dispositivos permiten almacenar grandes cantidades de datos, sigue siendo un reto la velocidad de transmisión.

Las técnicas de procesamiento morfológico, son utilizadas cuando se desea describir la forma u otras propiedad específica de los elementos que conforman una imagen. Esta técnica corresponde a los métodos que no generan imágenes como resultado, sino que entregan características extraídas de ellas.

La segmentación, por ejemplo, divide la imagen en regiones. Una segmentación precisa facilita enormemente tareas posteriores como la clasificación automática. A partir de esta segmentación, se puede realizar la extracción de características, lo que implica detectar ciertos elementos dentro de una imagen y describirlos de forma cuantitativa. Estas descripciones

deben ser lo más resistentes posible a cambios como rotaciones, escalado o variaciones de luz.

Finalmente, una vez que se tienen las características, se pueden clasificar objetos dentro de la imagen mediante métodos de reconocimiento de patrones. Esta clasificación puede realizarse con técnicas tradicionales o mediante modelos más avanzados, como las redes neuronales profundas, especialmente las convolucionales, que son de las más utilizadas en el procesamiento de imágenes.

Hay que considerar que no todas las aplicaciones deben pasar por todas las etapas mostradas en la Figura 3. Al realizar tareas simples, como mejorar visualmente una imagen para que sea interpretada por una persona, no es necesario cumplir con todas las etapas. Pero, mientras más compleja sea una aplicación, más aumenta el número de procesos involucrados.

Morfología de Imágenes

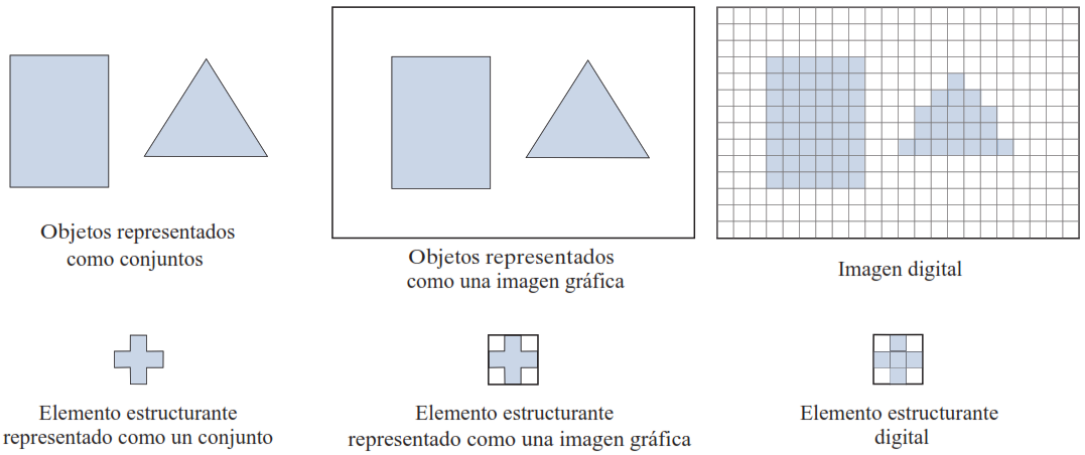
La morfología de imágenes es un procesamiento digital que emplea principios de álgebra de conjuntos, y es utilizada para interpretar la forma y estructura de objetos dentro de una imagen. Este tipo de procesamiento de imágenes se utiliza tanto para tareas de pre y posprocesamiento, como eliminación de ruido, filtrado, afinamiento de formas, en la extracción de atributos como contornos o componentes conexos.

Conceptos básicos

En las imágenes binarias, los elementos estructurantes (*structuring elements*) se utilizan como "máscaras" para examinar o modificar los objetos, que son conjuntos de pixeles en primer plano (Gonzales & Woods, 2018). Estos elementos pueden abarcar areas indiferentes (designadas como "x") que no se tienen en cuenta durante los procedimientos. La Figura 4 muestra el proceso por el cual los objetos y elementos estructurantes se transforman en imágenes digitales que pueden ser manipuladas.

Figura 4

Digitalización de objetos y elementos estructurantes



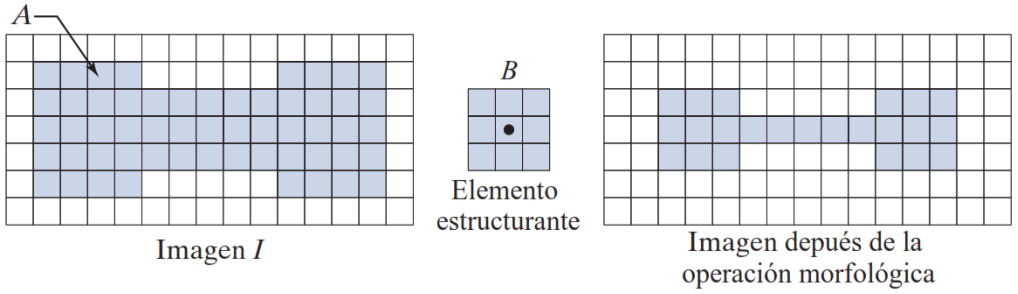
Nota. En la fila superior. Izquierda se muestra objetos representados como conjuntos gráficos. Centro: Objetos incrustados en un fondo para formar una imagen gráfica. Derecha: Objeto y fondo digitalizados para formar una imagen digital. Mientras que en segunda fila. El ejemplo de un elemento estructurante representado como un conjunto, una imagen gráfica y, finalmente, como un elemento estructurante digital. Tomado de (Gonzales & Woods, 2018)

Operaciones fundamentales

Erosión. Esta operación reduce el tamaño de los objetos de una imagen, eliminando píxeles en sus bordes. Se define por el encaje total del elemento estructurante dentro del objeto. La Figura 5 muestra cómo la erosión afecta la forma de un objeto.

Figura 5

Ejemplo del proceso de erosión



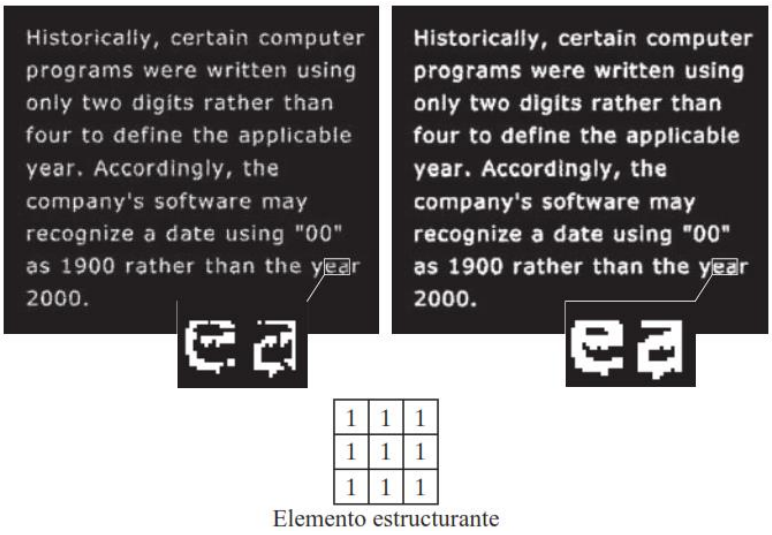
Nota. Tomado de (Gonzales & Woods, 2018)

En la primera ilustración de la Figura 5 se muestra una imagen binaria que contiene un objeto (A), en la siguiente un elemento estructurante (B) y por último una imagen resultante de una operación morfológica (erosión).

Dilatación. Contrario a la erosión, esta operación expande los objetos, añadiendo píxeles alrededor de su perímetro. Depende de la forma y tamaño del elemento estructurante. Un ejemplo de cómo se utiliza la dilatación para recuperar caracteres fracturados en un texto digitalizado se muestra en la Figura 6.

Figura 6

Ejemplo del proceso de dilatación



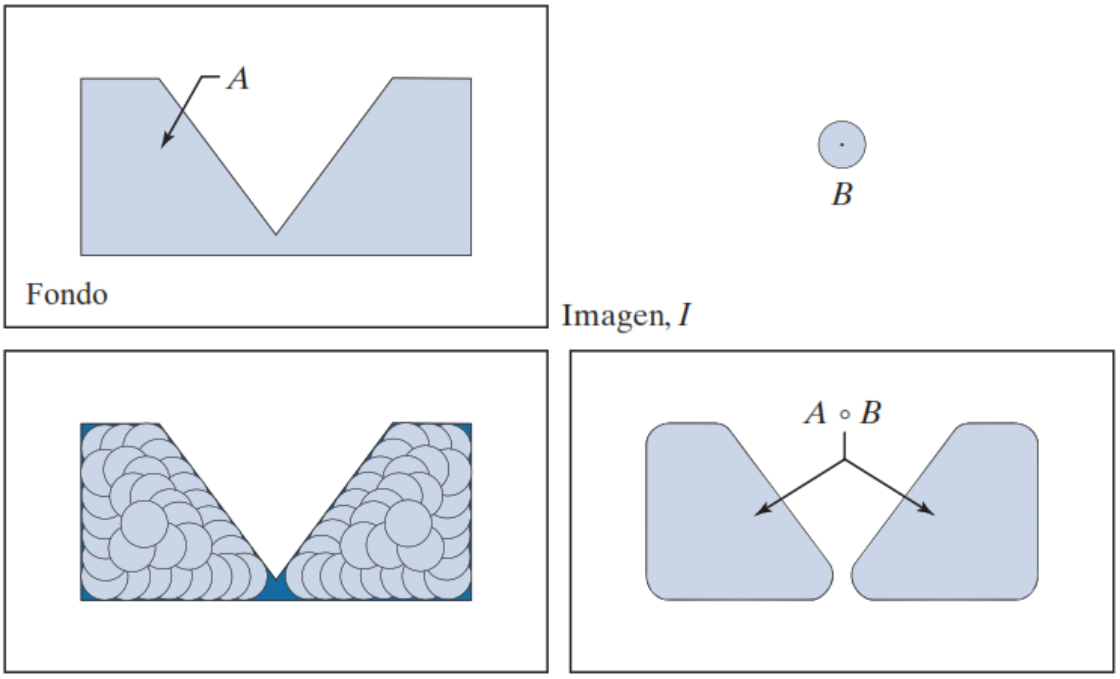
Nota. (a) Texto de baja resolución con caracteres rotos. (b) Elemento estructurante. (c) Dilatación de (a) por (b) uniendo los segmentos rotos. Tomado de (Gonzales & Woods, 2018)

Operaciones combinadas

Apertura. Es una erosión seguida de una dilatación. Su objetivo es eliminar detalles pequeños o estrechos sin alterar significativamente la forma general. La Figura 7 representa visualmente el proceso y su efecto.

Figura 7

Ejemplo del proceso de apertura

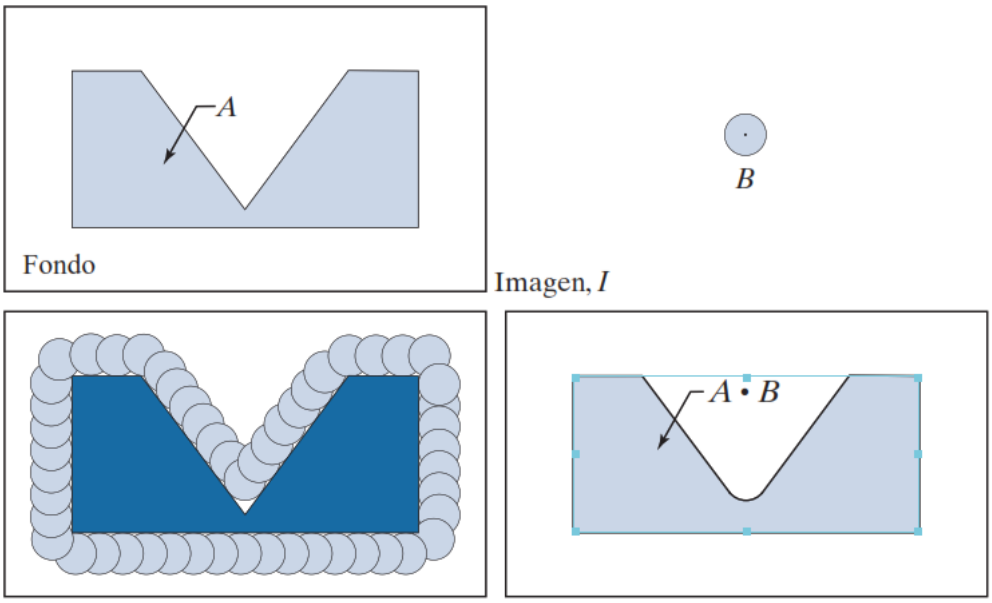


Nota. (a) Imagen compuesta por el conjunto A y el fondo. (b) Elemento estructurante B. (c) Traslaciones de B mientras está contenido en A. (d) Apertura de A por B. Tomado de (Gonzales & Woods, 2018)

Cierre. Inverso a la apertura, consiste en una dilatación seguida de una erosión. Se usa para rellenar huecos o unificar componentes cercanos. En la Figura 8 se muestra cómo se suaviza contornos y se elimina pequeños espacios vacíos.

Figura 8

Ejemplo del proceso de cierre



Nota. (a) Imagen compuesta por el conjunto A y el fondo. (b) Elemento estructurante B. (c) Traslaciones de B tales que B no se superpone a ninguna parte de A. (d) Cierre de A por B. Tomado de (Gonzales & Woods, 2018)

Algoritmos morfológicos útiles

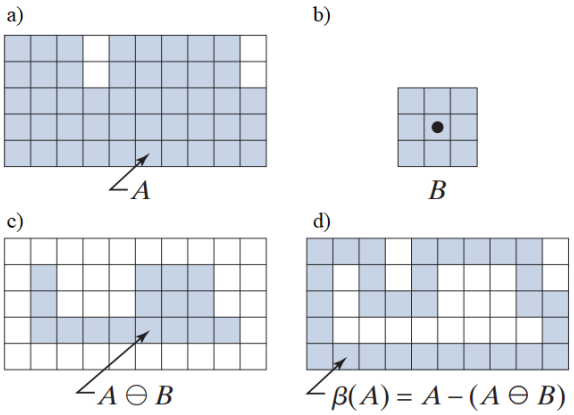
Es posible generar algoritmos que permiten resolver tareas en el procesamiento de imágenes haciendo uso de operaciones morfológicas básicas. Estos algoritmos son fundamentales para examinar la forma y organización de los objetos, y se utilizan en el hallazgo de componentes conexos, el relleno de vacíos, la creación de envolventes convexas, el análisis de bordes y la depuración de formas. Su empleo es fundamental en aplicaciones como el análisis de imágenes médicas, la detección de patrones y otras.

A continuación, se presentan estos algoritmos fundamentados y con ejemplos ilustrativos que enseñan su utilidad práctica en distintos contextos.

Extracción de bordes. Se obtiene restando una erosión del conjunto original. La Figura 9 muestra cómo esto resalta el contorno del objeto.

Figura 9

Ejemplo del algoritmo de extracción de bordes

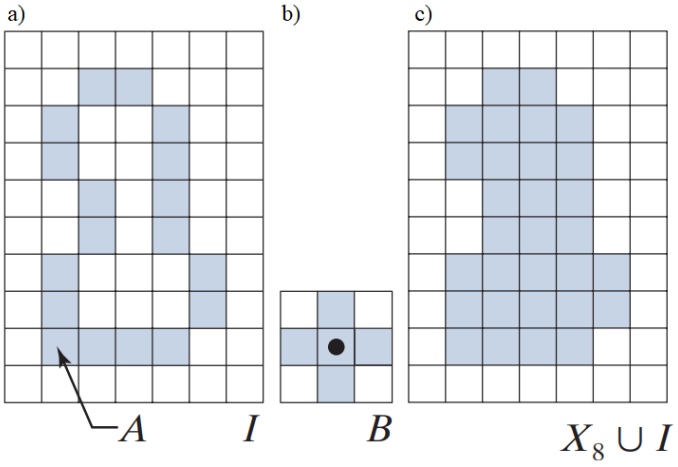


Nota. (a) Conjunto A, de píxeles de primer plano. (b) Elemento estructurante. (c) A erosionado por B. (d) Límite de A. Tomado (Gonzales & Woods, 2018)

Relleno de huecos. Usa dilatación condicional para completar zonas interiores rodeadas por píxeles del objeto. En la Figura 10, se detalla se observa el proceso.

Figura 10

Ejemplo del algoritmo de relleno de huecos

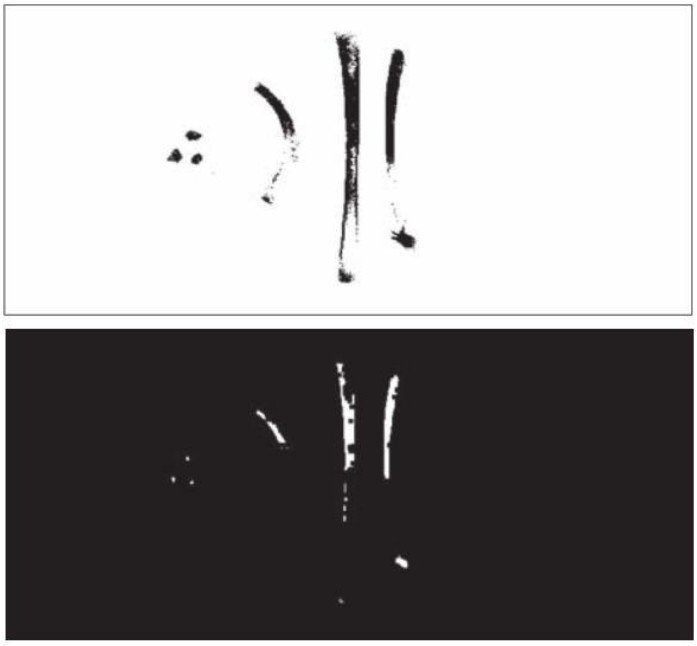


Nota. (a) Conjunto A contenido en la imagen I. (b) Elemento estructurante B. (c) Resultado final. Tomado de (Gonzales & Woods, 2018)

Componentes conexos. Se extraen regiones formadas por pixeles continuos, que generalmente comparten igual valor de intensidad. En la Figura 11 se muestra su uso para detectar objetos extraños en alimentos.

Figura 11

Ejemplo del algoritmo de componentes conexos



Componente conexo	No. de pixeles en el comp. conexo
01	11
02	9
03	9
04	39
05	133
06	1
07	1
08	743
09	7
10	11
11	11
12	9
13	9
14	674
15	85

Nota. Imagen de rayos X de un filete de pollo con fragmentos de hueso. Y la misma imagen erosionada con un error estándar de 5x5 elementos estructurantes de unos. Además, se muestra el número de píxeles en los componentes conectados. Tomado de (Gonzales & Woods, 2018)

Thinning (adelgazamiento). El adelgazamiento es una técnica morfológica que reduce paulatinamente el grosor de los objetos en una imagen binaria, conservando su estructura general. Se aplica mediante una serie de elementos estructurantes rotados, que detectan y borran píxeles en los bordes sin romper la conectividad. Este proceso se repite hasta que ya no ocurren cambios en la imagen.

Thickening (engrosamiento). El engrosamiento, al contrario, trata de expandir los objetos agregando píxeles en sus bordes. Es útil para fortalecer estructuras delgadas, unir

fragmentos o rellenar zonas erosionadas. Se considera el dual del adelgazamiento, ya que sigue una lógica similar, pero en sentido opuesto: en lugar de borrar bordes, los hace más grandes utilizando también transformaciones basadas en coincidencias con elementos estructurantes. Aunque no se detalla gráficamente tanto como el adelgazamiento, su función es esencial en tareas de restauración o realce de formas en imágenes binarias.

Redes Neuronales

Los estudios en matemática, computación y neurociencia, han permitido un desarrollo evolutivo en el avance de las redes neuronales artificiales (RNA). Su principio conceptual se remonta a las décadas de 1940, cuando en 1943 McCulloch y Pitts propusieron un modelo matemático que representaba cómo operan las neuronas biológicas (Rivera, 2005). Posteriormente, en la década de 1950, Frank Rosenblatt desarrolló el perceptrón, una de las primeras creaciones prácticas de una neurona artificial, que marcó el inicio de la era actual de las RNA (Papadopoulos et al., 2021).

El hallazgo del algoritmo de retropropagación del error, creado por Rumelhart, Hinton y Williams en la década de 1980, posibilitó el entrenamiento efectivo de redes multicapa, lo que permitió sobrepasar una de las limitaciones más importantes del perceptrón simple. Esta técnica marcó un hito en el entrenamiento de redes neuronales profundas (*Deep Learning*).

Las RNA se han transformado en herramientas indispensables en áreas tan diversas como la visión por computadora, el procesamiento del lenguaje natural o la medicina gracias a la disponibilidad de grandes cantidades de datos, al desarrollo de bibliotecas de software especializadas y a la capacidad de procesamiento que ha crecido exponencialmente. (Pons et al., 2022).

Definición

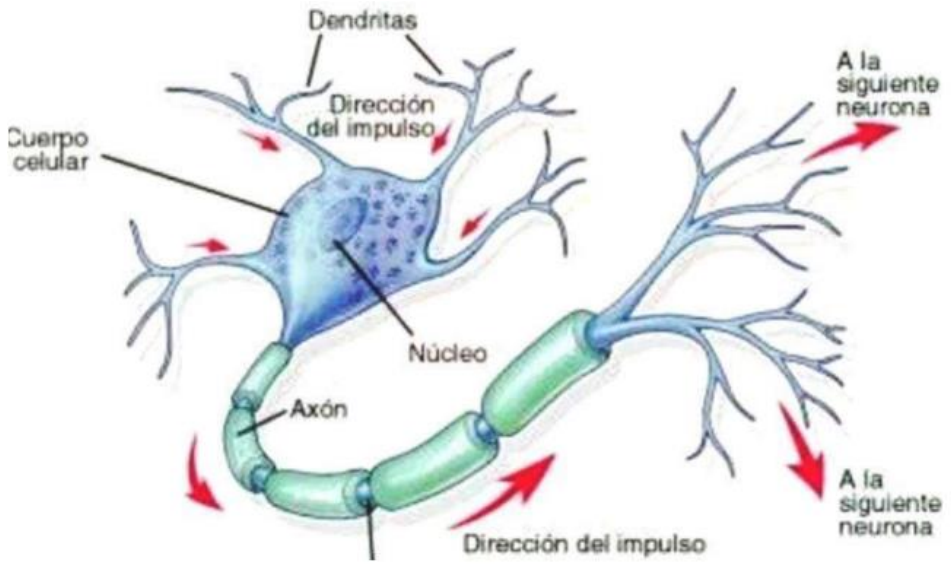
Una red neuronal artificial es un modelo computacional inspirado en la estructura del cerebro humano (Livingstone, 2011). Consiste en un conjunto de nodos (neuronas artificiales) organizados en capas que se tienen comunicación entre sí mediante conexiones ponderadas.

Su objetivo es aprender patrones a partir de datos, ajustando los pesos sinápticos durante el proceso de entrenamiento.

Modelo Biológico. Las neuronas biológicas Figura 12, son células especializadas que reciben señales eléctricas a través de sus dendritas, las procesan en el núcleo y transmiten la señal resultante mediante el axón hacia otras neuronas. La comunicación entre neuronas ocurre en las sinapsis, donde se regula la intensidad de la señal transmitida. Esta lógica ha sido imitada de forma simplificada en las redes neuronales artificiales (Pons et al., 2022).

Figura 12

Neurona biológica

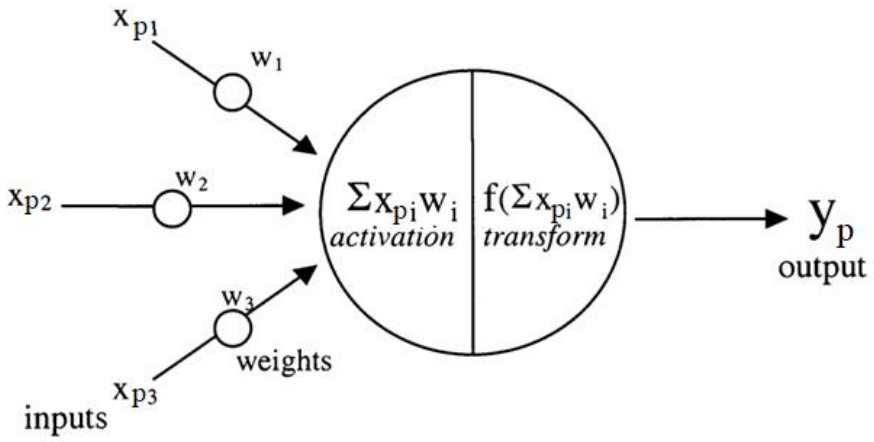


Nota. Tomado de (Pons et al., 2022).

Modelo de una neurona artificial. Una neurona artificial Figura 13, recibe varias entradas $(x_{p1}, x_{p2}, \dots, x_{pn})$, cada una asociada a un peso $(w_1, w_2, \dots, w_n)$. Realiza una suma ponderada de estas entradas, les aplica una función de activación y genera una salida (y_p) (Agatonovic-Kustrin & Beresford, 2000).

Figura 13

Modelo de una neurona artificial



Nota. Tomado de (Agatonovic-Kustrin & Beresford, 2000).

Elementos Básicos

Las redes neuronales están compuestas de varios elementos fundamentales, tanto a nivel estructural como practico. Estos elementos permiten a las redes ser capaces de representar funciones complejas y aprender de los datos, estos elementos son:

Neurona artificial. Es el bloque más básico del sistema debido a que cada neurona realiza un procesamiento sencillo, pero en conjunto forman una red con gran capacidad de cómputo y aprendizaje.

Pesos sinápticos. Indican la intensidad de conexión que existe entre neuronas. Estos valores pueden ser ajustados a medida que avanza el proceso de aprendizaje, además determina la influencia de una entrada sobre la salida de la neurona.

Rivera (2005) menciona que “el conocimiento aprendido por la red está codificado en estos pesos”, lo que hace que el ajuste de los pesos sea el objetivo principal del entrenamiento.

Funciones de activación. Convierten la salida del sumador en un formato apropiado para el siguiente nivel de la red. Algunas funciones habituales son:

- Sigmoide: Este tipo de función suaviza la salida de la neurona manteniendo su valor entre 0 y 1.

$$f(u) = \frac{1}{1 + e^{-u}}$$

- ReLU (Rectified Linear Unit): Es ampliamente utilizada en redes profundas por su eficiencia computacional.

$$f(u) = \max(0, u)$$

Capas. Las redes se estructuran en capas de neuronas, las cuales se clasifican en:

- Capa de entrada: recibe los datos del entorno.
- Capas ocultas: realizan el procesamiento intermedio. Pueden ser una o muchas.
- Capa de salida: entrega el resultado final de la red.

Abiodun et al. (2019) indica la cantidad de capas ocultas y neuronas afecta directamente la capacidad de la red para modelar relaciones complejas.

Aprendizaje y validación de la red neuronal

El aprendizaje de una red neuronal consiste en ajustar los pesos de sus conexiones internas para minimizar el error entre la salida estimada y la deseada. Se realiza mediante un algoritmo de optimización, como el descenso de gradiente, y habitualmente se emplea el método de retropropagación del error, como se explica en (Agatonovic-Kustrin & Beresford, 2000; Rivera, 2005).

Hiperparámetros fundamentales. Los hiperparámetros no se aprenden automáticamente; deben establecerse antes del entrenamiento y afectan profundamente el desempeño del modelo. Según Papadopoulos et al. (2021), entre los más comunes son los que se presentan a continuación.

- *Learning Rate* (Tasa de aprendizaje): este hiperparámetro sirve para determinar el tamaño del paso que se tiene en la actualización de los pesos. El tener un valor alto de *learning rate* puede causar inestabilidad y si es muy bajo, lentitud.

- *Batch Size:* Número de muestras procesadas antes de actualizar los pesos, con tamaños pequeños generan un entrenamiento más ruidoso, pero permiten mayor generalización.
- Épocas: Número de veces que la red observa a todo el conjunto de datos durante el entrenamiento.
- Número de Capas: Establece la profundidad de la red. A mayor cantidad, mayor capacidad para aprender patrones complejos (redes profundas o D*eep Learning*).
- Número de Neuronas por Capa: Define la capacidad de representación de cada capa.
- Número de *Workers*: Procesos paralelos utilizados en el entrenamiento (afecta la velocidad).

Proceso de entrenamiento. Al momento de realizar el entrenamiento de la red, esta procesa un conjunto de datos etiquetados, mientras realiza ajustes a sus pesos en iteración (época) dependiendo del error cometido. Este proceso incluye:

1. Establecer pesos iniciales de forma aleatoria.
2. Propagación hacia adelante (*forward pass*): se calcula la salida de la red.
3. Cálculo del error entre la salida real y la esperada.
4. Retropropagación del error (*backward pass*): se actualizan los pesos en función del gradiente del error.
5. Repetición del proceso en múltiples épocas hasta que el error sea aceptablemente bajo.

Validación y evaluación del desempeño. Una vez entrenada la red, es crucial medir su rendimiento sobre datos de prueba usando diversas métricas como las mostradas en (Matich, 2001). Mismas que se detallan a continuación.

- Exactitud (*Accuracy*): Proporción de predicciones correctas sobre el total de muestras.
- Precisión (*Precision*): De todas las predicciones positivas, cuántas fueron correctas. Se usa en problemas donde el falso positivo es crítico.
- Sensibilidad (*Recall*): De todos los casos positivos reales, cuántos fueron detectados correctamente.
- Confianza (*Confidence*): Este valor es la probabilidad que estima la red sobre la cuenta certeza tiene su predicción.
- Matriz de confusión: Es una representación gráfica de las clases verdaderas frente a las predichas, en la cual se presenta los aciertos y errores de cada clase.

Topologías de redes neuronales

La topología de una red neuronal alude a su estructura interna, o sea, la forma en que las neuronas están dispuestas y cómo se enlazan. El tipo de problema a resolver (regresión, clasificación, detección de patrones, etc.) y la naturaleza y cantidad de datos son determinantes en la selección de una topología apropiada.

Varios estudios profundizan en las topologías mostradas a continuación (Agatonovic-Kustrin & Beresford, 2000; Rivera, 2005).

Perceptrón Multicapa (MLP). Una de las arquitecturas más básicas y utilizadas, también conocida como red feedforward. Está compuesta por:

- Una capa de entrada
- Una o más capas ocultas
- Una capa de salida

Las señales viajan en una sola dirección de entrada a salida, sin ciclos ni retroalimentación. Esta topología es muy usada en tareas como predicción de series temporales, diagnóstico médico y reconocimiento de voz.

Redes Neuronales Convolucionales (CNN). Diseñadas para procesar datos con estructura espacial, como imágenes. Se componen de capas convolucionales, pooling y totalmente conectadas.

Características clave:

- Capturan características locales mediante filtros
- Reducen la dimensionalidad con submuestreo (pooling)
- Mantienen invariancia a traslaciones

Abiodun et al. (2019) menciona que las CNN han superado el rendimiento humano en tareas de reconocimiento de objetos como por ejemplo “ImageNet”.

Redes Recurrentes (RNN). Este tipo de redes contienen conexiones las cuales dan paso a una realimentación entre neuronas, haciéndolas precisas para procesar secuencias y datos temporales como audio o texto.

Tipos importantes:

- RNN básicas
- LSTM (*Long Short-Term Memory*)
- GRU (*Gated Recurrent Units*)

Según Rivera (2005), las LSTM son ampliamente utilizadas en traducción automática, modelado de lenguaje y análisis de series temporales.

Redes Auto-asociativas (*Autoencoders)*. Este tipo de redes aprende a reconstruir sus propias entradas, permitiéndoles aprender representaciones codificadas del *dataset*. Son utilizadas en detección de anomalías y preprocesamiento o para reducción de dimensionalidad.

Aplicaciones

Las redes neuronales artificiales han evolucionado desde herramientas académicas hasta convertirse en componentes esenciales de numerosas soluciones prácticas en la industria, la salud, la economía y otros sectores. Su capacidad para modelar relaciones complejas entre datos, adaptarse a contextos diversos y mejorar continuamente su desempeño las ha hecho especialmente valiosas en un mundo digital saturado de información.

Como menciona Livingstone (2011) “una red neuronal bien entrenada puede superar el desempeño humano en tareas específicas cuando existe una suficiente cantidad de datos estructurados”. Esta capacidad las convierte en aliadas poderosas en muchos dominios. A continuación, se presenta varias de las aplicaciones de las redes neuronales artificiales.

Medicina. Una de las áreas con mayor impacto ha sido la medicina, donde las redes neuronales han demostrado su eficacia en la detección de patrones en imágenes médicas (radiografías, resonancias), análisis de señales fisiológicas (EEG, ECG) y diagnóstico temprano de enfermedades como Alzheimer o cáncer. Por ejemplo, las CNN (Redes Neuronales Convolucionales) han sido fundamentales en el análisis automático de imágenes, siendo usado en la segmentación de tejidos y clasificación de tumores con tasas de precisión superiores al 90 % (Pons et al., 2022).

Industria. En entornos industriales, las redes neuronales se aplican al mantenimiento predictivo, analizando datos de sensores para anticipar fallas en los equipos. Los sistemas con redes neuronales artificiales pueden detectar anomalías en el comportamiento de motores, turbinas y maquinaria pesada con alta confiabilidad (Livingstone, 2011).

Visión por computadora. La visión por computadora es una de las áreas donde el impacto de las redes neuronales ha sido más visible. Ya que tiene un amplio campo de aplicaciones como es el reconocimiento facial, detección de objetos y reconocimiento óptico de caracteres (OCR) que son solo algunas de las aplicaciones comunes (Zhao et al., 2019).

En el ámbito de la detección de objetos, algunos de los modelos más utilizados son:

- YOLO
- R-CNN
- Fast R-CNN
- Faster R-CNN
- SSD
- RetinaNet
- DETR

Por otro lado, Jain (2023) muestra varias plataformas que integran uno o más modelos OCR dentro de una herramienta o librería completa como:

- Tesseract OCR
- EasyOCR
- PaddleOCR
- Keras-OCR

Procesamiento del lenguaje natural (PLN). Las redes neuronales también han revolucionado el procesamiento del lenguaje natural. Modelos como los RNN, y más recientemente las arquitecturas basadas en atención y transformers, han permitido desarrollar aplicaciones como:

- Traducción automática (Google Translate, DeepL)
- Chatbots
- Generación automática de texto

Estos sistemas están presentes en asistentes virtuales, buscadores inteligentes y plataformas de soporte al cliente. Pons et al. (2022) destaca los avances recientes en generación de lenguaje natural, mismos que permiten transformar documentos extensos en información digerible.

Software y Recursos Tecnológicos

En la actualidad, el implementar soluciones basadas en inteligencia artificial depende del uso de herramientas tecnológicas especializadas. Entre los recursos más utilizados destacan lenguajes de programación como Python, ampliamente reconocido por su sintaxis sencilla y versatilidad para diversas aplicaciones, entre ellas de inteligencia artificial (Lutz, 2013).

De igual forma, el uso de librerías especializadas como OpenCV ha permitido un desarrollo importante en el procesamiento digital de imágenes, facilitando tareas de segmentación, detección de patrones y visión por computadora (Bradski & Kaebler, 2008). Estas herramientas, combinadas con métodos sistemáticos de anotación de datos y frameworks como TensorFlow o PyTorch, han desarrollado un entorno tecnológico adaptable que favorece el entrenamiento de modelos de inteligencia artificial (Abadi et al., 2016; The PyTorch Foundation, n.d.).

Python

Python fue diseñado por Guido van Rossum en los años ochenta, fecha desde la cual ha ido evolucionando hasta convertirse en una herramienta fundamental en múltiples proyectos lo que le ha permitido ganar una posición privilegiada en el desarrollo tecnológico gracias a su claridad sintáctica, su extensa comunidad y disponibilidad de librerías especializada (Python Software Foundation, n.d.).

Python es un lenguaje de programación de alto nivel, con el objetivo ser fácil de leer y escribir ya que cuenta con una sintaxis clara y concisa. Este lenguaje de programación trata de mantener la legibilidad del código y coherencia estructural. Además, presenta un enfoque multiparadigma que permite desarrollar desde un siempre scripts hasta sistemas complejos de análisis de datos, simulaciones numéricas y etc.

Cuenta con un extenso conjunto de librerías que potencian sus capacidades en áreas como la computación (numpy, scipy), visualización de datos (matplotlib, seaborn) desarrollo web y aprendizaje automático (tensorflow, pytorch) (Lutz, 2013).

OpenCV

OpenCV (Open Source Computer Vision Library) es una librería de visión por computadora de código abierto diseñada para aplicaciones que requieren análisis de imágenes y videos en tiempo real. Fue desarrollada por Intel en el año 2000 con el objetivo de promover la investigación en visión por computadora y acelerar su adopción en la industria (Bradski & Kaebler, 2008). A partir de ahí, se ha consolidado como una de las herramientas más potentes para el procesamiento digital de imágenes.

Esta librería presenta soporte con varios lenguajes de programación, entre estos C++, Python y Java, OpenCV ofrece módulos especializados para tareas como detección de bordes, seguimiento de objetos y reconocimiento facial. También, cuenta con la capacidad de integrarse con librerías como `NumPy` y frameworks como TensorFlow o PyTorch lo que permite el desarrollo de aplicaciones de inteligencia artificial en visión por computadora (Pulli et al., 2012).

Métodos de Anotación y Etiquetado de Datos

El etiquetado o anotación de datos es un procedimiento fundamental en el desarrollo de sistemas de inteligencia artificial, sobre todo en el aprendizaje automático, que necesita conjuntos de datos clasificados con anterioridad para que los modelos puedan aprender a identificar patrones. El propósito de este procedimiento es organizar datos sin procesar, como imágenes, textos o audios, en categorías o etiquetas para su utilización en tareas concretas de entrenamiento (Russakovsky et al., 2015).

Hay diverso métodos de etiquetado, que varían dependiendo el volumen de la base de datos, tipo de dato y la precisión requerida, mismos que se presentan a continuación:

- Etiquetado manual: este método es realizado por las personas, lo que garantiza alta precisión, pero consume mucho tiempo. Usualmente se utiliza en pequeños proyectos o cuando se requiera bastante exactitud.
- Etiquetado asistido por IA: el método relaciona predicciones automáticas generadas por modelos previamente entrenados y la verificación por parte de una persona. Este enfoque mejora la eficiencia y se emplea en plataformas como Roboflow, que sugiere etiquetas basadas en modelos entrenados y permite corregir errores de forma visual e intuitiva.
- Etiquetado colaborativo: varios anotadores trabajan en paralelo o de forma secuencial, ya sea para aumentar el volumen de datos anotados o validar etiquetas mediante votación o consenso.

Existen diversas herramientas empleadas para desarrollar el etiquetado de datos entre las cuales se encuentran:

- LabelImg: esta herramienta permite realizar anotación de imágenes y exportar los datos en formato Pascal VOC o YOLO, lo que la hace ideal para pequeños proyectos.
- CVAT (*Computer Vision Annotation Tool*): es una herramienta desarrollada por Intel, la cual ofrece una interfaz web para anotación colaborativa, con soporte para vídeo e imágenes.
- *Label Studio*: presenta una plataforma de código abierto la cual permite etiquetar imágenes, texto y audio.
- *Roboflow Annotate*: es una solución en la nube que permite importar, etiquetar, preprocesar y exportar bases de datos en múltiples formatos. También, incluye herramientas de autoetiquetado y colaboración.

Frameworks y Librerías para IA

La creación de modelos de inteligencia artificial ha sido fomentada por librerías y frameworks que simplifican su entrenamiento y optimización. Estas herramientas posibilitan que los desarrolladores se concentren en experimentar y diseñar arquitecturas, sin la necesidad de preocuparse por las operaciones de bajo nivel. Algunas de las más comunes son TensorFlow, YOLO y PyTorch, cada una con rasgos específicos que se adaptan a diferentes tipos de proyectos.

PyTorch. Es un framework de aprendizaje profundo de código abierto, ampliamente reconocido por su enfoque dinámico y flexible. Una de sus principales fortalezas es el uso de define-by-run, lo que significa que el gráfico computacional se construye dinámicamente durante la ejecución del código. Esto lo hace especialmente útil para tareas de investigación y desarrollo rápido de prototipos (The PyTorch Foundation, n.d.)

TensorFlow. Es un framework desarrollado por Google y lanzado en 2015, diseñado para realizar cálculos numéricos de forma eficiente a través de gráficos computacionales. Su arquitectura permite ejecutar operaciones en diversos dispositivos (CPUs, GPUs, TPUs) y en distintos entornos (servidores, dispositivos móviles, navegadores) (Abadi et al., 2016).

YOLO (You Only Look Once). Es una arquitectura de detección de objetos la cual realiza predicciones precisas y rápidas en tiempo real. YOLO analiza toda la imagen en un único recorrido a través de la red, lo que posibilita un incremento significativo en la velocidad sin poner en riesgo el rendimiento. YOLO ha experimentado una evolución continua desde su versión inicial en 2016. Ha incorporado mejoras arquitectónicas, ha optimizado la exactitud de la detección de objetos pequeños y se ha ajustado a distintos contextos de implementación. La versión más reciente de Ultralytics, YOLOv11, brindará una mayor precisión y eficiencia. YOLOv11 se ha posicionado como una de las soluciones más completas y asequibles para trabajos de visión por computadora, debido a su integración con PyTorch y herramientas como Label Studio o Roboflow. (Ultralytics Inc., 2025).

Capítulo III

Diseño e Implementación

En este capítulo se presenta la evaluación de distintas arquitecturas de inteligencia artificial para la detección de elementos eléctricos. Además, se expone la metodología utilizada para el desarrollo de la aplicación que permite el reconocimiento de los elementos del diagrama de un circuito eléctrico básico y sus interconexiones.

Selección de la arquitectura de detección de objetos

Para seleccionar la arquitectura de inteligencia artificial más adecuada para la detección de los componentes eléctricos se hizo una comparación entre las siguientes:

- Faster R-CNN (Regions with CNN features)
- YOLO (You Only Look Once)
- SSD (Single Shot MultiBox Detector)
- RetinaNet
- DETR (Detection Transformer)

Faster R-CNN es una arquitectura que introduce la red propuesta regional (RPN) reemplazando el algoritmo de búsqueda de selección en Fast R-CNN, eliminando la necesidad de procesos externos para generar regiones candidatas. Este tipo de arquitectura se usa generalmente en aplicaciones que requieren alta precisión y procesamiento en tiempo real como vehículos autónomos de baja velocidad (Ren et al., 2016).

YOLO es una arquitectura de una sola etapa que para hacer la detección divide la imagen en una cuadrícula y predice las clases y cuadros delimitadores directamente en una sola red. Utilizada en aplicaciones en tiempo real como robots, drones y sistemas interactivos que requieran una mínima latencia (Redmon et al., 2016).

Al igual que YOLO, SSD es una arquitectura de una sola etapa la cual mejora la detección multiescala, permitiendo reconocer objetos grandes y pequeños simultáneamente.

Usada ampliamente en aplicaciones móviles o embebidas que requieran un balance entre velocidad y precisión (Liu et al., 2016).

De la misma manera, RetinaNet es una arquitectura de una sola etapa que aplica focal loss para mejorar la detección de objetos raros muy poco representados. Es aplicada en escenarios con objetos poco frecuentes, como contabilizar fauna silvestre. Pero presenta un mayor costo computacional frente a YOLO o SSD (Lin et al., 2017).

Por otro lado, DETR incluye el uso de transformadores en detección de objetos, lo que elimina la necesidad de usar de anclajes y regiones propuestas para usar mecanismos de atención para la detección de objetos de forma global y directa. El uso de esta arquitectura permite realizar análisis y comprensión de imágenes en contexto (Carion et al., 2020).

Considerando diferentes criterios de aplicabilidad al detectar elementos de un circuito eléctrico, La Tabla 2 presenta una comparación entre las diferentes arquitecturas.

Tabla 2

Comparación de arquitecturas de inteligencia artificial para detección de objetos

Criterio	Faster R-CNN	SSD	YOLO	RetinaNet	DETR
Precisión (mAP)	Alta	Media	Media-Alta	Alta	Muy alta
Velocidad de predicción	Lenta	Alta	Muy Alta	Media	Lenta
Robustez	Alta	Media	Media	Alta	Alta
Costo computacional	Alto	Bajo	Medio-Bajo	Medio	Alto
Facilidad de entrenamiento	Media	Alta	Muy Alta	Media	Baja

Considerando que el diagrama de un circuito eléctrico necesita una buena presión con un buen rendimiento computacional debido a que contiene elementos relativamente pequeños

y similares entre sí, además de presentar una densidad importante de elementos en la misma imagen, YOLO es la arquitectura para la detección de elementos de un circuito eléctrico más adecuada, ya que ofrece una buena combinación entre velocidad de predicción, precisión, costo computacional y facilidad de implementación.

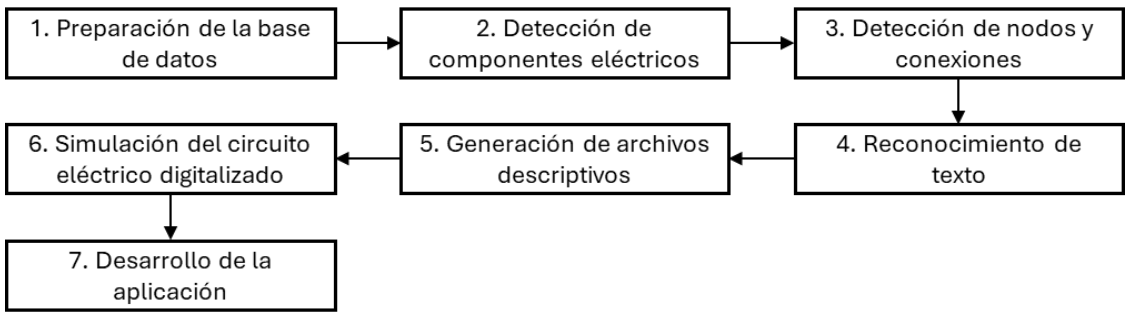
Esto debido a que arquitecturas como RetinaNet y Faster R-CNN presentan alta presión en la detección, pero requiere un mayor costo computacional y tiene una menor velocidad de predicción. Por otro lado, DETR presenta limitaciones al hacer predicciones en imágenes con objetos pequeños y cuenta con un alto costo computacional para su entrenamiento, lo que lo hace menos viable. Y SSD está enfocado mayormente para usarlo en aplicaciones móviles o embebidas.

Metodología

La metodología propuesta comprende 7 etapas importantes (Figura 14) para la digitalización de diagramas de un circuito eléctrico y posteriormente su simulación. Las etapas por describirse en este capítulo son: preparación de la base de datos, detección de los componentes eléctricos haciendo uso de una red neuronal convolucional YOLO, código desarrollado en Python para la detección de nodos y conexiones, reconocimiento de texto, generación de archivos descriptivos (Netlist), simulación del circuito eléctrico digitalizado y desarrollo de una aplicación que permita la selección del diagrama de un circuito eléctrico, su procesamiento y simulación.

Figura 14

Etapas para la digitalización y simulación del diagrama de un circuito eléctrico



Preparación de la base de datos

Para la preparación de la base de datos se estableció que el modelo de detección será entrenado con diagramas de circuitos eléctricos donde los componentes sigan el estándar de símbolos eléctricos “IEEE Standard 315-1975”, considerando únicamente los componentes y simbología presentada en la Tabla 1.

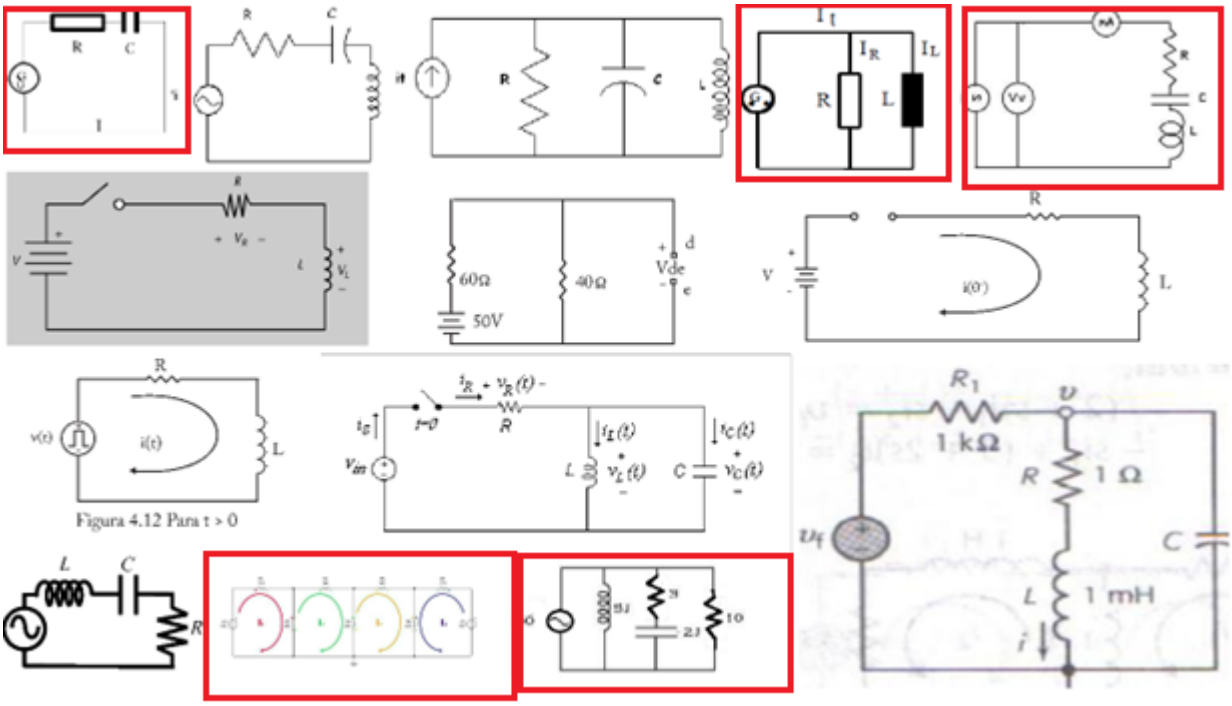
Además, de que el reconocimiento de elementos se hará únicamente de imágenes tomadas de documentos en formato pdf con el fin de mejorar la precisión del procesamiento y la detección de los elementos. Y también, deben estar en formato .png o .jpg con una resolución mínima de 640 pixeles de ancho.

El desarrollo de la base de datos en este proyecto parte en primera instancia del conjunto de datos desarrollado en el trabajo de (Muñoz Chuquimarca, 2021), el cual cuenta con un total de 735 imágenes en formato .png con elementos de circuitos eléctricos básicos como son fuentes, resistencias, capacitores e inductores.

Sin embargo, no se consideraron todas las imágenes ya que en algunas sus elementos no seguían el estándar de símbolos eléctricos IEEE/ANSI como se puede observar en la Figura 15.

Figura 15

Ejemplo de imágenes de la base de datos



Nota. La imagen muestra un ejemplo de los diferentes componentes eléctricos contenidos en las imágenes de la base de datos de (Muñoz Chuquimarca, 2021).

Después de hacer la selección de las imágenes acorde a lo establecido previamente, la base de datos se redujo a 670 imágenes. Sin embargo, esta cantidad de datos no es suficiente para entrenar un modelo de detección y tener buenos resultados.

Kellenberger et al. (2018) menciona que con un rango de 50 a 100 imágenes por clase se pueden obtener buenos resultados en detección de objetos con modelos preentrenados y prácticas de entrenamiento adecuadas. Al igual que opiniones de la comunidad en foros técnicos como Reddit, donde usuarios han compartido sus experiencias entrenando modelos de detección de objetos con cantidades limitadas de datos. Especialmente, se menciona que es posible obtener resultados funcionales utilizando alrededor de 100 imágenes por clase cuando se emplean técnicas de transferencia de aprendizaje en modelos preentrenados, siempre que la base de datos esté bien etiquetada y equilibrada (Reddit, Inc., 2021).

Con estas consideraciones y de acuerdo con las clases mostradas en la Tabla 3, se establece que la base de datos debe estar compuesta por un mínimo de 900 imágenes para el entrenamiento.

Tabla 3

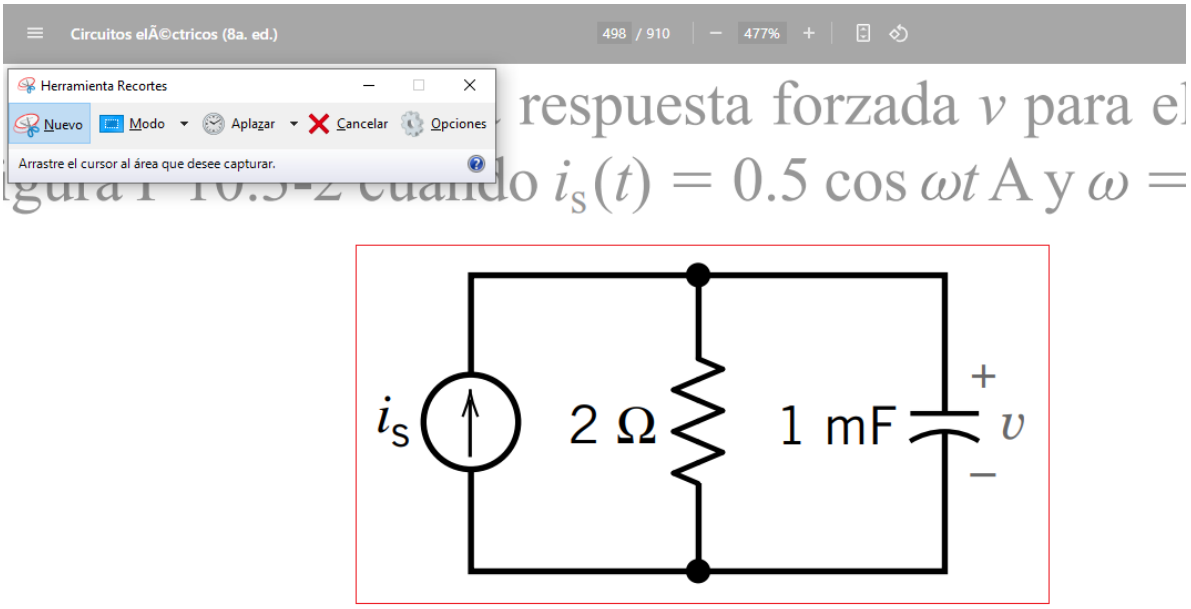
Etiquetas numéricas de las clases correspondiente a cada elemento

Etiqueta	Clase
0	Batería
1	Capacitor
2	Fuente de corriente
3	Fuente de voltaje AC
4	Fuente de voltaje DC
5	Inductor
6	Pila
7	Resistencia
8	Tierra

Para aumentar la cantidad de imágenes de la base de datos, se realizó capturas de imágenes de circuitos eléctricos usando la herramienta de recorte de Windows, como se muestra en la Figura 16. Cuidando que la imagen capturada cumpla con la resolución mínima establecida al igual que su formato al momento de guardar la misma.

Figura 16

Captura de diagramas de circuitos eléctricos



Para mantener la consideración previa de que cada clase debe tener al menos 100 imágenes, a las 670 imágenes de la base de datos de (Muñoz Chuquimarca, 2021) se las segmento por clase. La Tabla 4 muestra la cantidad de imágenes que contienen a cada clase en la base de datos inicial.

Tabla 4

Cantidad de imágenes por clase en la base de datos inicial

Clase	Imágenes
Batería	63
Capacitor	405
Fuente de corriente	122
Fuente de voltaje AC	53
Fuente de voltaje DC	330

Clase	Imágenes
Inductor	360
Pila	23
Resistencia	593
Tierra	87

Teniendo en cuenta la cantidad de imágenes por clase, se agregó las imágenes necesarias a la base de datos hasta que las clases con menos de 100 imágenes lleguen a este valor. Posteriormente, se agrupo todas las imágenes en una sola carpeta. Debido a que varios diagramas se repiten en la segmentación por clase, más las nuevas capturas incorporadas, da un total de 995 imágenes en la base de datos.

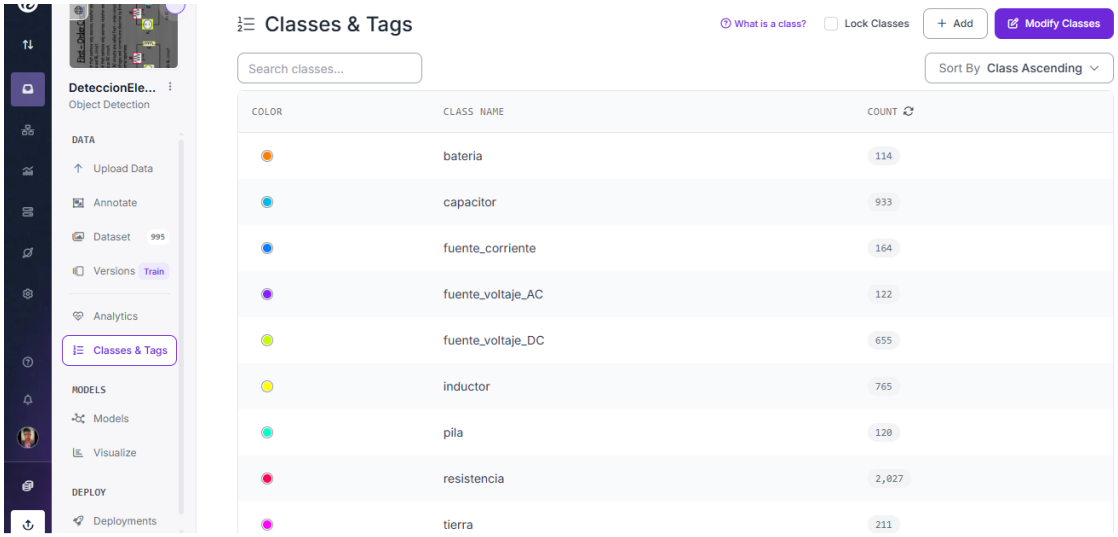
Etiquetado de datos

Para la detección de los componentes eléctricos se usó una red neuronal convolucional YOLO, con lo cual para su entrenamiento se debe realizar un proceso de etiquetado de datos de cada uno de los elementos de los circuitos. Como se desarrolló en el capítulo anterior, el proceso de etiquetado consiste en asignar información estructurada de los elementos de los circuitos eléctricos de la base de datos. Las etiquetas asignadas a cada elemento se muestran en la Tabla 3, de tal forma de que cada imagen de la base de datos mediante un cuadro delimitador se selecciona cada componente encontrado y se asigna una de las etiquetas.

Antes de iniciar el entrenamiento, es necesario generar archivos de texto que contengan la información correspondiente a cada imagen de la base de datos. Misma que debe pasar por un proceso de etiquetado, donde se identifican y marcan los elementos presentes en los circuitos que se desea que la red neuronal reconozca. Para realizar este proceso, se emplea la herramienta web Roboflow (Roboflow, Inc., 2025), la cual permite cargar la base de datos y crear las etiquetas necesarias para los nueve componentes que serán detectados. Este procedimiento se muestra en la Figura 17.

Figura 17

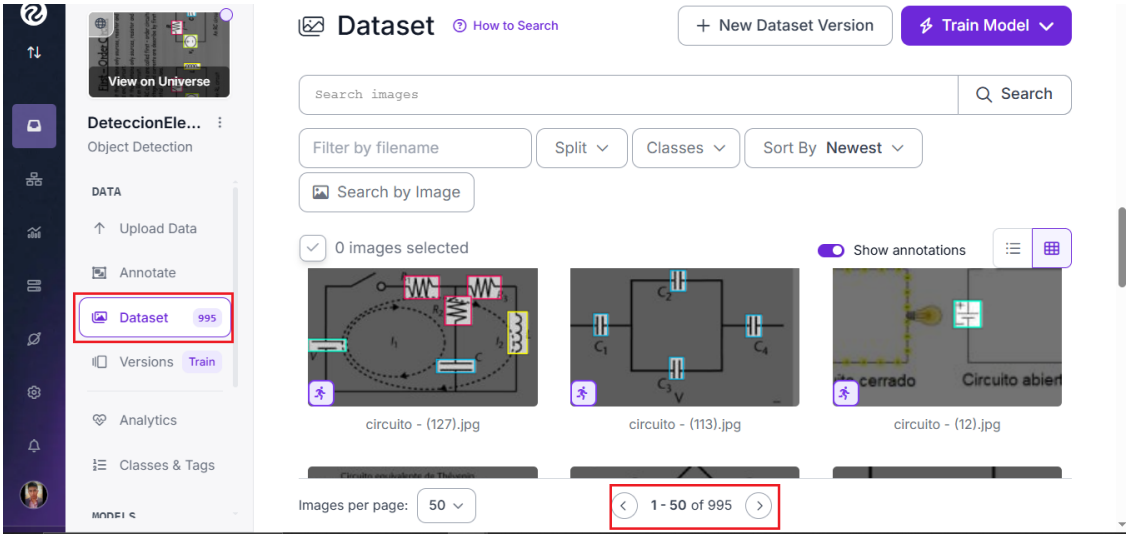
Clases para el etiquetado de datos



Una vez creada las etiquetas y cargada la base de datos se realiza el proceso de etiquetado. Como se observa en la Figura 18 en el apartado de “Dataset” se encuentran cargadas todas las imágenes de la base de datos.

Figura 18

Base de datos cargada en Roboflow

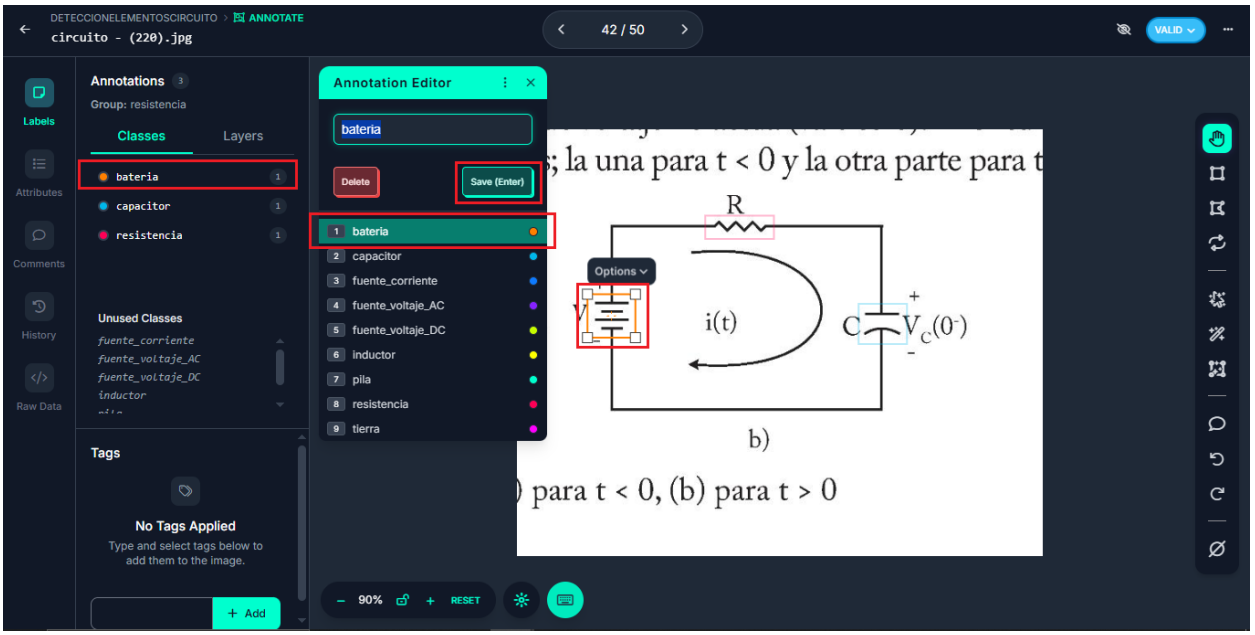


Para proceder con el etiquetado se selecciona la primera imagen del dataset. Una vez seleccionada la imagen se abre la ventana mostrada en la Figura 19, la cual permite crear

ventanas rectangulares que cubran la región de interés correspondiente a los elementos eléctricos presentes en ese diagrama, y a su vez asignar las etiqueta previamente creadas.

Figura 19

Clases para el etiquetado de datos



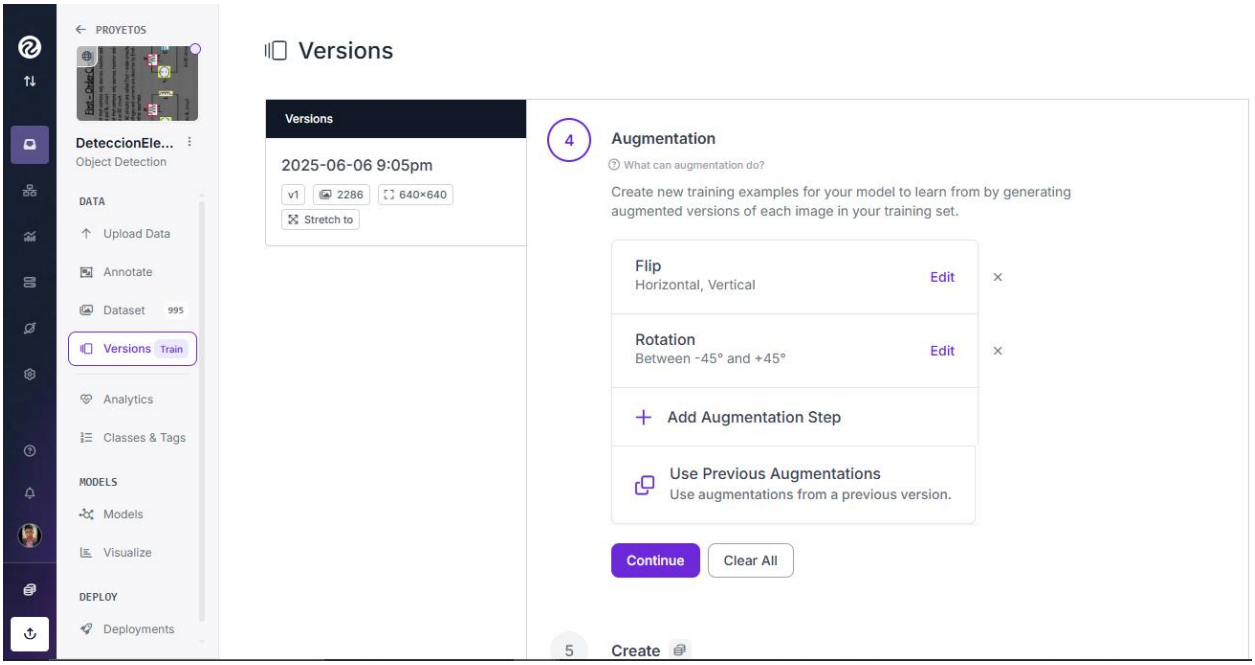
En la Figura 19, se presenta como ejemplo el etiquetado de una batería, a la que en primera instancia se marca la región de interés y después se elige la etiqueta correspondiente, en este caso “batería”, este procedimiento se replica con los demás elementos de las 995 imágenes del dataset.

Una vez finalizado el etiquetado, se debe exportar el dataset en junto con sus etiquetas en un formato compatible con YOLO. El apartado de Versions dentro del entorno de Roboflow, permite preparar las imágenes del dataset y los compila de acuerdo con la versión de YOLO deseada. Además, este apartado permite aplicar transformaciones de imagen a las imágenes del dataset, con el objetivo de incrementar la base de datos que se utilizara para entrenar el modelo de detección de objetos.

La Figura 20, muestra los medidas o tipos de aumentos que se le quiere dar al dataset, en este caso se seleccionó volteo horizontal y vertical, y rotación de la imagen entre -45° y +45°.

Figura 20

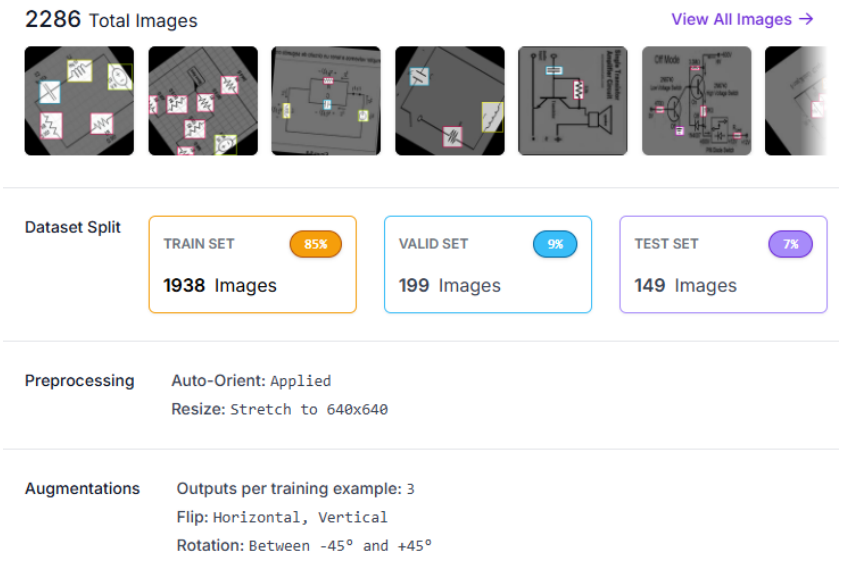
Preparación del dataset



Después de dar clic en continuar y crear, al haber establecido nuevas imágenes con versiones aumentadas, la base de datos ahora es de 2286 imágenes como se muestra en la Figura 21. Además, la cantidad total de imágenes se divide en 3 conjuntos de datos importantes para el entrenamiento del modelo como son: entrenamiento (train), validación (valid) y prueba (test) en porcentajes del 85%, 9% y 7% respectivamente. El agrupamiento de las imágenes en estos conjuntos de datos es realizado de manera aleatoria por la plataforma de Roboflow, sin embargo, en la opción “ver todas las imágenes” se puede establecer o cambiar de forma manual el conjunto de datos al que se asignó a cada una de las imágenes.

Figura 21

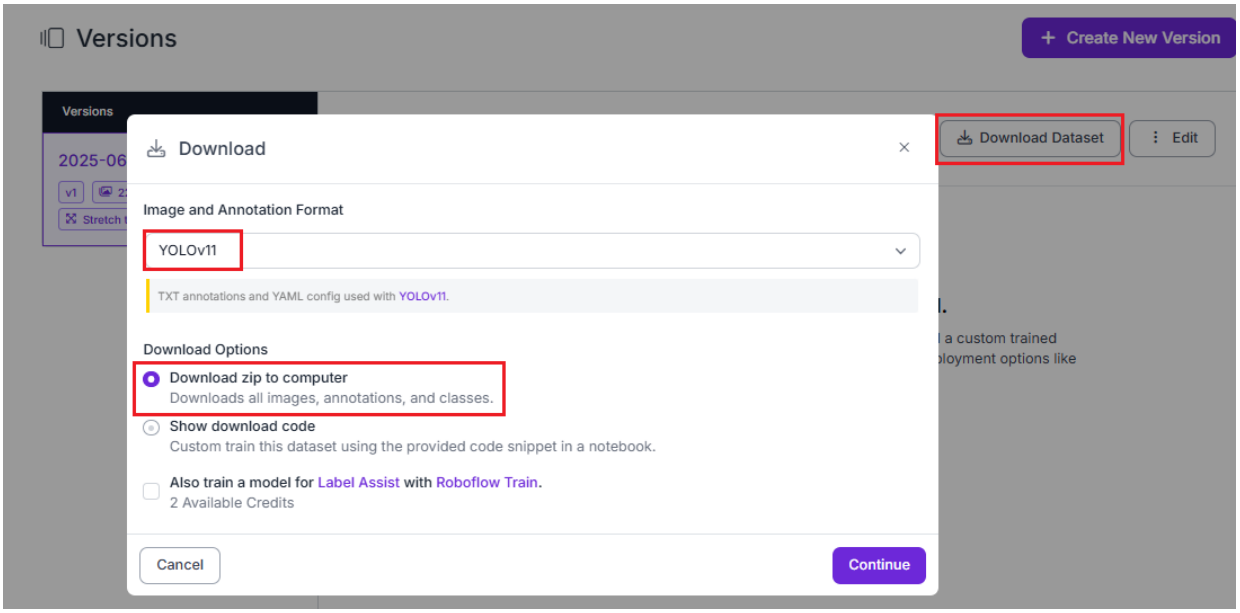
Total de imágenes del dataset



Al dar clic sobre la opción “Descargar dataset” mostrada en la Figura 22, se despliega una ventana donde permite establecer el formato de anotación de los datos, en este caso se selecciona YOLOv11 que es la versión del modelo de detección que se va a entrenar.

Figura 22

Descarga de la base de datos



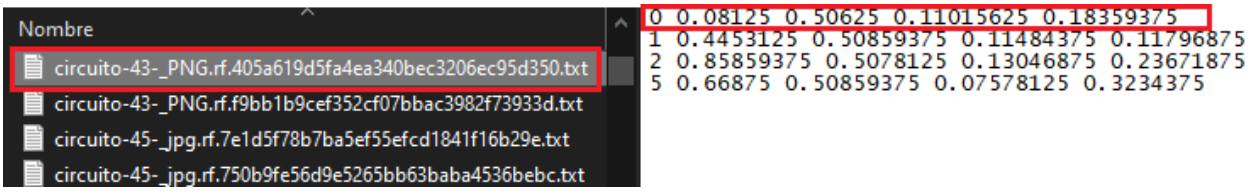
Por último, se elegie la opción descargar zip al computador, la cual proporciona todas las imágenes del dataset, las anotaciones de cada una de las imágenes en archivos de texto y sus clases.

Los archivos de texto contienen información importante correspondiente al etiquetado que se realizó a cada imagen. La Figura 23, muestra el ejemplo del contenido de un archivo de texto donde se puede apreciar la siguiente información:

- Primera columna: etiqueta correspondiente a cada elemento.
- Segunda columna: posición central en el eje *x* del rectángulo delimitador.
- Tercera columna: posición central en el eje *y* del rectángulo delimitador.
- Cuarta columna: ancho del rectángulo delimitador.
- Quinta columna: alto del rectángulo delimitador.

Figura 23

Anotaciones de las imágenes contenidas en los archivos de texto

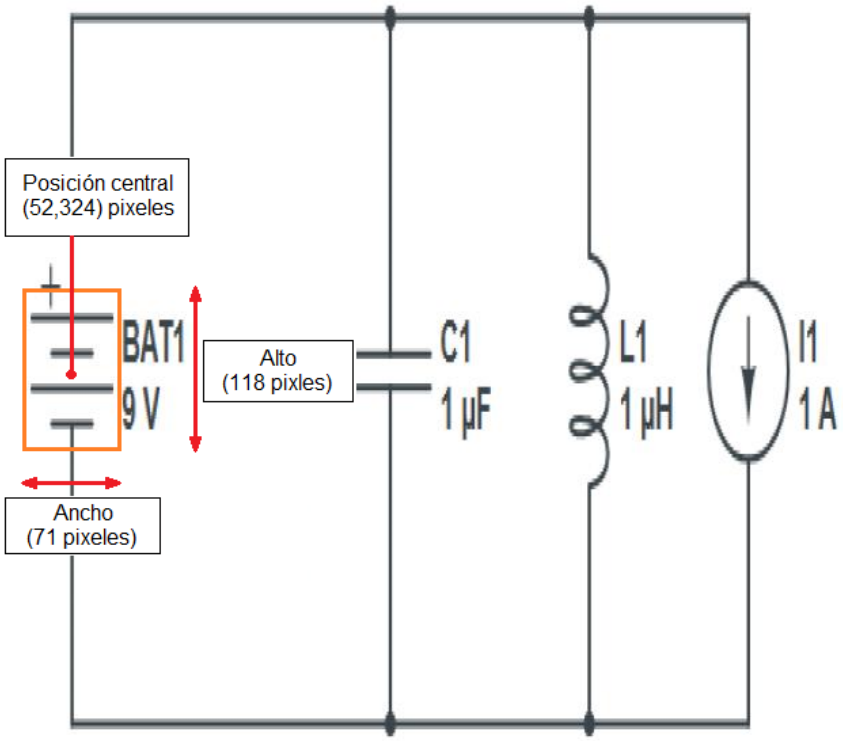


Es importante recalcar que estas anotaciones se encuentran normalizadas. Para tener un mayor contexto en la Figura 24 se presenta un ejemplo con los valores explicados previamente de la Figura 23.

Para obtener los valores de los pixeles mostrados en el ejemplo de la Figura 24, se parte de las anotaciones de los archivos de texto además de conocer el tamaño de la imagen en pixeles, en este caso la imagen tiene un tamaño de 640x640 pixeles.

Figura 24

Ejemplo de las anotaciones contenidas en los archivos de texto



Para obtener los valores mostrados en la Figura 24, se realiza una multiplicación del valor normalizado por ancho o alto del tamaño de la imagen acorde al parámetro deseado, ejemplo:

- Posición central eje *x*: 0.08125 x 640 = 52 pixeles.
- Posición central eje *y*: 0.50625 x 640 = 324 pixeles.
- Ancho del rectángulo delimitador: 0.11015625 x 640 = 71 pixeles.
- Alto del rectángulo delimitador: 0.18359375 x 640 = 118 pixeles.

En este momento ya se cuenta con las bases de datos junto con los archivos de texto generados durante la fase de etiquetado, cumpliendo con los requerimientos necesarios para entrenar el modelo de detección de objetos YOLO.

Detección de componentes eléctricos

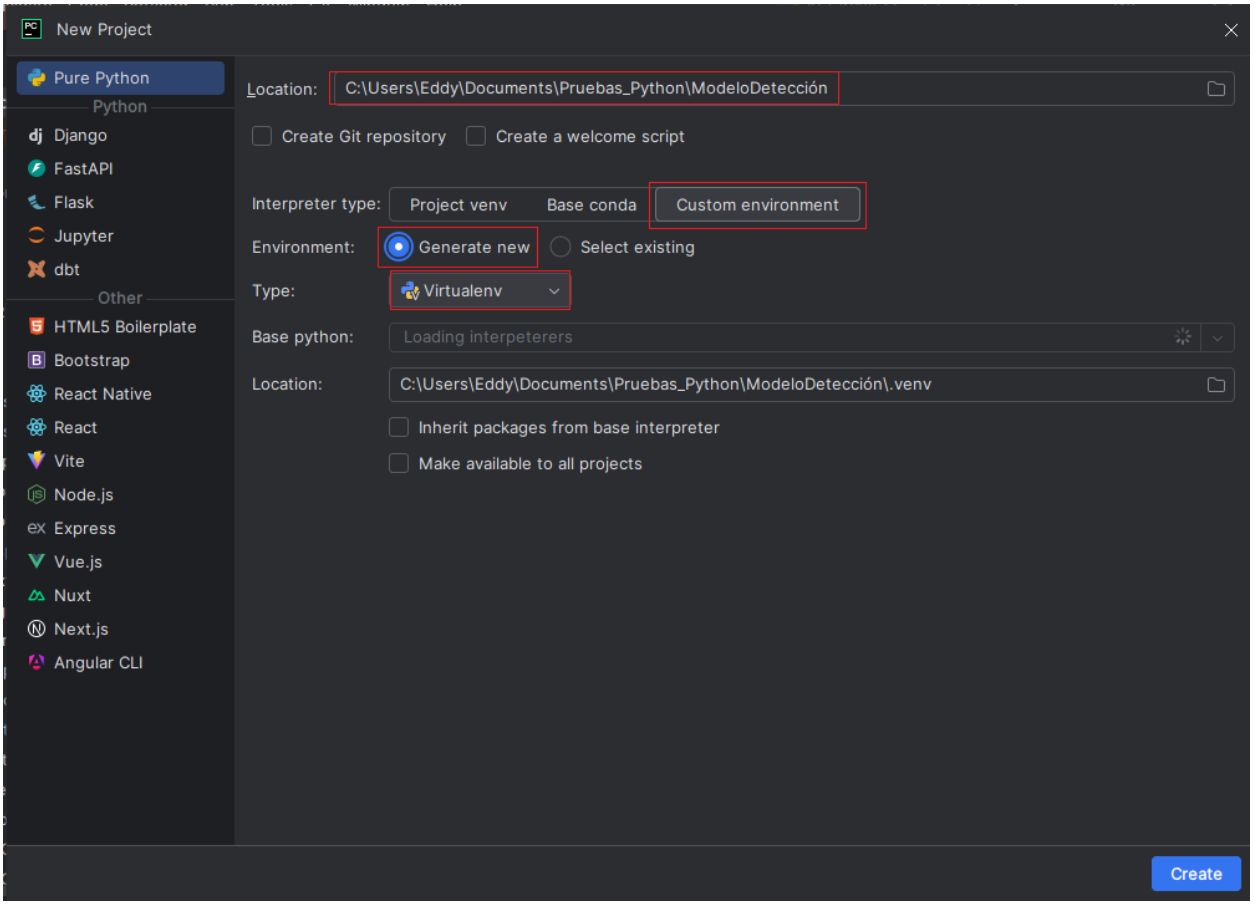
Para realizar el entrenamiento del modelo de detección al igual que las demás etapas del proyecto, se hizo uso del lenguaje Python en el Entorno de Desarrollo Integrado (IDE por sus siglas en inglés) “PyCharm”. Este IDE permite la edición, depuración, control de versiones y gestión de entornos en una sola herramienta (JetBrains, 2025).

Creación del proyecto

El primer paso es crear un nuevo proyecto dentro de este IDE. Como se muestra en la Figura 25. En primer lugar, se establece la dirección donde se va a alojar el proyecto al igual que el nombre del proyecto “ModeloDeteccion” y adicionalmente se debe crear un nuevo entorno virtual.

Figura 25

Creación de un nuevo proyecto en el IDE PyCharm



Un entorno virtual es una herramienta de Python que permite crear un entorno aislado para los proyectos, con el objetivo de poder instalar paquetes y librerías sin afectar el sistema global ni a otros proyectos.

De esta forma, el entorno virtual creado es de tipo venv el cual es un entorno aislado incluido por defecto en los módulos de Python, es ideal para el desarrollo de aplicaciones con la ventaja de ser ligero y rápido de crear.

Entrenamiento de la red neuronal

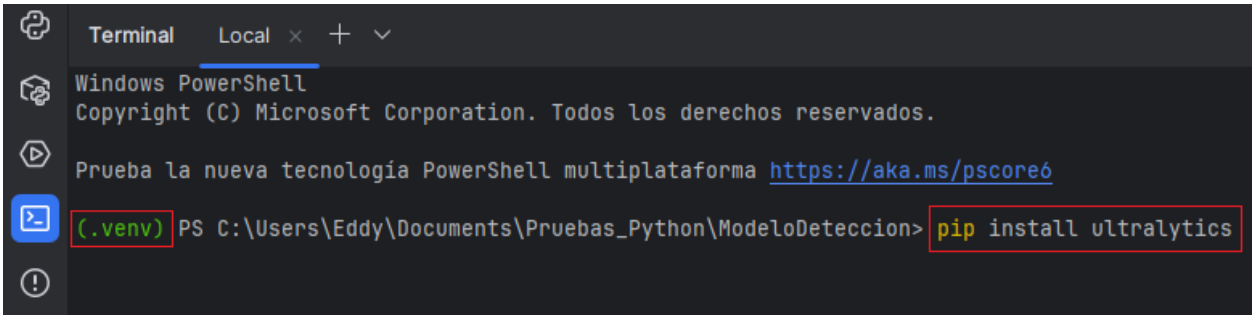
Para llevar a cabo la detección de objetos, fundamental entrenar una red neuronal que sea capaz de aprender las características visuales de cada clase a identificar. En este caso en particular, trabajos previos han demostrado que el uso de redes neuronales convolucionales permite localizar y clasificar eficazmente los diferentes componentes de los circuitos eléctricos. Entre las distintas arquitecturas de detección de objetos disponibles, YOLO se destaca como uno de los más eficientes debido a su rapidez, precisión y capacidad para reconocer múltiples objetos. Estas ventajas, además de su éxito demostrado en tareas similares de identificación de elementos eléctricos, lo convierten en la elección más adecuada para esta aplicación.

Dentro del entorno virtual, para entrenar el modelo YOLO se debe instalar la librería `ultralytics` (Ultralytics Inc., 2025). La instalación de esta librería se hace en el terminar dentro del IDE de PyCharm, ya que de esta forma se garantiza que el entorno virtual en el cual se está instalando la librería esta activo y pertenece a nuestro proyecto. Además, para realizar la instalación se usa el comando `pip` como se muestra en la Figura 26. `Pip` es el sistema de gestión de paquetes de Python, usado para instalar o desinstalar librerías y herramientas desde el repositorio oficial de Python llamado PyPI (Python Package Index).

Con el comando presentado en la Figura 26 se instala la todas las versiones de YOLO incluida la versión más reciente que es “yolov11” la misma que se utilizó para entrenar el modelo.

Figura 26

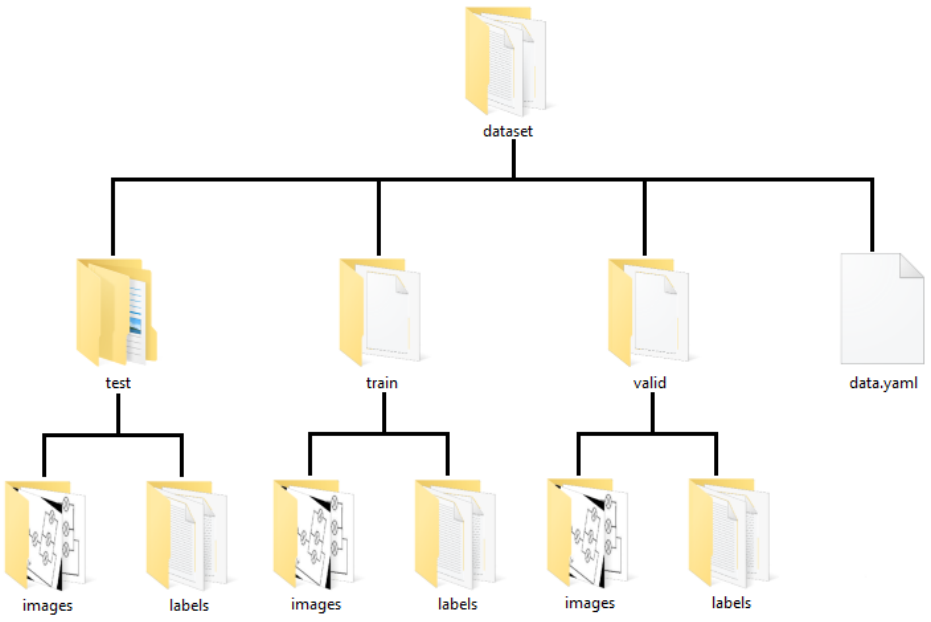
Instalación de la librería ultralytics en el entorno virtual del proyecto



Con el fin de llevar a cabo el entrenamiento del modelo, se debe copiar la base de datos, junto a sus anotaciones dentro del proyecto en una carpeta con nombre “dataset”. La carpeta “dataset” está estructurada como se muestra en la Figura 27, donde la base de datos se dividida en tres carpetas “train”, “valid” y “test”, estas carpetas a su vez contienen las carpetas “images” que contiene las imágenes y “labels” que contiene las anotaciones en archivos de texto. Además, se encuentra un archivo de configuración “data.yaml” el cual contiene la estructura y la información esencial del dataset.

Figura 27

Estructura de la carpeta dataset para el entrenamiento



Para realizar el entrenamiento de la red neuronal se crea un script (Figura 28) en el cual se estable los parámetros necesarios, mismos que se detallan a continuación:

- Se importa la clase YOLO desde la librería Ultralytics, para poder manejar los modelos YOLO.
- Cargamos un modelo preentrenado “yolov11s” como punto de partida. Este archivo contiene pesos ya aprendidos, que serán afinados con nuestro propio dataset.
- Para empezar el entrenamiento se indica la ruta al archivo “data.yaml”. Este archivo describe dónde están las imágenes de entrenamiento y validación, el número de clases, y cómo se llaman.
- Se establece el número de épocas o ciclos para el entrenamiento.
- Con el objetivo de reducir el uso de recursos de la PC y acelerar el entrenamiento, las imágenes se redimensionan a un tamaño de 512x512 píxeles.
- Un bajo valor de “*batch*” es adecuado para sistemas con CPU. Por este motivo se establece de 4, lo que indica que se entrenan cuatro imágenes a la vez.
- Se fuerza a que el entrenamiento use el procesador (CPU) en lugar de una GPU. Esto hace más lento el entrenamiento, pero es útil si el computador no cuenta con una GPU disponible.
- Se limita a 0 el número de procesos en paralelo para la carga de datos (*workers*). Esto se hace con el fin de prevenir errores en el entrenamiento CPU.
- Usamos el optimizador Adam, con el objetivo de que el entrenamiento sea más estable y converja mejor sin tanto ajuste.
- Establecemos una tasa de aprendizaje inicial (lr0) igual a 0.0005. Debido a que usar un valor bajo ayuda a evitar que el entrenamiento se vuelva inestable.

- Se aplica una regularización (*weight_decay*) para evitar que el modelo se sobreajuste al conjunto de entrenamiento.
- Indicamos que no se almacenen las imágenes en RAM, esto es útil ya que los recursos de la PC son limitados.
- Se activa la validación automática al final de cada época para evaluar el rendimiento del modelo.
- Finalmente se asigna un nombre al entrenamiento, para poder identificar los resultados generados al finalizar.

Figura 28

Fragmento de código utilizado para entrenar la red neuronal yolov11

```python
from ultralytics import YOLO

model = YOLO("yolo11s.pt") #Cargar modelo yolo11 pequeño

model.train(
    data="./dataset/data.yaml",
    epochs=50,           # No más de 50 en CPU, o tomará días
    imgsz=512,           # Más pequeño para acelerar y ahorrar RAM
    batch=4,             # Tamaño de lote muy bajo (CPU limitado)
    device='cpu',        # Forzar uso de CPU
    workers=0,           # Carga de datos más segura en CPU
    optimizer="Adam",     # Converge mejor sin tanto ajuste
    lr0=0.0005,           # Menor para evitar oscilaciones
    weight_decay=0.0005, # Regularización básica
    cache=False,         # No uses RAM extra en cache
    val=True,            # Realiza validación
    name="yolo11_modelo_deteccion"
)
```

Una vez establecido todos los parámetros de entrenamiento, se ejecuta el script y se espera a que finalice el entrenamiento. Una vez que culmina el entrenamiento de la red neuronal, se genera un archivo “best.pt” como se observa en la Figura 29. Este archivo contiene los pesos del modelo entrenado que obtuvo el mejor rendimiento durante el proceso

de validación. Además, guarda la información sobre la arquitectura del modelo para después poder hacer inferencias en imágenes nuevas.

Este archivo se copia a la carpeta principal del proyecto y se lo renombra como “detec_componentes.pt” para ser usado en la aplicación para la detección de los componentes eléctricos.

Figura 29

Validación del modelo una vez finalizado su entrenamiento

```
50 epochs completed in 20.102 hours.
Optimizer stripped from runs\detect\yolo11_modelo_deteccion\weights\last.pt, 19.2MB
Optimizer stripped from runs\detect\yolo11_modelo_deteccion\weights\best.pt, 19.2MB

Validating runs\detect\yolo11_modelo_deteccion\weights\best.pt...
Ultralytics 8.3.149  Python-3.12.9 torch-2.7.0+cpu CPU (Intel Core(TM) i5-7200U 2.50GHz)
YOLO11s summary (fused): 100 layers, 9,416,283 parameters, 0 gradients, 21.3 GFLOPs
                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100%|██████████| 25/25 [00:43<00:00,  1.73s/it]
                   all        199        980       0.94      0.941      0.957      0.761
               bateria         22         24          1       0.91      0.967      0.638
             capacitor        129        187      0.953      0.985      0.993      0.756
      fuente_corriente         24         29      0.966          1       0.98      0.866
     fuente_voltaje_AC         23         24       0.91      0.839      0.922       0.79
     fuente_voltaje_DC         92        129      0.977      0.977      0.994      0.886
              inductor        105        141      0.951      0.972      0.987      0.787
                  pila         21         21      0.808      0.905      0.838      0.644
           resistencia        163        372      0.995      0.978      0.993        0.8
                tierra         33         53      0.898      0.906      0.934      0.686
Speed: 1.2ms preprocess, 204.3ms inference, 0.0ms loss, 0.8ms postprocess per image
Results saved to runs\detect\yolo11_modelo_deteccion

Process finished with exit code 0
```

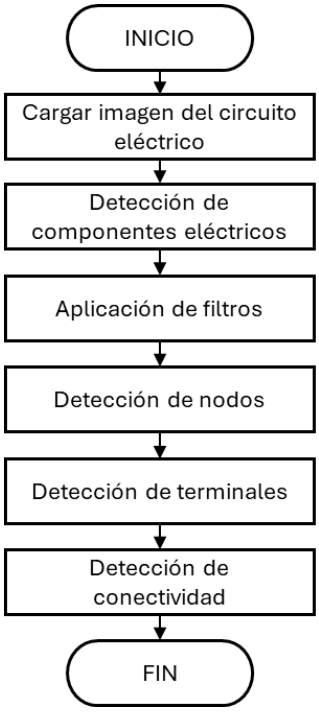
Los resultados obtenidos del entrenamiento de la red neuronal se detallan en el siguiente capitulo.

Detección de nodos y conexiones

Una vez entrenado el modelo de detección de componentes eléctricos, se desarrolla el código para identificar los nodos y conexiones entre elementos, este proceso se realiza siguiendo las etapas mostradas en el diagrama de flujo de la Figura 30.

Figura 30

Diagrama de flujo para la detección de nodos y conexiones



Como primer paso se carga la imagen que se quiere procesar, luego se ejecuta las siguientes etapas.

Detección de componentes eléctricos

Una vez cargada la imagen en el programa se carga el modelo entrenado desde el archivo “detec_componentes.pt”, el cual contiene los pesos y la configuración óptima aprendida tras el entrenamiento para detectar los componentes eléctricos.

Luego, se aplica el modelo a la imagen cargada para obtener una lista con los resultados de los *bounding boxes* (cajas delimitadoras) de los objetos detectado (Figura 31). En la lista se encuentra los siguientes detalles:

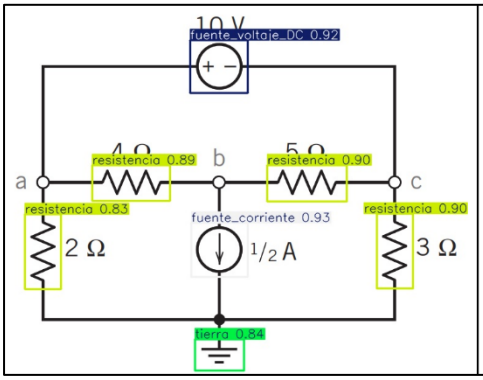
- Coordenadas xyxy: entrega las coordenadas de los punto mínimo y máximo de los *bounding boxes*.
- Coordenadas xywh: entrega las coordenadas del centro, ancho y alto de los *bounding boxes*.

- Confianza: entrega el nivel de confianza de cada detección en valores entre 0 y 1.
- Clases: entrega el índice de clase de cada objeto detectado.

La información de interés de toda la lista de resultados obtenida al aplicar el modelo de detección a la imagen para continuar con las siguientes etapas son las coordenadas xywh y el índice de clases.

Figura 31

Inferencia del modelo para la detección de elementos un diagrama de circuito eléctrico



Coordenadas xyxy	Coordenadas xywh	Confianza	Clase	Nombre de clase
(301, 57, 397, 144)	(349, 101, 96, 87)	0.906841	4	fuente_voltaje_DC
(605, 339, 666, 463)	(636, 401, 60, 123)	0.899048	7	resistencia
(428, 261, 555, 319)	(492, 290, 127, 57)	0.896418	7	resistencia
(143, 261, 270, 319)	(206, 290, 127, 57)	0.896022	7	resistencia
(32, 341, 91, 463)	(61, 402, 59, 122)	0.893364	7	resistencia
(304, 354, 396, 443)	(350, 398, 92, 89)	0.884093	2	fuente_corriente
(312, 545, 388, 596)	(350, 571, 76, 50)	0.874699	8	tierra

Nota. A la izquierda se muestra la detección de los elementos y a la derecha la lista de resultados de los *bounding boxes.*

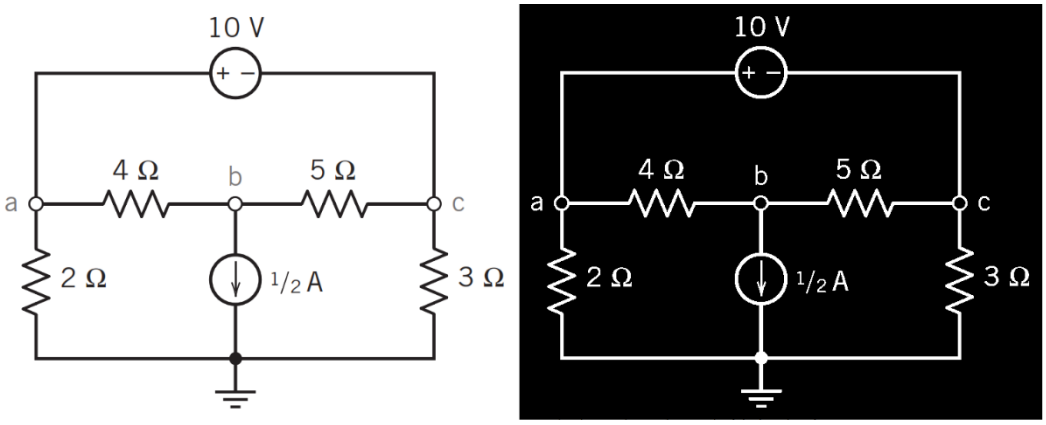
Aplicación de filtros

Para realizar la detección de nodos, terminales y conectividad entre elementos, se hizo un análisis de componentes conexos. Para realizar este análisis se convierte la imagen a escala de grises, luego se invierte la imagen para que las líneas sean blancas sobre el fondo negro y finalmente se binariza la imagen manteniendo un umbral fijo para que la imagen final contenga solo la información necesaria, eliminando pixeles que puedan crear confusión en el proceso de la detección.

La Figura 32, muestra un ejemplo de la aplicación de filtros previo a realizar el análisis de componentes conexos para la detección de nodos y terminales.

Figura 32

Ejemplo de la aplicación de filtros a un diagrama de circuito eléctrico



Nota. A la izquierda se muestra la imagen original y a la derecha la imagen filtrada.

Detección de nodos

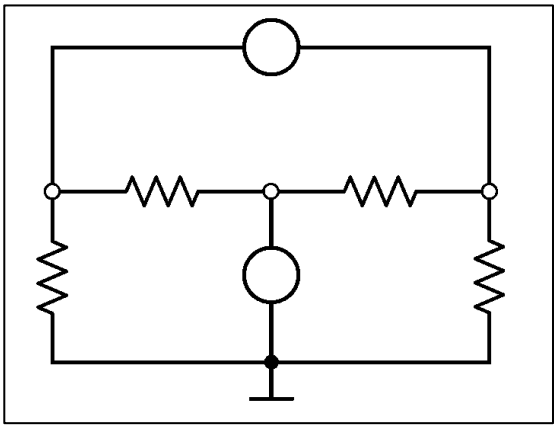
Con la imagen filtrada, se utiliza la función de componentes conexos incluida en la librería de OpenCV. Esta función permite identificar y etiquetar regiones de píxeles conectados en una imagen binaria, devolviendo las etiquetas de los componentes y también información estadística detallada de cada uno. Los valores que devuelve son:

- Número total de componentes conexos detectados, incluyendo el fondo (etiquetado como 0).
- Imagen del mismo tamaño que la original, donde cada píxel tiene el número de componente al que pertenece.
- Una matriz donde cada fila representa un componente, y las columnas las siguientes estadísticas: coordenada (x, y) del borde superior izquierdo del componente, ancho y alto del *bounding box* del componente, y el número de píxeles (área).
- Coordenadas del centroide de cada componente.

Para realizar la detección de nodos, como primer paso se extrae el componente conexo con mayor área ya que representa a todo el circuito conectado, eliminando el texto que para esta etapa es irrelevante (Figura 33).

Figura 33

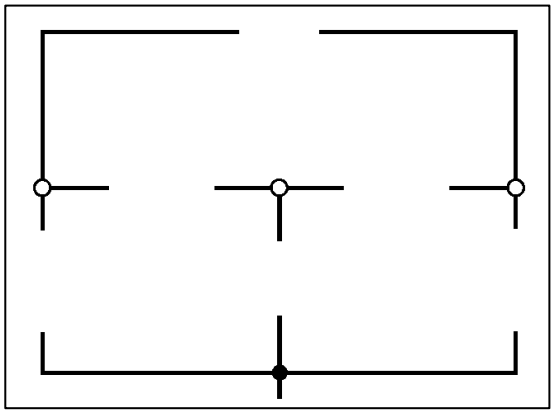
Visualización del componente conexo más grande



El siguiente paso es suprimir los componentes eléctricos para dejar únicamente las líneas de conexiones. Para esto se utilizó las coordenadas de los *bounding boxes* de los componentes eléctricos detectados, con estas coordenadas se crea áreas rectangulares donde los pixeles de estas áreas van a ser rellenadas con el color del fondo, obteniendo como resultado la imagen mostrada en la Figura 34.

Figura 34

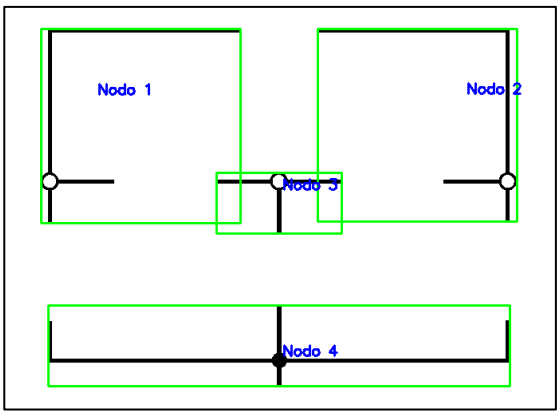
Eliminación de componentes eléctricos del circuito



Finalmente, las líneas de conexiones mostrados en la Figura 34 representan los nodos de un circuito eléctrico, así que, para identificarlos y etiquetarlos se vuelve a detectar los componentes conexos dentro de esta imagen y así se obtiene los nodos del circuito, tal como se muestra en la Figura 35.

Figura 35

Detección de nodos del circuito eléctrico

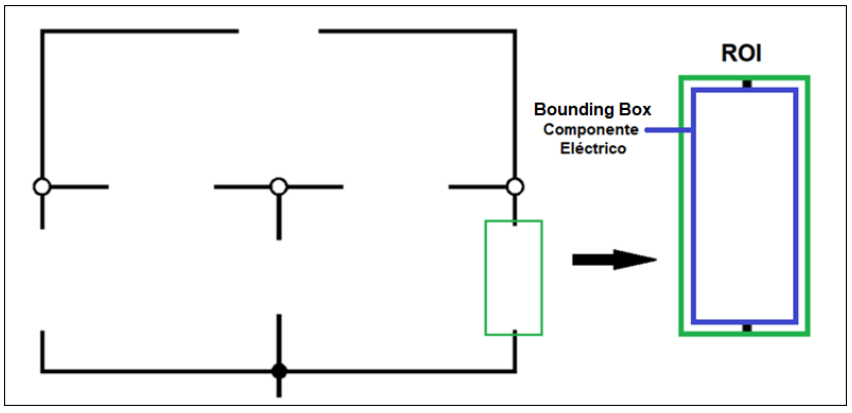


Detección de terminales

Para realizar la detección de los terminales se parte de la imagen mostrada en la Figura 34. Debido a que se conoce las coordenadas de los *bounding boxes*, se crean rectángulos delimitadores con un aumento de 2 pixeles a cada lado al correspondiente a cada componente eléctrico, obteniendo así pequeñas regiones de interés (ROI), como se observa en la Figura 36.

Figura 36

Creación de regiones de interés



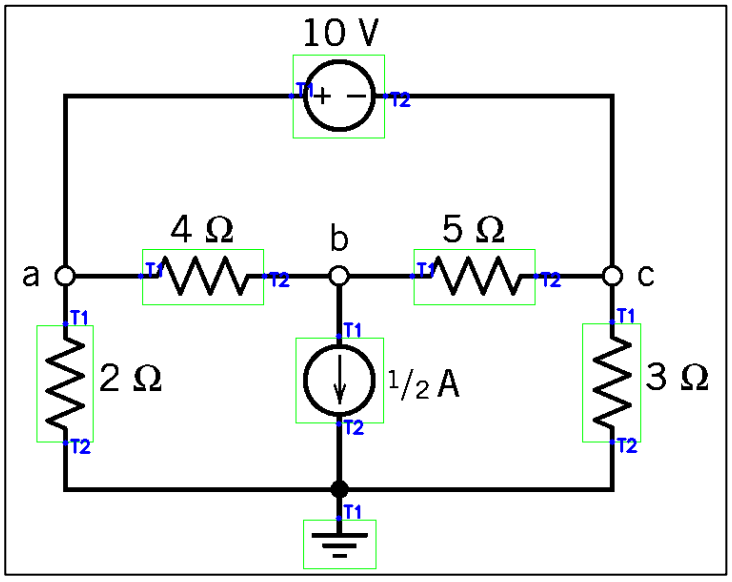
De esta forma, se obtiene pequeñas ROI de acuerdo con el número de componentes eléctricos que se presenten en el circuito. Para la detección de los terminales se aplica el análisis de componentes conexos a cada una de las ROI creadas.

Una vez detectado los componentes conexos de cada región de interés, se extrae las etiquetas y las coordenadas de los centros de cada componente detectado, ya que cada etiqueta corresponde a un terminal de los componentes eléctricos y las coordenadas del centro su posición la cual es útil para posteriormente realizar la detección de conectividad.

Se opto por usar componentes conexos para la detección de terminales ya que de esta forma la orientación de los elementos no es un factor determinante. En la Figura 37 se observa los terminales detectados en cada uno de los componentes del circuito eléctrico.

Figura 37

Detección de terminales de cada componente eléctrico



Detección de conectividad

Con la información obtenida de la detección de nodos y terminales, se determina la conectividad entre elementos. Sabiendo que la función de componentes conexos devuelve una imagen (matriz) del tamaño de la original donde cada píxel tiene la etiqueta del componente al

que pertenece, es decir, el índice del nodo detectado. Esta imagen actúa como un mapa de referencias que permite determinar si una coordenada especifica pertenece o no a un nodo.

Además, para iniciar el proceso de detección se cuenta con una lista de componentes, cada uno con las coordenadas en pixeles de sus respectivos terminales. Por cada terminal se consulta la etiqueta correspondiente en el mapa de referencia. Si la coordenada se encuentra dentro del área válida de la imagen y su etiqueta coincide con alguna de las etiquetas identificadas como nodos, entonces se considera que dicho terminal está conectado a ese nodo. Caso contrario, si no registra ninguna etiqueta, se está indicando que el terminal no establece ninguna conexión.

De esta manera, se construye una función de mapeo que refleja las conexiones eléctricas del circuito, donde se especifica a qué nodos se conectan los terminales de cada componente. La Figura 38, muestra una lista de resultados donde se almacena el nombre asignado a cada componente eléctrico detectado, su clase, los terminales y nodos al que están conectados.

Figura 38

Lista de conexiones

Nombre	Clase	Terminal	Nodo	Terminal	Nodo
V1	fuente_voltaje_DC	T1	1	T2	2
R1	resistencia	T1	2	T2	0
R2	resistencia	T1	3	T2	2
R3	resistencia	T1	1	T2	3
R4	resistencia	T1	1	T2	0
I1	fuente_corriente	T1	3	T2	0

La asignación de nombres se realiza acorde a lo establecido en la Tabla 5, con el fin de identificar a cada componente y sus nodos de conexión. Para el caso de tener varios componentes de la misma clase, se realiza un programa que detecte cada elemento y más de asignarle el nombre se agrega una etiqueta numérica.

Tabla 5

Asignación de nombres acorde a la clase

Clase	Nombre
Batería	V
Capacitor	C
Fuente de corriente	I
Fuente de voltaje AC	V
Fuente de voltaje DC	V
Inductor	L
Pila	V
Resistencia	R
Tierra	GND

Detección e interpretación de texto

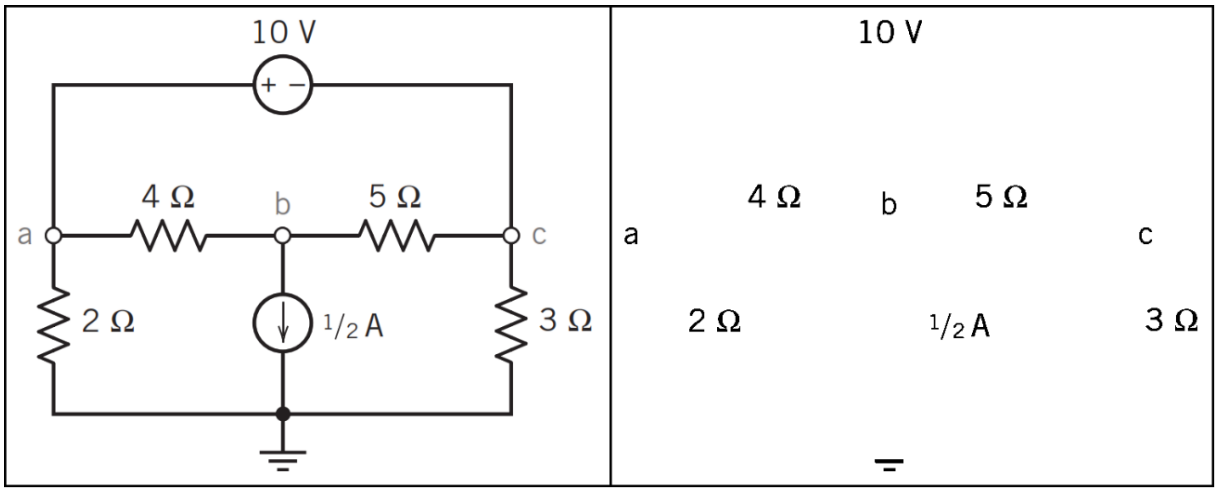
La detección de textos tiene un papel muy importante en este proyecto, ya que el texto de los diagramas de circuitos eléctricos muestra los valores de cada componente, información vital para la generación de Netlist. Para la detección e interpretación de texto, hay varias herramientas de reconocimiento óptico de caracteres (OCR) de código abierto y disponibles para trabajar en el entorno de Python, como es, Tesseract, Easyocr, Padleocr, etc. Previo a la detección de texto, se hizo pruebas de cada una de estas herramientas para identificar la herramienta ideal a usar en este proyecto, siendo “Padlleocr” (PaddlePaddle, 2025) la elegida. Los resultados de las pruebas se observan en el siguiente capítulo. Para hacer uso de esta herramienta es importante instalar las librerías de `PaddlePaddle` y `PaddleOCR` en el entorno virtual de nuestro proyecto.

Para que la detección del texto sea optima, se filtró a la imagen para mostrar únicamente el texto, suprimiendo el diagrama de circuito. Este proceso se realizó usando

componentes conexos, lo que permitió identificar el componente con el área más grande (circuito) y eliminarlo. Tal como se presenta en la Figura 39.

Figura 39

Filtrado para detección de texto



Nota. A la izquierda se muestra la imagen original y a la derecha la imagen filtrada.

Detección de texto con PaddleOCR

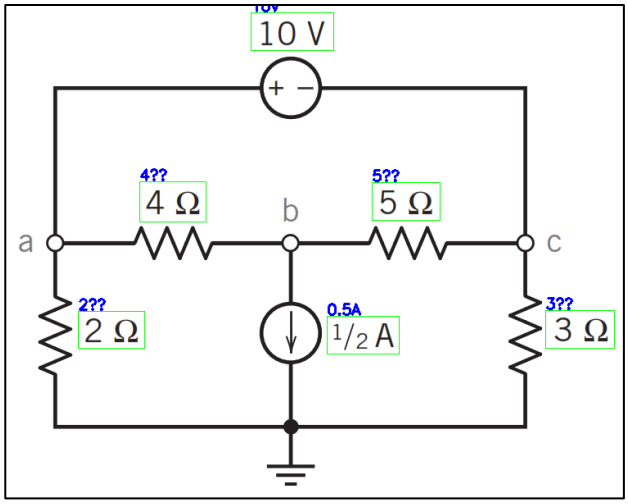
Con la imagen del circuito filtrada, se aplica la función de `PaddleOCR` a la imagen, obteniendo una lista de resultados con la siguiente información:

- Texto reconocido por el modelo OCR.
- Nivel de confianza del reconocimiento de texto en cada región.
- Coordenadas del polígono que delimita al texto detectado.

Esta información permite saber qué texto fue reconocido, qué tan confiable es, y dónde está ubicado dentro de la imagen. De esta forma se crean los *bounding boxes* de los textos, como se observa en la Figura 40.

Figura 40

Detección de texto enmarcado por sus bounding boxes



Enlace texto – componentes eléctricos

Para asociar el texto con su componente eléctrico, se utilizó la distancia euclidiana entre los puntos medios (centros) de cada uno de los *bounding boxes*. Esta distancia es una medida que calcula qué tan lejos o cerca están dos objetos entre sí, la cual se obtiene usando el mismo principio que se aplica para calcular la distancia recta entre dos puntos en un plano.

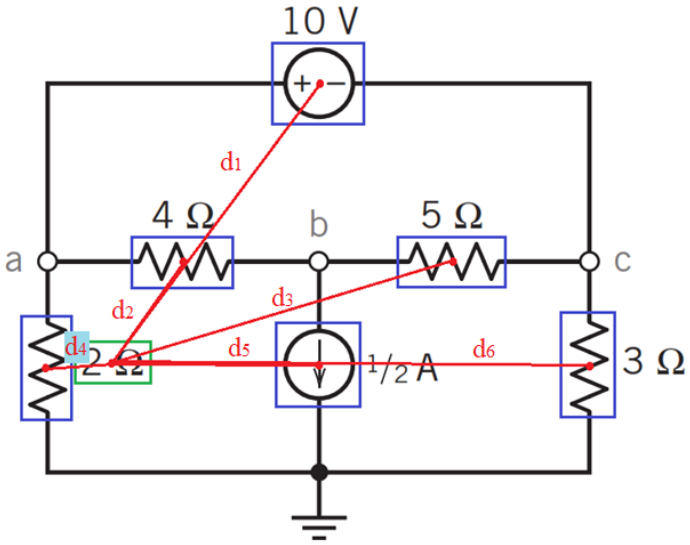
Bajo este principio, el procedimiento parte de con la lista de componentes detectados representados en una matriz, junto con sus centroides obtenidos previamente, y una lista de textos detectados con sus respectivos centros.

En la vinculación de los textos con los componentes eléctricos, se utiliza la distancia d_n euclidiana entre el centroide de cada texto y los centros de los *bounding box*es de los componentes (Figura 41), con el objetivo de asignar cada texto al componente más cercano. Además, esto garantiza que cada texto se asigne solo a un componente, priorizando al más cercano y de esta forma evitar duplicidades.

En caso de que un componente no tenga un texto cercano asignado, se le otorga un valor por defecto de “1”, con el objetivo de continuar con la digitalización del circuito.

Figura 41

Comparación distancias euclidianas entre los centros de cada texto y los componentes



El resultado final es una nueva matriz, estructuralmente igual a la original (Figura 38), pero con un campo adicional que representa el valor asociado a cada componente, tal como se muestra en la Figura 42. Esta asignación es importante porque se obtiene una lista con información detallada del circuito.

Figura 42

Lista de detalle del diagrama de un circuito eléctrico

Nombre	Clase	Terminal	Nodo	Terminal	Nodo	Valor
V1	fuente_voltaje_DC	T1	1	T2	2	10V
R1	resistencia	T1	2	T2	0	3Ω
R2	resistencia	T1	3	T2	2	5Ω
R3	resistencia	T1	1	T2	3	4Ω
R4	resistencia	T1	1	T2	0	2Ω
I1	fuente_corriente	T1	3	T2	0	0.5A

Generación de archivos descriptivos

Con la lista obtenida después de hacer la detección de texto, se tiene la información necesaria para crear un archivo de texto con estos detalles del circuito y el netlist (lista de conexiones).

Un netlist es una representación textual de las conexiones eléctricas entre los componentes de un circuito eléctrico (Franco, 2015). Un netlist contiene líneas de texto donde cada línea representa un componente del circuito. Generalmente incluye:

- Identificador del componente (V1, R1, C1, etc.)
- Nodos de conexión (1, 2)
- Valor o parámetro como resistencia en ohmios, voltaje, corriente, etc.

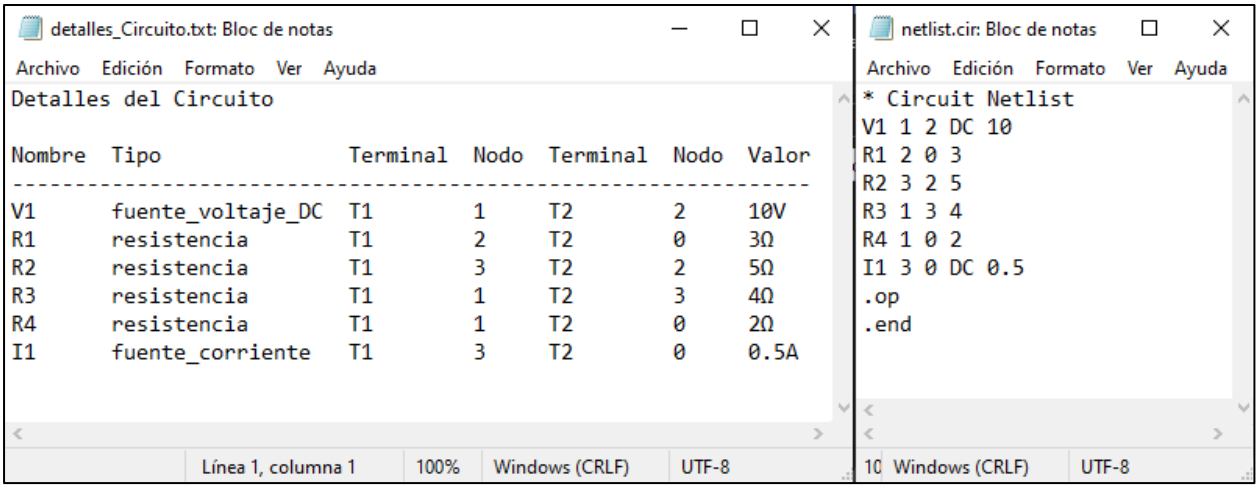
Los detalles del circuito eléctrico se guardan en un archivo de texto con el nombre "detalles_Circuito.txt". Para crear este archivo de texto se utiliza la función *open("detalles_Circuito.txt", "w")* la cual crea un archivo de texto con el nombre mostrado en el primer parámetro. El segundo parámetro es el modo en el que va a operar la función, en este caso `w` es escritura. En este archivo se incluye toda la información mostrada en la Figura 42 en columnas alineadas, incluyendo encabezados y títulos.

Para generar el archivo netlist se utiliza la misma función `open` con la diferencia de que en el primer parámetro el nombre del archivo termina en extensión `.cir`, de esta forma el contenido del netlist incluye el identificador del componente, los nodos de conexión, el valor de cada componente y al final se incluye una línea de texto que indica el tipo de simulación que va a tener el netlist. Para este caso en particular se añade la instrucción `.op` que indica que se debe hacer una simulación de punto de operación, esta simulación calcula las tensiones en todos los nodos y las corrientes que fluyen por cada componente. Y se cierra el netlist con una instrucción `.end` la cual indica el fin del archivo o del bloque de simulación.

De esta forma se obtiene los archivos descriptivos generados de la identificación de los componentes de un diagrama de circuito eléctrico (Figura 43).

Figura 43

Archivos descriptivos del diagrama de un circuito eléctrico



Nota. A la izquierda se muestra el archivo de texto con los detalles del circuito y a la derecha el archivo `.cir` con la netlist del circuito.

Simulación del circuito

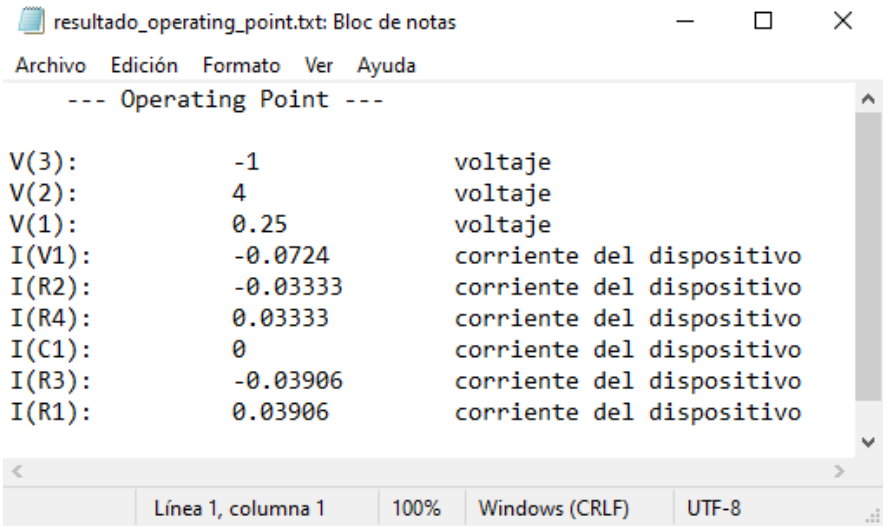
Para simular el archivo netlist se usa la librería `LTspice`, esta es una herramienta que permite leer, analizar y extraer datos de los archivos de simulación generados en LTspice.

Se diseño una función que recibe el archivo netlist, el cual describe el circuito a simular. Posteriormente, se construyó un comando que permita ejecutar LTspice (instalado en la PC) en modo *batch* (sin intervención del usuario). Al realizar esto, se genera automáticamente un archivo de resultados del tipo .raw, el cual contiene los valores de la simulación. Este archivo es analizado usando la librería `LTspice`, lo que facilitó la extracción de las variables registradas durante la simulación.

Una vez extraídos los resultados correspondientes a la simulación del punto de operación, fueron clasificados según su naturaleza (voltaje o corriente). Finalmente, los resultados obtenidos se escribieron en un archivo de texto con el nombre de “resultado_operating_point.txt”, en el cual se registra cada variable con su respectivo valor y tipo como se observa en la Figura 44.

Figura 44

Resultados de la simulación del archivo netlist



Desarrollo de la aplicación

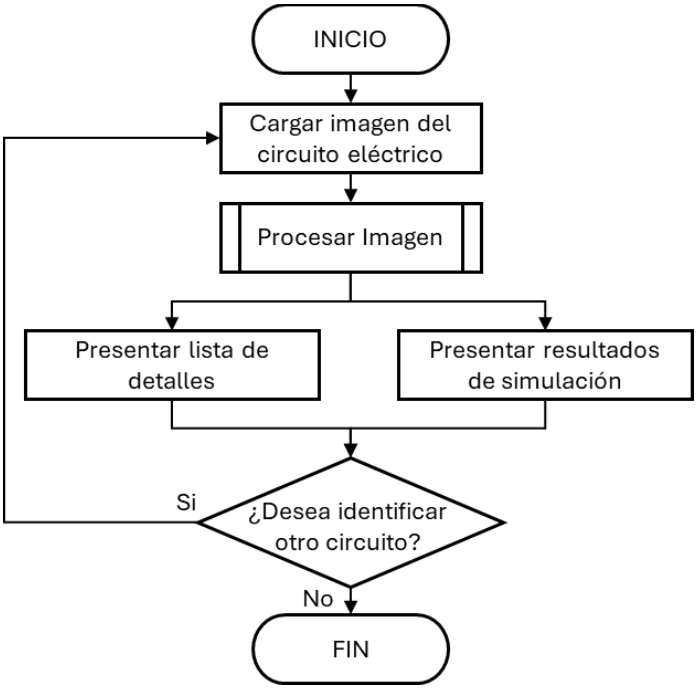
Una vez desarrollado todas las etapas necesarias para la digitalización de diagramas de un circuito eléctrico es necesario integrarlas dentro de una aplicación que las ejecute.

Para el desarrollo de la aplicación se utilizó “Tkinter”, que es una librería estándar e incluida de forma predeterminada en Python, utilizada para el desarrollo de interfaces graficas. Esta librería permite crear ventanas interactivas haciendo uso de diverso componentes visuales como ventanas emergentes, marcos, tablas, botones, etiquetas de texto, etc.

De esta forma, la aplicación se diseñó siguiendo la lógica mostrada en el diagrama de flujo de la Figura 45, lo que permitió implementar de forma estructurada las funcionalidades de la aplicación y a la vez facilitó el desarrollo de botones en un panel de operación para cumplir con estas funcionalidades.

Figura 45

Diagrama de flujo de la funcionalidad de la aplicación



La aplicación fue diseñada para que en primera instancia permita cargar el diagrama de un circuito eléctrico, mismo que será mostrado en la interfaz de la aplicación.

Como siguiente paso se ejecuta un subproceso el cual realiza todo el procesamiento de la imagen lo que incluye la detección de los componentes eléctricos, detección de nodos y conexiones, detección de texto y generación de archivos descriptivos.

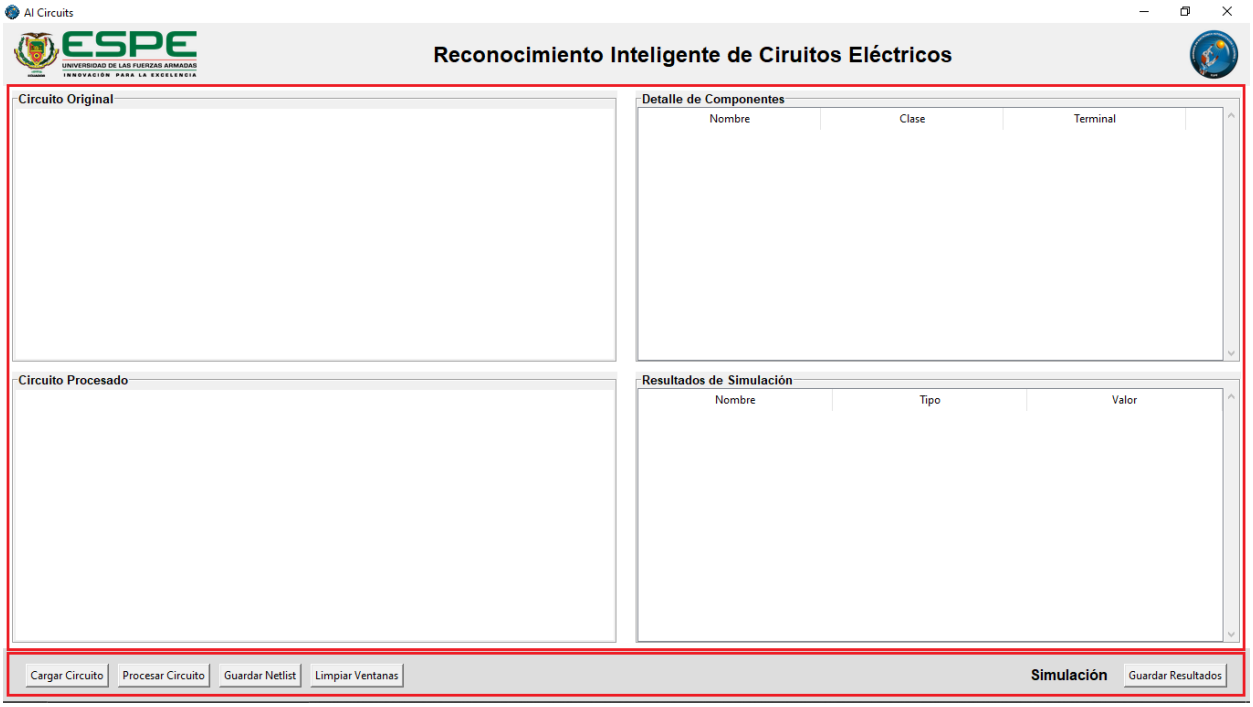
Una vez realizado este procesamiento, se presenta los resultados en la interfaz tanto la información detallada del circuito como los resultados de la simulación. Después se da la opción de cargar otro diagrama o simplemente cerrar la aplicación.

En base a lo expuesto se diseñó la interfaz como se observa en la Figura 46, la misma que presenta dos áreas de usabilidad importantes que son:

- Área de visualización
- Panel de operación

Figura 46

Archivos descriptivos del diagrama de un circuito eléctrico



Área de visualización

Esta área se compone de cuatro ventanas que presentan información importante del digitalizar un diagrama de circuito eléctrico. Las ventanas ubicadas en la parte izquierda de la interfaz presentan información visual, mientras que las ventanas de la derecha información textual.

Como se observa en el título de cada ventana mostrado en la Figura 46, el contenido que estas presentan es:

- Circuito original: Diagrama del circuito eléctrico cargado a la interfaz
- Circuito procesado: Diagrama del circuito eléctrico resaltando las detecciones de componentes, terminales, nodos y texto.
- Detalle de componentes: Tabla detallada de las conexiones entre nodos y terminales de cada componente, junto a sus valores nominales.

- Resultados de simulación: Tabla de valores de voltaje de nodos y corrientes de cada componentes del circuito eléctrico.

Panel de operación

El panel de operación cuenta con botones que cumplen con ciertas funciones claves para el uso adecuado de la aplicación mismas que se detallan en la Tabla 6, permitiendo un manejo fácil e intuitivo por parte del usuario.

Tabla 6

Funciones de la aplicación

Función	Descripción
Cargar circuito	Permite seleccionar un diagrama de circuito eléctrico para digitalizar.
Procesar circuito	Ejecuta todas las etapas para la digitalización del circuito eléctrico.
Guardar netlist	Guarda el archivo `netlist.cir` en la ubicación especificada por el usuario.
Limpiar ventanas	Limpia todas las ventanas de la interfaz para poder procesar otro diagrama.
Guardar resultados	Permite guardar los resultados de la simulación en un archivo de texto.

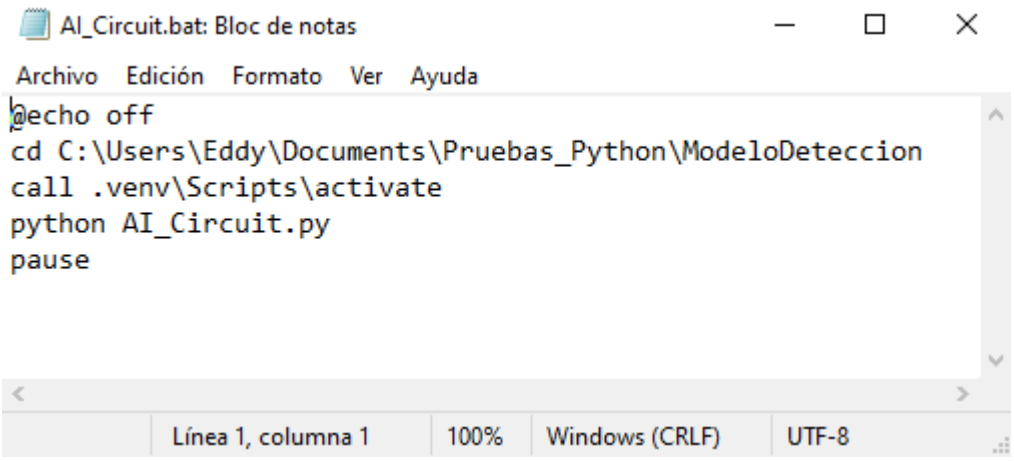
Acceso directo a la aplicación

Para tener acceso directo a la aplicación sin tener que ejecutar el proyecto desde el IDE “PyCharm” o ejecutar el script de la interfaz junto el entorno virtual que contiene a todas las dependencias del proyecto mediante líneas de código desde el símbolo del sistema (CMD) del computador, se creó un icono de acceso directo de la aplicación denominada “AI Circuit”.

Para esto se creó un archivo de extensión .bat, que es un “archivo por lotes” el cual contiene una serie de comandos que se ejecutan de manera secuencial en el CMD de Windows. Los comandos contenidos en el archivo .bat que permite ejecutar nuestra aplicación desde un acceso directo son las que se presentan en la Figura 47.

Figura 47

Creación del archivo .bat



Estas serie de comando permiten activar el entorno virtual de nuestro proyecto y ejecuta desde el CMD es script “AI_Circuit.py”, archivo que corresponde a la aplicación de detección de elementos eléctricos que creamos.

Adicional a esto, se crea un icono de acceso directo del archivo .bat en el escritorio, con el objetivo de que se visualice como una aplicación más dentro del sistema de Windows como se observa en la Figura 48.

Figura 48

Acceso directo de la aplicación en el escritorio de Windows



Capítulo IV

Pruebas y Evaluación de Resultados

En este capítulo se presenta las pruebas y resultados obtenidos del entrenamiento del modelo de detección de componentes, detección de texto y funcionalidad de la aplicación.

Evaluación del modelo de detección

A continuación, se presenta la evaluación del modelo de detección de componentes eléctricos. Para el entrenamiento del modelo de detección se usó una base de datos compuesta por 2286 imágenes la cual estaba dividida en tres conjuntos de datos, donde el primero corresponde a datos de entrenamiento, el segundo a datos de validación y el tercero a datos de prueba, conjuntos que alojan el 85%, 9% y 7% de la base de datos total respectivamente. Siendo los conjuntos de entrenamiento y validación los usados en el entrenamiento del modelo y el conjunto de prueba se utilizó una vez finalizado el entrenamientos para pruebas del modelo, mismas que se presentan más adelante.

Como se detalló en el capítulo de diseño, los para parámetros usados para entrenar el modelo de detección, son los presentados en la Tabla 7.

Tabla 7

Parámetros de entrenamiento del modelo de detección

Función	Descripción
Modelo	Yolov11s
Epochs	50
Imgsz	520
batch	4
device	CPU
workers	0
optimizer	Adam

Función	Descripción
Lr0	0.0005
weight_decay	0.0005
cache	False
val	True

Se considero estos parámetros debido a que el computador utilizado para entrenar el modelo cuenta con características un poco limitadas, mismas que se presentan en la Tabla 8.

Tabla 8

Características del computador

Marca	Sistema Operativo	Procesador	RAM	Tarjeta Gráfica
DELL	Windows 10 Enterprise	Intel Core i5 7200U	16GB	Intel® HD Graphics 620

Evaluación del modelo.

Una vez finalizado el entrenamiento se consideró las siguientes métricas de evaluación, para verificar que tan bien identifica y localiza los objetos el modelo de detección:

- Precisión (Precision)
- Sensibilidad (Recall)
- Promedio de la presión media (mAP)
- Promedio del mAP (mAP50-95)

Los resultados que mostró el entrenamiento son los presentados en la Tabla 9, la cual presenta las métricas de evaluación del modelo en general y de cada clase.

Tabla 9

Métricas de evaluación del modelo de detección

Clase	Precision	Recall	mAP50	mAP50-95
Todas	0.940	0.941	0.957	0.761
Batería	1	0.91	0.967	0.638
Capacitor	0.953	0.985	0.993	0.756
Fuente de corriente	0.966	1	0.980	0.866
Fuente de voltaje AC	0.910	0.839	0.922	0.790
Fuente de voltaje DC	0.977	0.977	0.994	0.886
Inductor	0.951	0.72	0.987	0.787
Pila	0.808	0.905	0.838	0.644
Resistencia	0.995	0.978	0.993	0.800
Tierra	0.898	0.906	0.934	0.686

Nota. Los datos fueron generados automáticamente al momento de finalizar el entrenamiento.

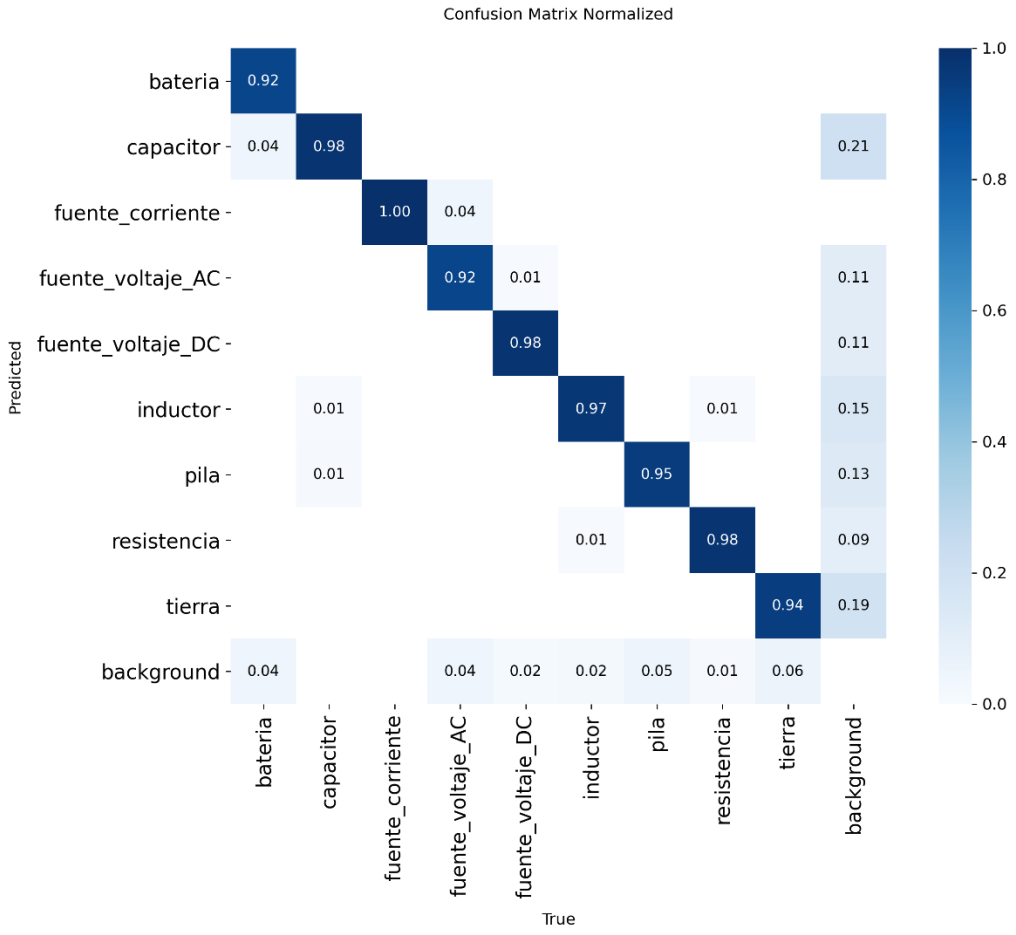
Tomando como referencia el resultado general de todas las clases, se puede apreciar que un 94% de las detecciones ejecutadas por el modelo fueron realizadas de manera correcta lo que se concluye que existe muy pocos falsos positivos. También se indica que el modelo detecto un 94.1% de los objetos reales, lo que deriva en muy pocos falsos negativos.

Visualizando las otras dos métricas, se puede concluir que el modelo detecta los componente eléctricos con gran precisión ya que presenta un valor del 95.7% del mAP50 y también los localiza con precisión ya que el valor de mAP50-95 está por encima del 70%, considerándolo como un modelo de alto rendimiento.

Para corroborar este análisis, en la Figura 49 se presenta una matriz de confusión normalizada donde se puede apreciar una diagonal dominante, la cual nos indica que el modelo predice bien ya que las clases reales coinciden con las predichas.

Figura 49

Archivos descriptivos del diagrama de un circuito eléctrico



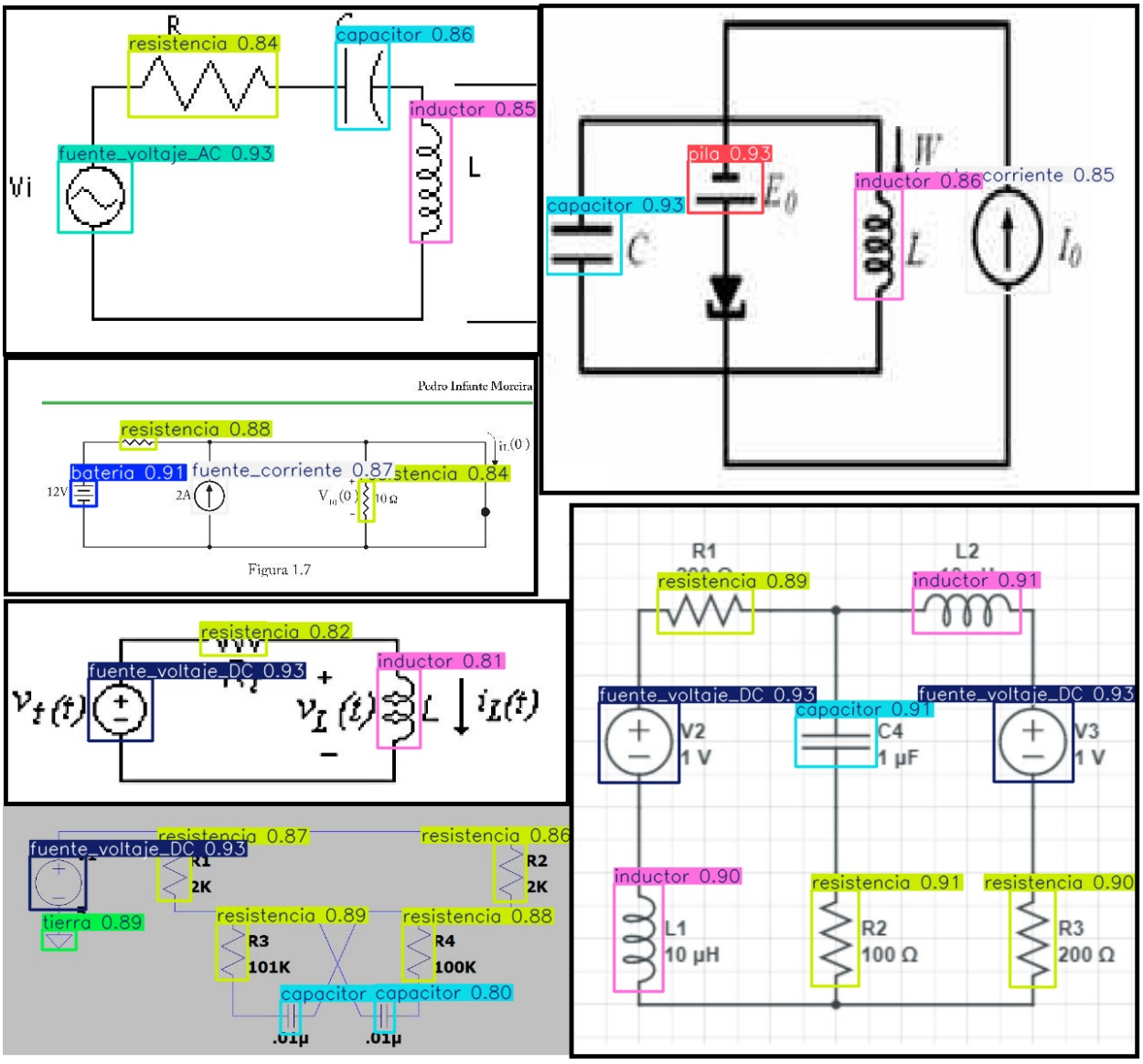
Pruebas de detección

Las pruebas de detección se realizaron con dos grupos de prueba, el primer grupo corresponde al conjunto de prueba (*test*) tomado de base de datos y el segundo grupo a capturas de diagramas de circuitos eléctricos.

El resultado de la detección de componentes aplicado al primer grupo se observa en la Figura 50.

Figura 50

Detección de componentes del conjunto de prueba de la base de datos

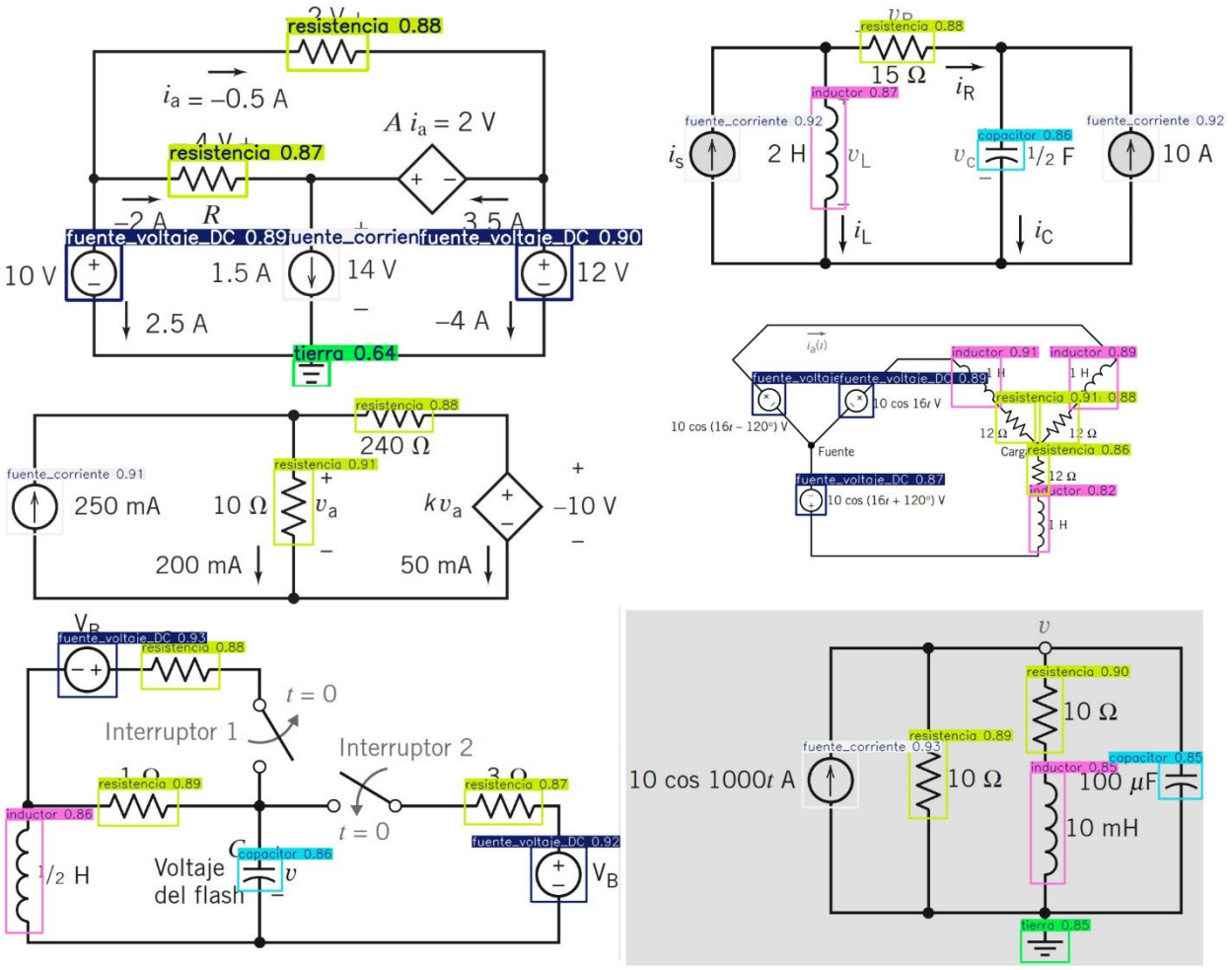


La detección de componentes se realizó a 95 imágenes que son las que conforman el primer grupo de las cuales 4 imágenes presentaron una mala detección, en particular la clase correspondiente a “pila”, que como se presenta en la Tabla 9 es la que menor precisión presenta con respecto a las demás clases. Es importante mencionar que en este conjunto de imágenes se encuentran diagramas con baja resolución y aun así se logró una detección correcta en el 95.8% de las imágenes.

Una muestra de la detección realizada al segundo grupo de imágenes se presenta en la Figura 51.

Figura 51

Detección de componentes al segundo grupo de pruebas



Este grupo estaba conformado por 26 imágenes, en el cual no hubo errores de detección en ninguno de los diagramas. Un factor importante que considerar, son características que presentan las imágenes de diagramas de circuitos eléctricos, ya que mientras mayo resolución presente el circuito, mejor va a ser la detección de los componentes eléctricos. Como es el caso de este grupo de imágenes que fueron extraídas de documentos en formato pdf y presentaban una resolución mayor a 640 pixeles.

Evaluación de la detección de texto

Como se explica en el capítulo anterior, se consideró 3 herramientas de reconocimiento óptico de caracteres (OCR) para la detección de textos, que son:

- TesseractOCR
- EasyOCR
- PaddleOCR

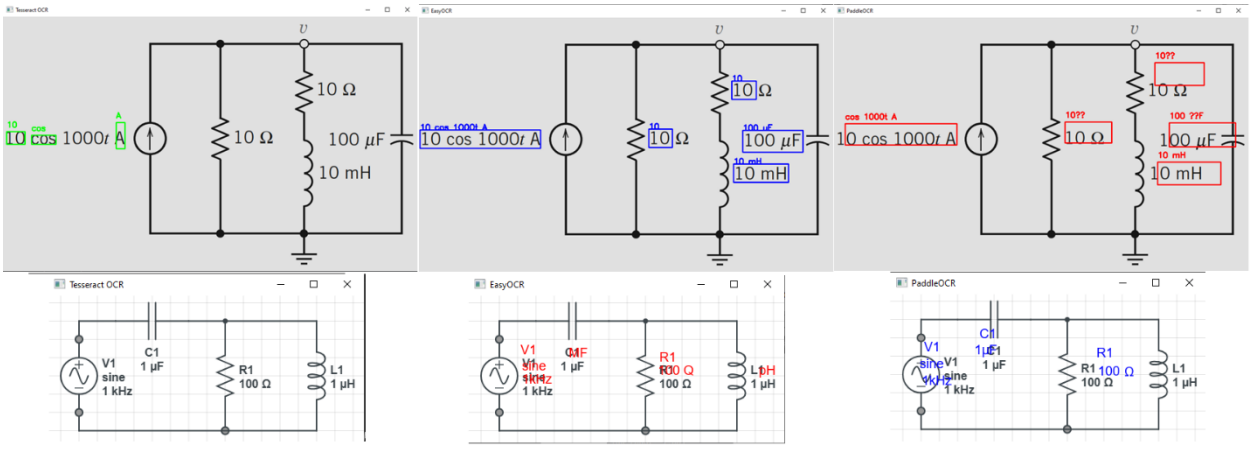
Estas tres herramientas cuentan con librerías compatibles con Python lo que las hace una buena opción para usarlas en este proyecto. Sin embargo, presentan diferencias entre sí y la principal diferencia es la precisión con la que detectan los textos en las imágenes.

Por esta razón se realiza una prueba con las tres herramientas, todas comparando una misma imagen con el objetivo de identificar que herramienta es más precisa y capaz de reconocer los textos que se encuentran en los diagramas de circuitos eléctricos.

Para la prueba se consideró dos imágenes, mismas que se muestran en la Figura 52 tras aplicar las herramienta de detección de texto a cada una.

Figura 52

Detección de textos con herramientas OCR



Nota. A la izquierda se muestra las detecciones con TesseractOCR, en el centro EasyOCR y a la derecha PaddleOCR.

Al realizar un análisis visual de las detecciones realizadas con cada herramienta OCR, se puede observar que existe una diferencia considerable en la precisión al momento de realizar las detecciones de texto Dejando a notar que la más deficiente es TesseractOCR, ya que en la primera imagen de prueba apenas reconoce pocos caracteres mientras que en la segunda no logra detectar ninguno.

Las otras dos herramientas muestras mayor precisión en la detección de textos, ya que como se observa en la Figura 52, ambas logran reconocer el texto presente con las siguientes observaciones:

- EasyOCR no detecta símbolos matemáticos como es el caso del símbolo de ohmios (Ω), además interpreta valores de “0” como “O” o viceversa y también valores de “1” como “l”. Sin embargo, la ventaja que posee, es que presenta una velocidad de procesamiento muy buena la cual no supera los 10 segundos al momento de realizar una detección.
- PaddleOCR detecta textos más complejos incluyendo símbolos matemáticos y no confunde caracteres lo que lo vuelve más preciso. Su velocidad de procesamiento es buena pero menor comparándola con EasyOCR, ya que esta herramienta varia su tiempo de procesamiento entre 15 a 30 segundos dependiendo la capacidad de procesamiento que presente el computador.

Con esta comparación el detector de caracteres de ser utilizo es PaddleOCR ya que presenta mayor precisión en la detección de texto sin confundir caracteres y considerando que el tiempo de procesamiento no es excesivo ni muy significante comparándolo con el tiempo que lleva el transcribir un diagrama de un circuito eléctrico a un software de simulación.

Evaluación de la funcionalidad de la aplicación

Una vez culminado el diseño de la aplicación, se verifico que cada ventana, botón o demás funciones implementadas en la interfaz cumplan con el objetivo para el cual fue diseñado. Además, se realiza una comparación de los resultados generados en la aplicación

como la netlist y la simulación, con los generados por otro software con motor “spice” como LTspice o PSpice.

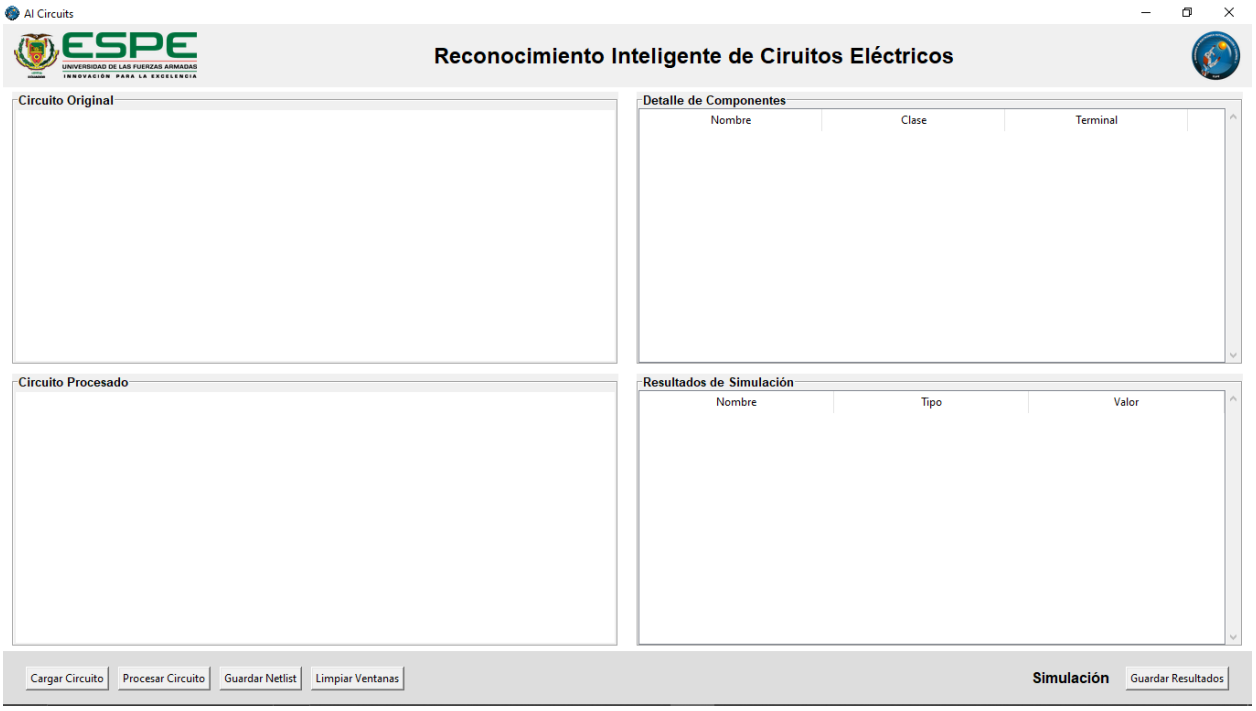
Pruebas funcionales de la interfaz gráfica

Para para verificar la funcionalidad de la interfaz, se generó un tercer grupo de pruebas con diagramas de circuitos eléctricos que presenten valores válidos para simulación, estos diagramas fueron capturados de documentos en formato pdf y otros fueron diseñados en una plataforma en línea desarrollada para el diseño y simulación de circuitos eléctricos (CircuitLab, Inc., 2025).

Al ejecutar la aplicación, se inicializa con la ventana presentada en la Figura 53.

Figura 53

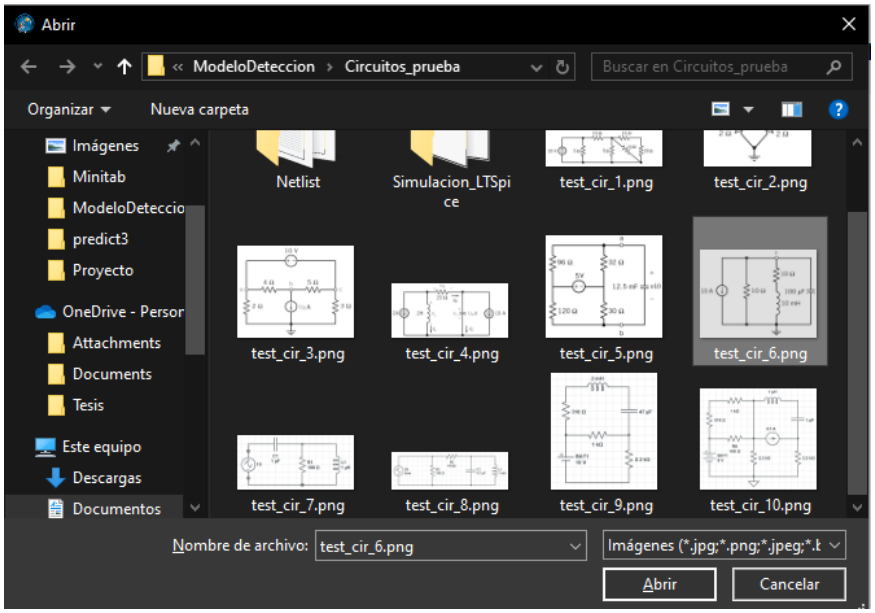
Vista general de la aplicación de reconocimiento inteligente de circuitos eléctricos



Como siguiente paso se presiona el botón “Cargar Circuito” en cual abre una ventana flotante (Figura 54) que permite al usuario seleccionar el diagrama de un circuito eléctrico que este alojado en cualquier ruta del computador y que este en formatos de imagen (.jpg, .png y .jpeg).

Figura 54

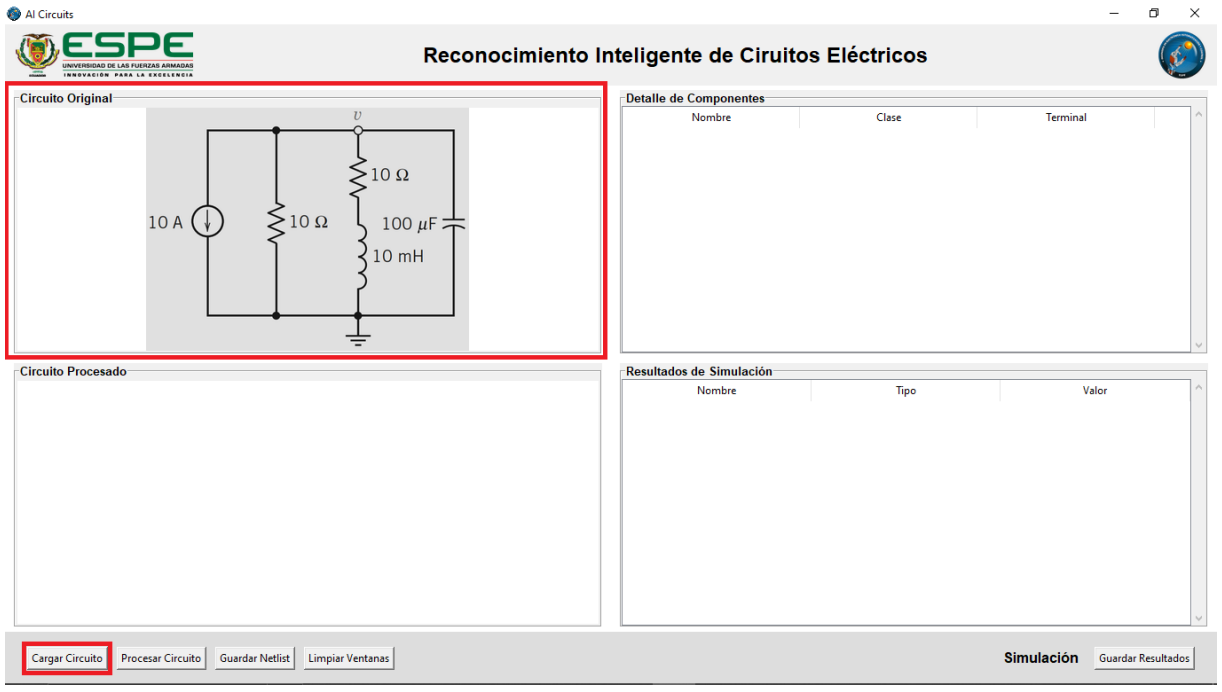
Selección del diagrama de circuito eléctrico



Al momento de seleccionar la imagen del circuito y dar clic en “Abrir”, el circuito se carga en la primera ventana (superior-izquierda) de la interfaz (Figura 55).

Figura 55

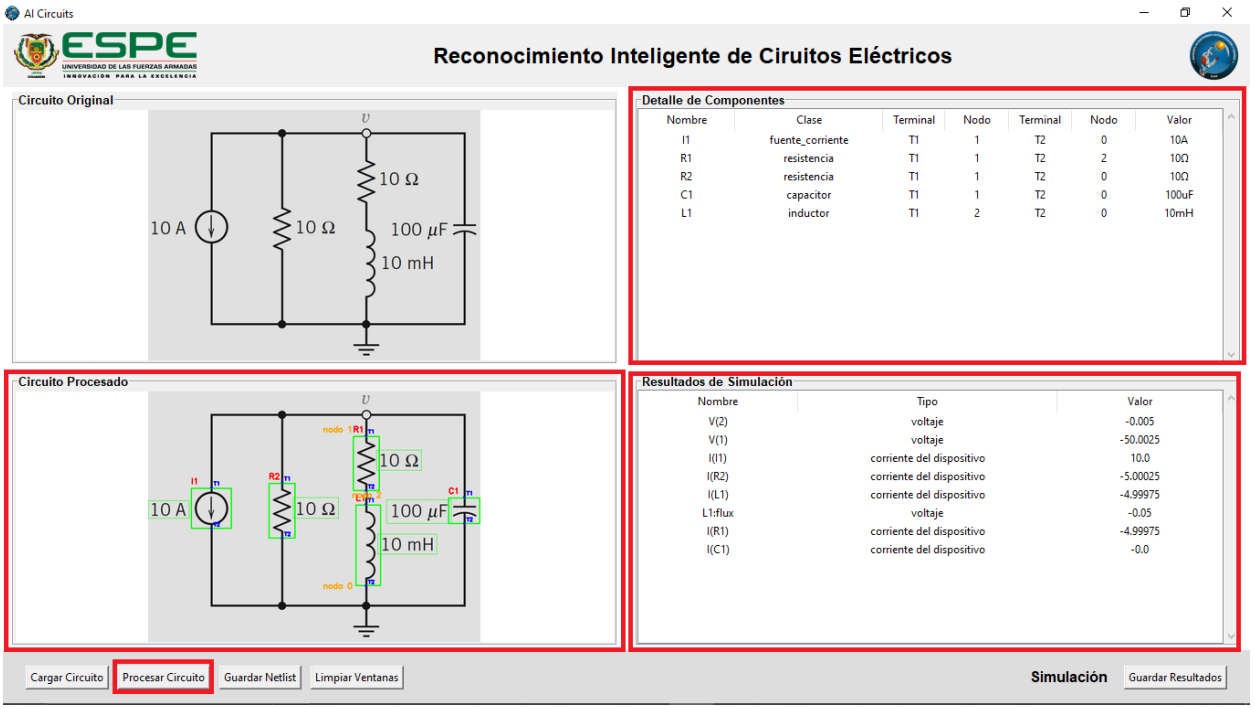
Diagrama del circuito eléctrico cargado en la interfaz



Una vez cargado el circuito se presiona el botón “Procesar Circuito” lo que inicia el procesamiento del diagrama de circuito eléctrico ejecutando las etapas de detección de componentes, detección de nodos y conexiones, detección de texto, la creación de archivos descriptivos y la simulación del circuito, mostrando en la ventana inferior izquierda el diagrama del circuito marcando con cada una de las diferentes detecciones (componente eléctrico, terminales, nodos y texto). En las ventanas de la derecha se muestra detallado la información del circuito y los resultados de la simulación, tal como se muestra en la Figura 56.

Figura 56

Resultados del procesamiento del diagrama de circuito eléctrico



Una vez realizado el procesamiento del diagrama del circuito, el usuario tiene las siguientes opciones:

- Guardar el archivo netlist generado con extensión `.cir`.
- Guardar los resultados de la simulación (voltajes de nodos y corrientes de cada componente) en un archivo de texto.
- Limpiar todas las ventas para cargar un nuevo diagrama.

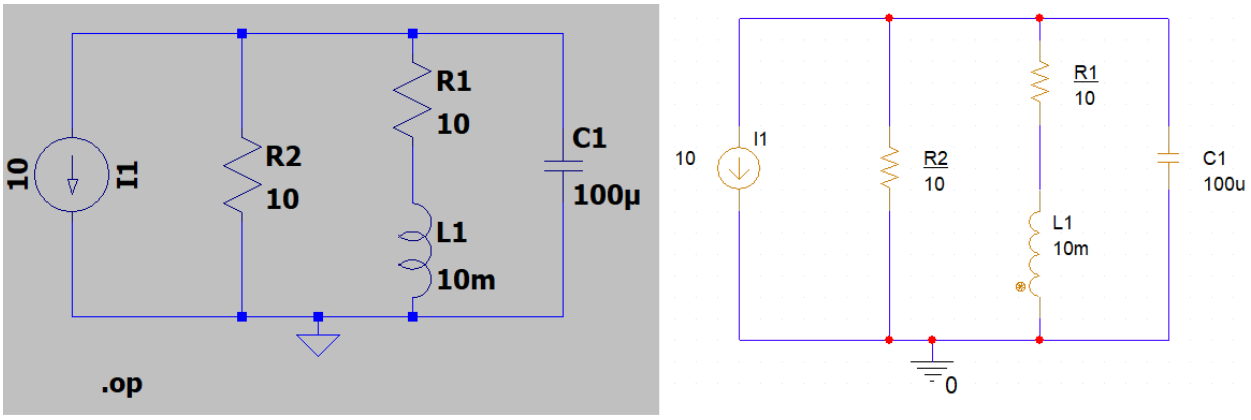
Simulación Netlist en Softwares con motor Spice

Para verificar la efectividad de la digitalización de los diagramas de un circuito eléctrico y la creación y simulación de los archivos netlist. Se compara con los resultados obtenidos en simuladores de referencia.

En primera instancia se compara el netlist creado por nuestra aplicación con la generada por el software LTspice y PSpice. En la Figura 57, se muestra los circuitos implementados en estos softwares.

Figura 57

Diagrama del circuito eléctrico de prueba implementado en dos softwares de simulación

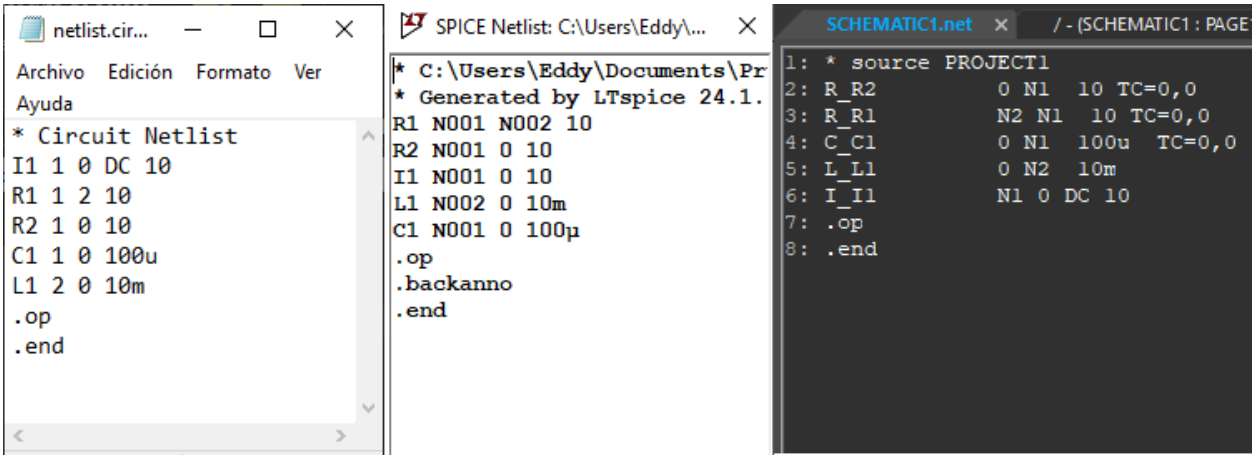


Nota. A la izquierda se muestra el circuito diseñado en LTspice y a la derecha en PSpice.

Una vez implementado los circuitos en estos softwares de simulación, se exporta los archivos netlist para realizar la comparación con el creado por nuestra aplicación.

Figura 58

Comparación de netlist generada por softwares de simulación



Nota. A la izquierda se muestra el netlist creado por nuestra aplicación, en el centro el generado por LTspice y a la derecha el generado por PSpice.

Como se puede observar en la Figura 58, el archivo netlist creado por nuestra aplicación mantiene la misma estructura que los archivos generados en los softwares de simulación lo que lo hace apto para que sea reconocido y permita su simulación en estos softwares.

Como último punto, se comparó los resultados obtenidos de la simulación, con los simulados en el software PSpice. No se compara con la simulación en LTspice, porque los resultados simulados en nuestra aplicación son obtenidos de este software directamente.

A continuación, la Tabla 10 se presenta la comparación entre los resultados obtenidos de las simulaciones.

Es importante mencionar que las simulaciones en softwares con motor “spice” solo generan voltajes de nodos y corrientes de componentes activos (como fuentes, amplificadores, etc) cuando se ejecuta una simulación del tipo punto de operación (.op), a excepción de LTspice que si genera las corrientes a través de componentes pasivos como resistencias, capacitores e inductores.

Tabla 10

Comparación de resultados de la simulación del netlist

Índice	Tipo	Aplicación (LTspice)	PSpice
V(1)	Voltaje	-50.0025	-50.0000
V(2)	Voltaje	-0.005	0.0000
I(I1)	Corriente	10.0	10.0000
I(R1)	Corriente	-4.99975	-
I(R2)	Corriente	-5.00025	-
I(L1)	Corriente	-4.99975	-
I(C1)	Corriente	-0.0	-

Como se muestra en la Tabla 10, los datos que se pueden comparar son los voltajes de nodos, de esta comparación se puede observar que son valores muy cercanos manteniendo una diferencia del 0.005% concluyendo en que hay concordancia entre los dos resultados ya que, en el contexto de las simulaciones eléctricas, esta variación acostumbra a ser despreciable.

Capítulo V

Conclusiones y Recomendaciones

Conclusiones

En este trabajo se creó una aplicación que reconoce elementos del diagrama de un circuito eléctrico básico y sus interconexiones utilizando inteligencia artificial y con esto genera archivos descriptivos con información detallada del circuito al igual que resultados de la simulación de este.

Se seleccionó la arquitectura YOLO para la detección de elementos de un circuito eléctrico, ya que ofrece una buena combinación entre velocidad de predicción, precisión, costo computacional y facilidad de implementación frente a otras arquitecturas. Lo que permitió realizar detecciones de manera eficaz de los elementos, incluyendo imágenes de diagramas que presentaban presencia visual de ruido o poca resolución.

La base de datos seleccionada para el entrenamiento del modelo de detección fue una adaptación a la base de datos creada en un trabajo similar, de la cual se extrajo únicamente imágenes que contengan diagramas eléctricos basados en el estándar “IEEE Standard 315-1975”, lo que permitió tener estilos normalizados de gráficos y clases de símbolos. Lo que resulto crucial para tener un entrenamiento solido del modelo.

El algoritmo de detección fue implementado de manera exitosa, ya que logra reconocer y ubicar resistencias, fuentes, capacitores y demás con gran precisión porque presenta un valor del 95.7% del mAP50 y también los localiza de forma acertada ya que el valor de mAP50-95 es de 76.1%, lo que al está por encima del 70% se lo puede considerar un modelo de alto rendimiento.

El algoritmo de reconocimiento de conexiones se implementó utilizando procesamiento digital de imágenes y principalmente el principio de componentes conexos logrando detectar de forma automática las conexiones entre los elementos del diagrama de un circuito eléctrico.

Con los resultados de las detecciones de componentes eléctricos, sus interconexiones y texto, se generó de forma exitosa los archivos descriptivos del diagrama de un circuito. Un archivo de texto con información detallada del circuito y un archivo netlist que incluye la lista de componentes y parámetros fundamentales. Estos archivos posibilitaron la simulación del circuito en softwares de simulación como LTspice, pudiendo así verificar la operatividad eficaz de la aplicación creada.

Recomendaciones y Trabajos Futuros

Para realizar el entrenamiento del modelo de detección se recomienda hacerlo en una PC que cuente con mayores recursos computaciones como el disponer de una GPU y tarjeta de procesamiento gráfico, al igual que mayor capacidad de memoria RAM, debido a que esto ayudaría a disminuir tiempos de entrenamiento y generar un entrenamiento más eficiente.

Aumentar el volumen de la base de datos incluyendo imágenes generadas de forma sintética haciendo uso de herramientas como RoboFlow, ya que de esta forma se puede potenciar aún más la expansión del modelo frente a nuevos diagramas. Además de que esta herramienta cuenta con opciones de etiquetado de datos automático lo que permite etiquetar grandes volúmenes de datos y reducir significativamente los tiempos de preparación de la base de datos.

Se recomienda incorporar una salida del archivo netlist en formatos (`.cir,.asc,.xml`), para favorecer la interoperabilidad del netlist con diverso softwares de simulación. Además, de incluir en la aplicación un menú de opciones que permita al usuario elegir entre diferentes tipos de simulación que se desee ejecutar con el netlist y no únicamente como “punto de operación” (tipo de simulación actual).

Como trabajos futuros se propone probar otros tipos de arquitecturas de inteligencia artificial para detección de objetos, como puede ser SSD (*Single Shot MultiBox Detector*). Esta alternativa presenta características de desempeño similares a YOLO y permitiría migrar este proyecto a un aplicativo móvil.

Aumentar la base de datos, para lograr una mayor precisión en la detección de objetos del modelo entrenado. Además, incluir elementos como diodos, transistores o amplificadores operacionales que permitan a la aplicación procesar diagramas de circuitos más complejos.

Utilizando los resultados obtenidos de este proyecto puede servir de base para crear aplicaciones enfocadas en la inclusión digital, integrando tecnologías de accesibilidad para personas con discapacidad visual o con dificultades en el aprendizaje, permitiendo el acceso a conocimientos técnicos a un público más amplio.

Referencias

Lin, T.-Y., Goyal, P., Girshick, R., He, K., & Dollár, P. (2017). Focal Loss for Dense Object Detection. *IEEE International Conference on Computer Vision.* Venecia, Italia. https://doi.org/10.1109/ICCV.2017.324

Abadi, M., Barham, P., Chen, J., Chen, Z., Davis, A., Dean, J., . . . Zheng, X. (2016). *TensorFlow: A System for Large-Scale Machine Learning*. Retrieved from https://www.usenix.org/conference/osdi16/technical-sessions/presentation/abadi

Abiodun, O. I., Kiru, M. U., Jantan, A., Omolara, A. E., Dada, K. V., Umar, A. M., . . . Gana, U. (2019). Comprehensive Review of Artificial Neural Network Applications to Pattern Recognition. *IEEE Access, 7*, 158820-158846. https://doi.org/10.1109/access.2019.2945545

Agatonovic-Kustrin, S., & Beresford, R. (2000). Basic concepts of artificial neural network (ANN) modeling and its application in pharmaceutical research. *Journal of Pharmaceutical and Biomedical Analysis, 22*(5), 717–727. https://doi.org/10.1016/s0731-7085(99)00272-1

Alvarado Moya, J. P. (2012). *Procesamiento y Análisis de Imágenes Digitales.* Instituto Tecnológico de Costa Rica.

Bhanbhro, H., Hooi, Y. K., Kusakunniran, W., & Amur, Z. H. (2023). A Symbol Recognition System for Single-Line Diagrams Developed Using a Deep-Learning Approach. *Applied Sciences*. https://doi.org/10.3390/app13158816

Bohara, B., & Krishnamoorthy, H. S. (2024). Deep Learning-Based Framework for Power Converter Circuit Identification and Analysis. *IEEE Access, 12*, 115356-115369. https://doi.org/10.1109/access.2024.3444702

Bradski, G., & Kaebler, A. (2008). *Learning OpenCV.* O'Reilly Media, Inc.

Carion, N., Massa, F., Synnaeve, G., Usunier, N., Kirillov, A., & Zagoruyko, S. (2020). End-to-End Object Detection with Transformers. *Lecture notes in computer science.* https://doi.org/10.1007/978-3-030-58452-8_13

CircuitLab, Inc. (2025). *CircuitLab*. Retrieved from Online circuit simulator & schematic editor - CircuitLab: https://www.circuitlab.com/

Dorf, R. C., & Svoboda, J. A. (2011). *Circuitos Eléctricos, Octava Edición.* Alfaomega Grupo Editor, S.A. de C.V., México.

Elizondo, J. E., & Maestre, L. P. (2005). *Fundamentos de procesamiento de imágenes.* Universidad Autónoma de Baja California, México.

Franco, S. (2015). *Design with operational amplifiers and analog integrated circuits (4th ed.).* McGraw-Hill Education.

Gonzales, R. C., & Woods, R. E. (2018). *Digital Image Processing.* Pearson Education Limited.

Institute of Electrical and Electronics Engineers. (1975). *IEEE standard graphic symbols for electrical and electronics diagrams.* IEEE.

Jain, A. (2023, marzo 27). *Comparing OCR Methods: Paddle OCR vs Tesseract OCR vs EasyOCR vs KerasOCR.* Retrieved from Medium: https://medium.com/%40aayushjain_69695/comparing-ocr-methods-paddle-ocr-vs-tesseract-ocr-vs-easyocr-vs-kerasocr-bbaed1ce9d5b

Jamieson, L., Elyan, E., & Moreno García, C. F. (2024). FewShot Symbol Detection in Engineering Drawings. *Applied Artificial Intelligence, 38*. https://doi.org/10.1080/08839514.2024.2406712

JetBrains. (2025). *PyCharm*. Retrieved from Getting started | PyCharm: https://www.jetbrains.com/help/pycharm/getting-started.html

Kellenberger, B., Marcos, D., & Tuia, D. (2018). Detecting mammals in UAV images: Best practices to address a substantially imbalanced dataset with deep learning. *Remote Sensing of Environment, 216*, 139-153. https://doi.org/10.1016/j.rse.2018.06.028

Li, C., Li, L., Jiang, H., Weng, K., Geng, Y., Li, L., . . . Wei, X. (2022). YOLOv6: A Single-Stage Object Detection Framework for Industrial Applications. *arXiv (Cornell University)*. https://doi.org/10.48550/arxiv.2209.02976

Liu, C. (2024). A Review of Digital Image Processing Techniques and Future Prospects. *International Journal of Computer Science and Information Technology, 4*(3). https://doi.org/10.62051/ijcsit.v4n3.22

Liu, W., Anguelov, D., Erhan, D., Szegedy, C., Reed, S., Fu, C.-Y., & Berg, A. C. (2016). SSD: Single Shot MultiBox Detector. *Lecture notes in computer science.* https://doi.org/10.1007/978-3-319-46448-0_2

Livingstone, D. J. (2011). *Artificial neural networks Methods and Applications.* Humana Press. https://doi.org/https://doi.org/10.1007/978-1-60327-101-1

Lutz, M. (2013). *Learning Python, 5th Edition.* O'Reilly Media, Inc.

Matich, D. J. (2001). *Redes Neuronales: Conceptos Básicos y Aplicaciones.* Universidad Tecnológica Nacional, México.

Muñoz Chuquimarca, A. G. (2021). *Reconocimiento de los elementos del diagrama de un circuito eléctrico básico mediante. Tesis Ing.* Universidad de las Fuerzas Armadas ESPE.

PaddlePaddle. (2025). *GitHub*. Retrieved from PaddlePaddle/PaddleOCR: https://github.com/PaddlePaddle/PaddleOCR

Papadopoulos, T., Kosmas, I., & Michalakelis, C. (2021). Artificial Neural Networks: Basic Methods and Applications in Finance, Management and Decision Making, a Roadmap. *Novel Research in Sciences, 6*(5). https://doi.org/10.31031/NRS.2021.06.000648

Pons, C., Pérez, G., & Baum, G. (2022). La Nueva Inteligencia Artificial Conceptos Básicos y Aplicaciones. *Universidad Nacional de La Plata - Revista Química*(372).

Pulli, K., Baksheev, A., Kornyakov, K., & Eruhimov, V. (2012). Real-time computer vision with OpenCV. *Communications of the ACM, 55*(6), 61-69. https://doi.org/10.1145/2184319.2184337

Python Software Foundation. (n.d.). *Python*. Retrieved from Python documentation: https://docs.python.org/3.12/

Rachala, R. R., & Panicker, M. R. (2022). Hand-Drawn Electrical Circuit Recognition Using Object Detection and Node Recognition. *SN Computer Science, 3*(3). https://doi.org/10.1007/s42979-022-01159-0

Reddit, Inc. (2021). *Reddit.* Retrieved from r/MLQuestions: https://www.reddit.com/r/MLQuestions/comments/obbibq/how_many_images_per_class_is_ideal_for_object/

Redmon, J., Divvala, S., Girshick, R., & Farhadi, A. (2016). You Only Look Once: Unified, Real-Time Object Detection. *IEEE Conference on Computer Vision and Pattern Recognition.* https://doi.org/10.1109/CVPR.2016.91

Ren, S., He, K., Girshick, R., & Sun, J. (2016). Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks. *IEEE Transactions on Pattern Analysis and Machine Intelligence, 39*(6), 1137-1149. https://doi.org/10.1109/tpami.2016.2577031

Rezaeva, M. A., & Semendyaev, R. Y. (2021). Development and Application of Convolutional Neural Network for the Recognition of Objects in the Scheme of Electric Grid. *Journal of Physics Conference Series, 2096*. https://doi.org/10.1088/1742-6596/2096/1/012020

Rivera, E. (2005). Introducción a las Redes Neuronales Artificiales. *Universidad Don Bosco - Revista Científica, 6*, 59-70.

Roboflow, Inc. (2025). *Roboflow*. Retrieved from Roboflow: Computer vision tools for developers and enterprises: https://roboflow.com/

Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., . . . Fei-Fei, L. (2015). ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision, 115*(3), 211-252. https://doi.org/10.1007/s11263-015-0816-y

Sadiku, M., & Alexander, C. K. (2006). *Fundamentos de Circuitos eléctricos.* McGraw-Hill Interamericana.

The PyTorch Foundation. (n.d.). *PyTorch*. Retrieved from PyTorch documentation: https://docs.pytorch.org/docs/stable/index.html#pytorch-documentation

Ultralytics Inc. (2025). *Ultralytics*. Retrieved from Ultralytics YOLO Docs: https://docs.ultralytics.com/es/

Zhao, Z.-Q., Zheng, P., Xu, S.-T., & Wu, X. (2019). Object Detection with Deep Learning: A review. *IEEE Transactions on Neural Networks and Learning Systems, 30*(11), 3212-3232. https://doi.org/10.1109/tnnls.2018.2876865

Apéndices