Procesamiento de imágenes usando inteligencia artificial para el desarrollo de aplicaciones industriales

Trabajo de grado para optar al título de ingeniero en mecatrónica

Trabajo de Grado

Autor:
Sergio Luis Beleño Díaz

Tutores:
Ph.D César Augusto Peña Cortés
Ph.D Carol Viviana Martínez Luna

Junio 2020

Procesamiento de imágenes usando inteligencia artificial para el desarrollo de aplicaciones industriales

Trabajo de grado ingeniería mecatrónica

Autor
Sergio Luis Beleño Díaz

Modalidad
Pasantía de investigación

Tutores
Ph.D César Augusto Peña Cortés
Universidad de Pamplona
Ph.D Carol Viviana Martínez Luna
Pontificia Universidad Javeriana

Universidad de Pamplona

Pamplona, Junio 2020

Preámbulo

En el presente documento se plantea el desarrollo de algoritmos de procesamiento de imágenes usando inteligencia artificial, enfocado en redes neuronales artificiales convolucionales o también conocidas como *Convolutional Neural Networks* (CNNs), para la detección de objetos o patrones de interés, en este caso aplicado a la detección en el entorno industrial y redes eléctricas de alta tensión, para esto se emplearán algoritmos prediseñados con librerías de programación como tensorflow y keras, estos algoritmos publicados de manera abierta con licencias MIT.

Uno de los principales problemas que tienen los algoritmos tradicionales de detección de objetos y patrones es que demoran relativamente mucho tiempo para ejecutar en tiempo real y no logran ser tan robustos con respecto a la detección de múltiples clases de objetos a detectar, logrando identificar apenas un cierto número limitado de objetos, por otro lado utilizando las redes neuronales convolucionales como en el algoritmo *You Only Look Once* (YOLO) se han logrado detectar hasta 9000 clases de objetos diferentes y predecir su ubicación espacial en el entorno en el que se encuentra (Redmon y Farhadi, 2017).

Agradecimientos

Quisiera agradecer al Semillero de Investigación de Automatización, Robótica y Control (SIARC) de la Universidad de Pamplona y al Semillero de Investigación de Sistemas Inteligentes, Robótica y Percepción (SIRP) de la Pontificia Universidad Javeriana por brindarme con tanta hospitalidad todo el apoyo necesario para la realización del presente proyecto de grado.

Agradecimientos a mi director de tesis, el profesor César Augusto Peña Cortés por estar siempre disponible para solucionar cualquier duda en todo el proceso de desarrollo del proyecto para conseguir los mejores resultados posibles.

Agradecer al equipo de trabajo en el Centro Tecnológico de Automatización Industrial (CTAI) de la Pontificia Universidad Javeriana por siempre estar pendiente de todo el desarrollo del presente proyecto.

Agradecimientos en especial a la profesora Carol Martínez de la Pontificia Universidad Javeriana por sus asesorías y acompañamiento a lo largo de esta investigación. Sin dejar atrás a los profesores Iván Mondragón como director del Centro Tecnológico de Automatización Industrial y al profesor Carlos Parra como líder del grupo de investigación.

Agradecer de corazón a mis amigos más cercanos que siempre estuvieron apoyándome, motivándome y deseándome lo mejor en cada paso que daba.

Dedico este proyecto a...

Mi padre Martín Beleño Piñeres, mi principal mentor en la vida, una persona con grandes sueños e inspirando a los demás siempre a hacer grandes cosas.

Mi madre Martha Luz Díaz Alvarado, que entre regaños y cariño me formó para ver las cosas de un punto de vista más neutro y afrontar los retos que se puedan presentar en la vida.

Mis hermanos Martín Andrés Beleño Díaz y Juan Sebastián Beleño Díaz, que siempre me han apoyado y formado para ser mejor persona cada día.

Tu tiempo es limitado,
así que no lo malgastes viviendo la vida de otro...

Vive tu propia vida,
Todo lo demás es secundario.

Steve Jobs.

Índice general

Índice de figuras

Índice de tablas

Índice de Códigos

Resumen

A lo largo de los años se han implementado diferentes dispositivos para la inspección de redes eléctricas de alta tensión con el fin de reducir costos en las empresas dedicadas al sector de la industria energética. Actualmente el dispositivo mas usado para este proceso de inspección son los drones (Sampedro Pérez y cols., 2014). Al realizar vuelos sobre las redes eléctricas almacenan vídeos durante un largo período de tiempo dependiendo de la duración de la batería de cada dron. Estos videos de larga duración representan un problema, ya que los expertos necesitan separar las secciones de video mas importantes durante el proceso de inspección. Esta separación de videos le puede tomar horas de edición al operario de las redes eléctricas de alta tensión.

Esta investigación se centra en el desarrollo e implementación de algoritmos de procesamiento de imágenes basados en inteligencia artificial para facilitar el proceso de inspección de redes eléctricas, con el fin de automatizar procesos relacionados con la industria de la energía eléctrica.

En el proceso de desarrollo de los algoritmos de inteligencia artificial se realizaron vuelos con drones sobre infraestructuras eléctricas de alta tensión. De los videos captados se extrajeron 22000 imágenes para el entrenamiento de los sistemas de inteligencia artificial. Se entrenaron diferentes arquitecturas para el aprendizaje automático de los sistemas de inteligencia artificial y se implemento la mejor arquitectura de aprendizaje automático para su aplicación.

Con los algoritmos desarrollados se realizó un proceso de segmentación de videos por tipos de infraestructura eléctrica, mostrando a los operarios una información mas detallada en el proceso de inspección de torres eléctricas. El algoritmo de segmentación de videos logro el mismo desempeño que un operador humano en cuanto a la detección de los tipos de infraestructura eléctrica. Además de poder extraer imágenes con una visualización cercana de los aisladores encontrados en infraestructuras eléctricas, ya que los aisladores eléctricos son los elementos mas importantes en el proceso de inspección de torres eléctricas (Pernebayeva y cols., 2017).

Esta investigación fue llevada a cabo usando la modalidad de pasantía de investigación realizada en el Centro Tecnológico de Automatización Industrial (CTAI) de la Pontificia Universidad Javeriana.

Abstract

Over the years, different devices have been implemented for the inspection of high voltage electrical networks in order to reduce costs in companies dedicated to the energy industry sector. Currently, the most used device for this inspection process is drones (Sampedro Pérez y cols., 2014). By flying over power lines, they store videos for a long period of time depending on the battery life of each drone. These long videos pose a problem as experts need to separate the most important video sections during the inspection process. This separation of videos allows the operator of the high voltage electrical networks to take hours of editing.

This research focuses on the development and implementation of image processing algorithms based on artificial intelligence to facilitate the inspection process of electrical networks, in order to automate processes related to the electrical energy industry.

In the process of developing artificial intelligence algorithms, drone flights were carried out over high-voltage electrical infrastructure. 22,000 images were extracted from the captured videos for the training of artificial intelligence systems. Different architectures were trained for machine learning from artificial intelligence systems and the best machine learning architecture for their application was implemented.

With the algorithms developed, a process of segmentation of videos by type of electrical infrastructure was carried out, showing operators more detailed information about the inspection process of electrical towers. The video segmentation algorithm achieved the same performance as a human operator by detecting the types of electrical infrastructure. In addition to being able to extract images with a close-up view of the insulators found in electrical infrastructures, since electrical insulators are the most important elements in the process of inspecting electrical towers (Pernebayeva y cols., 2017).

This research was carried out using the research internship modality at the Technological Center for Industrial Automation of the Pontifical Xavierian University.

1. Introducción

Las redes eléctricas de alta tensión necesitan estar constantemente inspeccionadas por compañías dedicadas a la industria de la energía eléctrica para la supervisión de un correcto y seguro funcionamiento de las mismas. Para esta inspección de las infraestructuras eléctricas de alta tensión se han optado por el uso de drones para realizar la inspección de las líneas de manera automática (Sampedro Pérez y cols., 2014). Los drones graban vídeos durante un largo período de tiempo dependiendo de la duración de la batería de cada dron. Estos videos representan un problema al realizar procesos de inspección, ya que los expertos realizan el proceso de separación en secciones de video importantes para la inspección, un proceso que le puede tomar horas.

Para resolver este problema, se desarrolló un algoritmo de segmentación de información para el proceso de inspección de infraestructura eléctrica de alto voltaje. El algoritmo utiliza sistemas de inteligencia artificial basados en redes neuronales artificiales. Se usa un sistema de inteligencia artificial con el fin de reconocer líneas eléctricas y torres eléctricas. Separando los videos de larga duración capturados por un dron en secciones de video más cortas donde cada sección de video muestra un tipo diferente de infraestructura eléctrica. Esta separación de videos ahorra tiempo a las empresas dedicadas al sector de distribución de energía eléctrica al transformar la inspección de la infraestructura eléctrica en la inspección de dos clases diferentes de inspección: líneas eléctricas y torres eléctricas.

Los aisladores eléctricos representan un papel importante en el proceso de inspección de redes eléctricas de alta tensión. Los aisladores eléctricos de alta tensión están ampliamente desplegados sobre las infraestructuras eléctricas de alta tensión, donde estos aisladores se encargan de separar las líneas de alta tensión con la red a tierra, evitando arcos eléctricos y pérdidas de energía (Pernebayeva y cols., 2017). Los aisladores son los que principalmente deben de ser inspeccionados para comprobar el estado de deterioro en el que se encuentran.

Para una inspección correcta de los aisladores eléctricos de alta tensión se planteó el desarrolló un algoritmo basado en redes neuronales artificiales convolucionales. Esto con el fin de detectar los aisladores eléctricos y extraer la imagen del aislador de manera que le permita al experto en la inspección de las líneas eléctricas de alta tensión realizar una correcta visualización de los aisladores a inspeccionar.

2. Objetivos

2.1. Objetivo General

Implementar procesamiento de imágenes usando inteligencia artificial para el desarrollo de aplicaciones industriales

2.2. Objetivos específicos

Generar conjuntos de datos (*Datasets*), para realizar el entrenamiento de algoritmos de segmentación en imágenes de redes eléctricas de alta tensión usando vídeos capturados por drones.

Programar algoritmos de procesamiento de imágenes usando técnicas de inteligencia artificial para la segmentación de vídeos de larga duración en la inspección de infraestructuras eléctricas de alta tensión.

Detectar aisladores eléctricos en infraestructuras de alta tensión para la supervisión del estado en el que se encuentran.

Evaluar el funcionamiento de los algoritmos programados basándose en parámetros de visión artificial para comprobar el desempeño de los diferentes modelos de predicción en imágenes.

3. Marco Teórico

3.1. Inteligencia Artificial (IA)

La inteligencia artificial se considera una rama interdisciplinaria de las tecnologías informáticas que llevan consigo cierta complejidad matemática (Al-Imam y cols., 2020). La inteligencia artificial utiliza modelos matemáticos para el desarrollo de algoritmos referentes a redes neuronales artificiales (Abiodun y cols., 2018) y árboles de clasificación(Lowe y Kulkarni, 2015).

Una de las ramas más destacadas en el campo de la inteligencia artificial son las redes neuronales artificiales, la principal ventaja de las redes neuronales es obtener mejores aproximaciones a relaciones no lineales que los modelos lineales al utilizar relaciones de funciones complejas (Morales Castro, 2020).

En los últimos años, el aprendizaje automático o *deep Learning* ha cosechado un enorme éxito en una variedad de aplicaciones. Este nuevo campo de aprendizaje automático ha crecido rápidamente y se ha aplicado a dominios de aplicaciones más tradicionales, así como algunas áreas nuevas que presentan más oportunidades, las cuales han propuesto diferentes métodos basados en diferentes categorías de aprendizaje, incluyendo aprendizaje supervisado, semi-supervisado y no supervisado (Alom y cols., 2019).

El aprendizaje automático usando redes neuronales ha demostrado ser exitoso recientemente para muchas aplicaciones complejas que van desde el reconocimiento de imágenes hasta la medicina de precisión. Sin embargo, su aplicación directa a problemas en física cuántica es desafiante debido a la complejidad exponencial de los sistemas de muchos cuerpos (Cong y cols., 2019).

Las redes neuronales convolucionales (CNNs) son un tipo alternativo de red neuronal que se puede utilizar para reducir las variaciones espectrales y modelar las correlaciones espectrales que existen en las señales. Dado que las señales de voz exhiben estas dos propiedades, las CNNs son un modelo más eficaz para el habla en comparación con las Redes Neuronales Profundas (DNN) (Sainath y cols., 2013).

3.2. Redes Neuronales Artificiales (ANN)

Las redes neuronales artificiales fueron desde un principio basadas con sus contra partes las redes neuronales biológicas. Para empezar a abarcar en las redes neuronales artificiales es necesario saber que el cerebro consta de una gran cantidad de elementos altamente conectados llamadas sinapsis (Martin T. Hagan, 2014).

Las neuronas constan de tres componentes principales: las dendritas, el cuerpo celular y el axón como se puede ver en la figura 3.1. Las dendritas son redes receptivas en forma de árbol de fibras nerviosas que llevan señales eléctricas al cuerpo celular. El cuerpo celular suma y estandariza estas señales entrantes que son propagadas hacia el axón con una activación o inhibición. El axón es una fibra larga que transporta la señal dada por el cuerpo celular a otras neuronas (Martin T. Hagan, 2014).

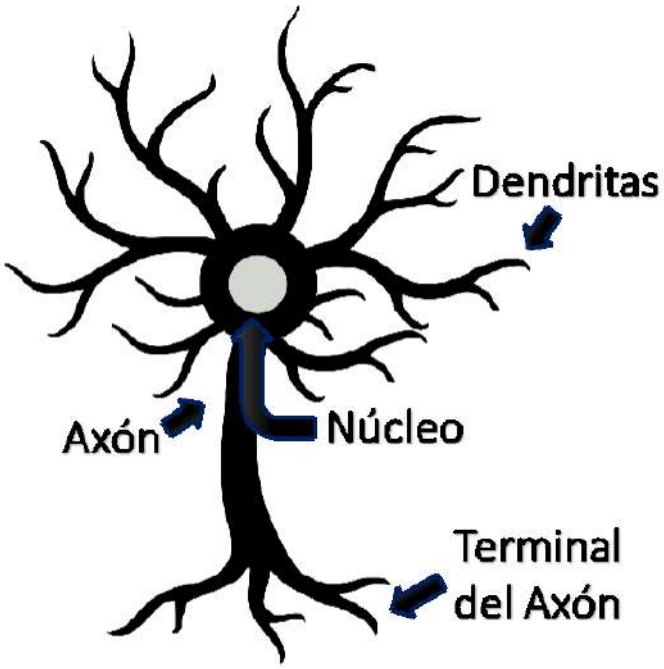

Figura 3.1: Neurona Biológica
Fuente: Autor

El punto de contacto entre un axón de una célula y una dendrita de otra célula se llama sinapsis. Es la disposición de las neuronas y las fuerzas de las sinapsis individuales, determinadas por un proceso químico complejo dentro del cuerpo celular, lo que establece la función de la red neuronal (Martin T. Hagan, 2014).

3.3. Redes Neuronales Convolucionales (CNNs)

Las redes neuronales convolucionales o CNNs por sus siglas en ingles, se han inspirado en la estructura modular de las cortezas visuales que conducen a una arquitectura que contiene muchas capas que intentan imitar algún comportamiento del sistema visual humano (Bertoni y cols., 2019).

En particular, cada capa convolucional obtiene su entrada de la capa convolucional previa. Las conexiones y similitudes entre las CNNs y el sistema visual se han estudiado ampliamente en los últimos años (Bertoni y cols., 2019). Las redes neuronales convolucionales últimamente han resurgido debido a su desempeño de vanguardia en varias disciplinas tales como la visión artificial, el procesamiento de audio y texto (Konstantinidis y cols., 2019).

Las redes neuronales convolucionales han llegado a romper barreras en el reconocimiento de imágenes obteniendo características de bajo, medio y alto nivel dentro de un conjunto de datos(He y cols., 2016). En el caso de la clasificación de dígitos escritos a mano y la detección de rostros, las cuales han demostrado un excelente desempeño (Zeiler y Fergus, 2013), tomando en cuenta los bordes que se puedan encontrar en la imagen y características propias de la red para la correcta detección desde filtros de convolución de nivel bajo hasta filtros de nivel alto para reconocer características generales, como se muestra en la figura 3.2 (Albawi y cols., 2017).

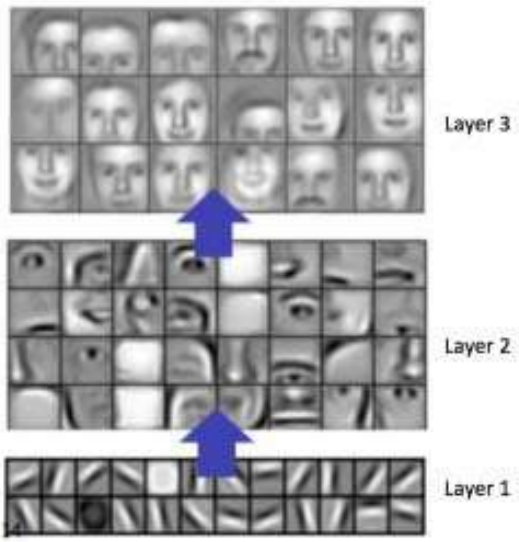

Figura 3.2: Características aprendidas de una red neuronal convolucional
Fuente: Albawi y cols.

El reconocimiento de objetos plantea un desafío importante debido a la alta variabilidad de los objetos en las imágenes a nivel de píxeles. Lo que conlleva a que obtener características informativas de una imagen es un componente necesario para lograr un rendimiento avanzado en la clasificación y detección de objetos (Sohn y cols., 2011).

3.3.1. Conceptos básicos acerca de las convoluciones

3.3.1.1. Convoluciones

Las convoluciones corresponden a las transformaciones relacionadas de un vector de entrada, que se multiplica con una matriz para producir una salida (Dumoulin y Visin, 2016). Lo cual se puede aplicara cualquier tipo de entrada, como una imagen, un clip de sonido o una colección desordenada de características. Su representación siempre se puede aplanar en un vector.

Un ejemplo de como se aplica matemáticamente hablando las operaciones convolucionales se encuentran en la figura 3.3. La cuadrícula azul clara puede ser la representación de una matriz de entrada pudiendo ser una imagen. Para mantener el dibujo simple la imagen se representa en un solo canal (Con una profundidad dimensional equivalente a 1 "uno"). Es común encontrar imágenes en 3 dimensionales dado que la imágenes de una foto se pueden expresar en el espacio del color RGB (Dumoulin y Visin, 2016).

La cuadricula gris a la derecha representa la matriz de convolución que realiza la multiplicación sobre la matriz de entrada, también conocida como Kernel (Dumoulin y Visin, 2016). La cuadricula del centro representa la matriz de salida procedente de la multiplicación del kernel sobre la cuadricula de entrada, el resultado de la cuadricula se suma el resultado resumiéndose en un solo valor sobre la cuadricula de salida. Reduciendo la dimencionalidad de salida con respecto a la cuadricula de entrada.

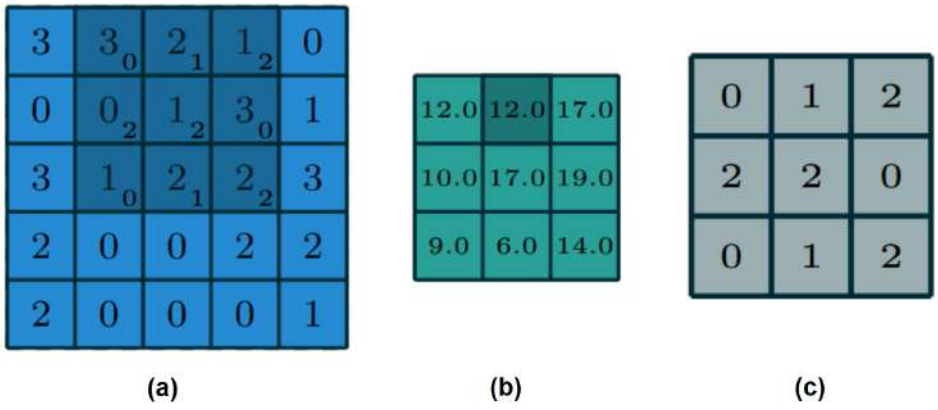

Figura 3.3: (a) Cuadricula de entrada (b) Cuadricula de salida (c) Kernel
Fuente: Dumoulin y Visin

3.3.1.2. Agrupación máxima

La Agrupación máxima o *Max pooling* es el resultado de múltiples convoluciones que extraen las características más relevantes usando los valores máximos del producto entre el kernel y la cuadricula de entrada (Saeedan y cols., 2018).

3.3.1.3. Agrupación promedio

La Agrupación promedio o *average pooling* se conoce como el resultado de múltiples convoluciones que extraen las características más relevantes usando los valores promedios del producto entre el kernel y la cuadricula de entrada. Se hace evidente la perdida de detalles en el resultado visualizando su efecto de reducción de escala sobre la cuadricula de salida (Saeedan y cols., 2018).

3.3.2. Arquitecturas para el entrenamiento de CNNs

En el proceso de entrenamiento de redes neuronales se busca implementar una arquitectura de redes neuronales que demuestre un porcentaje relativamente bajo en cuanto operaciones matemáticas y parámetros a entrenar, a su vez que tenga una precisión alta en cuanto a la detección de clases (Bianco y cols., 2018).

Existen diferentes arquitecturas de redes neuronales convolucionales la clasificación múltiples clases de objetos en imágenes implementados en el *dataset* Imagenet-1k el cual contiene mil clases de objetos diferentes (Bianco y cols., 2018). Las arquitecturas más importantes investigadas se encuentran en la Figura 3.4, donde el tamaño de cada bola corresponde a la complejidad del modelo en cuanto a parámetros de la red.

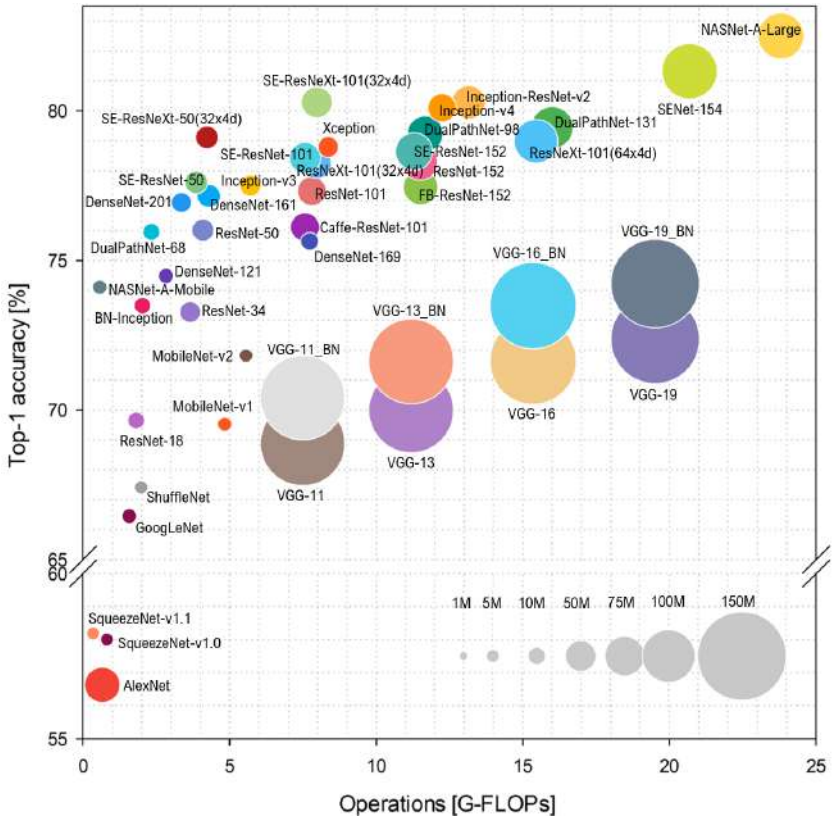

Figura 3.4: Gráfico de bolas que informa la precisión frente a la complejidad computacional.
Fuente: Bianco y cols.

3.3.2.1. AlexNet

La arquitectura AlexNet empieza con convoluciones de 11x11. Continua con capa convolucional de 5x5. Un Max pooling que permite maximizar las características obtenidas de los filtros de convolución. Se aplican convoluciones de 3x3. Un Max pooling seguido de una convolución 3x3. Continua con una convolución 3x3. Max pooling y un aplanado a las matrices de profundidad donde desemboca hacia las redes neuronales artificiales clásicas (Alex y cols., 2012). La arquitectura completa de la AlexNet se muestra en la Figura 3.5.

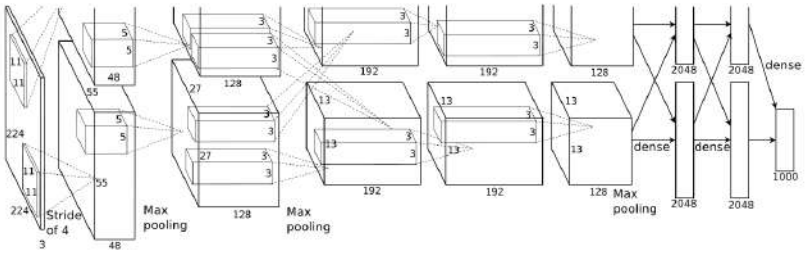

Figura 3.5: Ilustración de la arquitectura AlexNet
Fuente: Alex y cols.

3.3.2.2. VGG16

La arquitectura de VGG16 se caracteriza por contar con bancos de convoluciones a diferentes resoluciones (Simonyan y Zisserman, 2014). Usa convoluciones manteniendo la dimensionalidad y adicionalmente un max pooling a la salida de cada bloque de convoluciones. Cada vez disminuyendo la dimensionalidad y aumentando la profundidad. La arquitectura de bancos de convolución característicos de la VGG16 se muestra en la Figura 3.6.

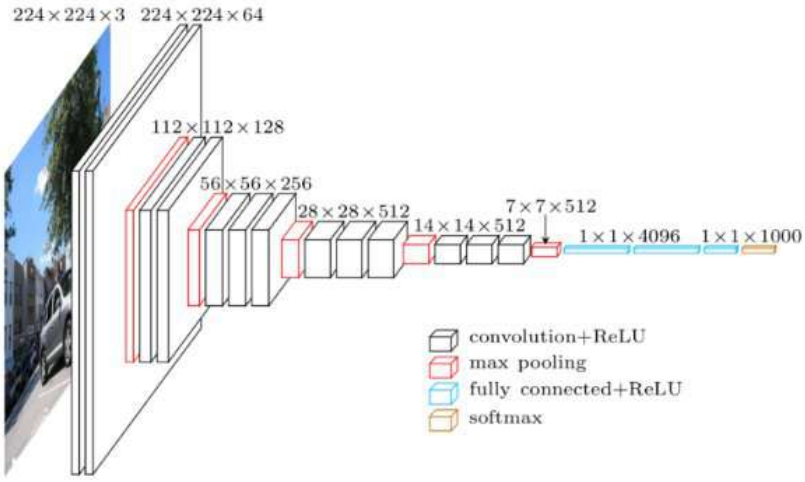

Figura 3.6: Arquitectura de la VGG16
Fuente: Sugata y Yang

3.3.2.3. InceptionV3

InceptionV3 es una arquitectura creada de manera en que el número de operaciones contra la precisión de detección de la red neuronal sea de las más óptimas gracias a sus bloques de convolución en paralelo que permiten obtener diferentes características en diferentes niveles y poder concatenarlas de nuevo para conseguir una extracción de características reducida pero con mucha más información de por medio (Szegedy y cols., 2016). La complejidad de lo bloques Inception se muestran en la Figura 3.7.

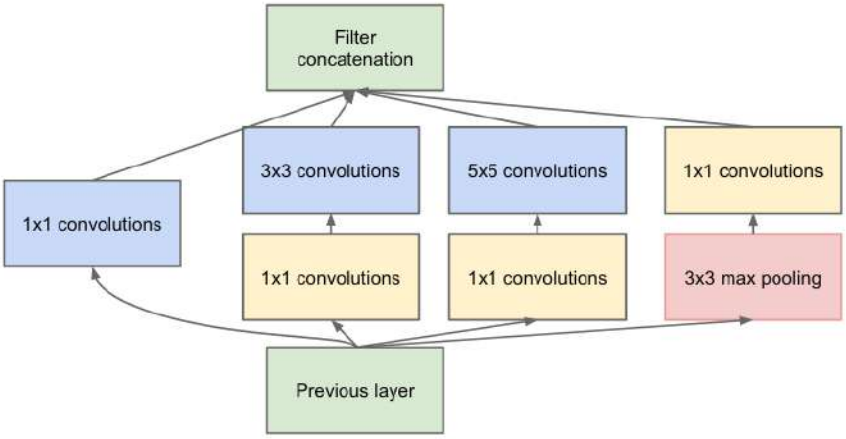

Figura 3.7: Modulo Inception con reducción de dimensión
Fuente: Szegedy y cols.

3.3.2.4. Resnet 50

ResNet es una arquitectura basada en llamados bloques residuales que permiten que la información de entrada se vea reflejada a la salida del bloque de convolución (He y cols., 2015). Agregando el resultado de la extracción de las características propias del bloque como se muestra en la Figura 3.8.

3.3.2.5. DenseNet 121

La arquitectura DenseNet esta compuesta por tres operaciones consecutivas: normalización por lotes (BN), seguida de una unidad lineal rectificada (ReLU) como una función activa de las neuronas y una convolución 3×3 (Conv) (Feng y cols., 2019). Las capas de transición se usan entre cada bloque denso, que consiste en una capa de normalización por lotes y una capa de convolución 1×1 seguida de una capa de agrupación promedio de 2×2 como se muestra en la Figura 3.9.

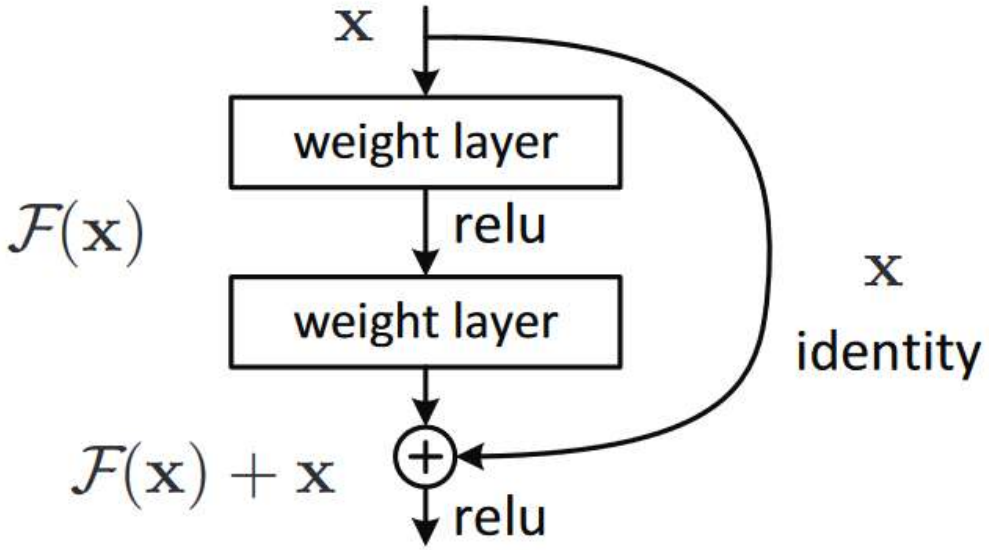

Figura 3.8: Bloque residual
Fuente: He y cols.

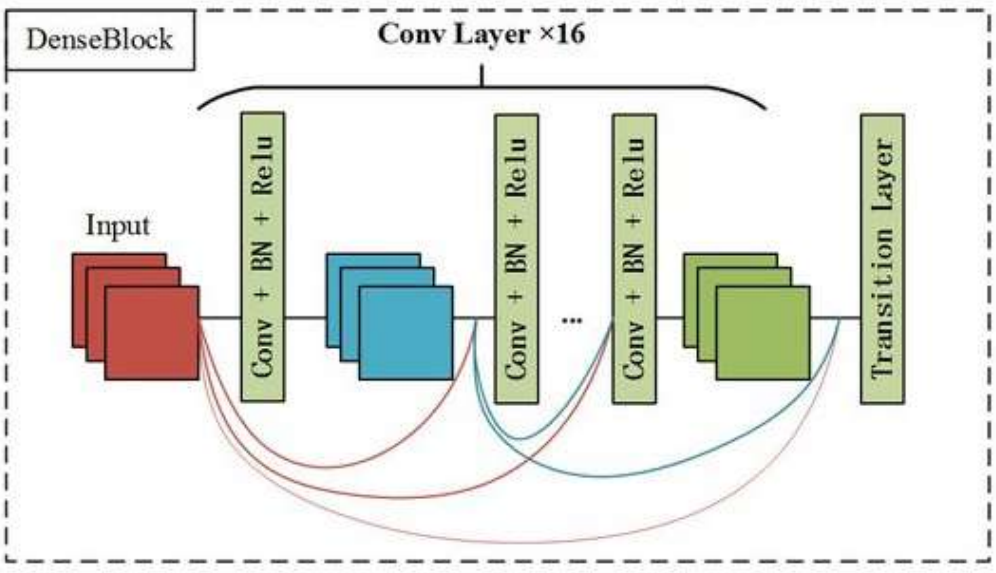

Figura 3.9: Bloques convolucionales de la DenseNet
Fuente: Feng y cols.

3.4. You Only Look Once (YOLO)

Es un sistema neuronal encargado de detectar diferentes objetos en imágenes, detectando así múltiples clases de objetos con la probabilidad de pertenecer a esa clase de objeto e indicar dónde se encuentra el objeto dentro de la imagen como se muestra en la Figura 3.10.

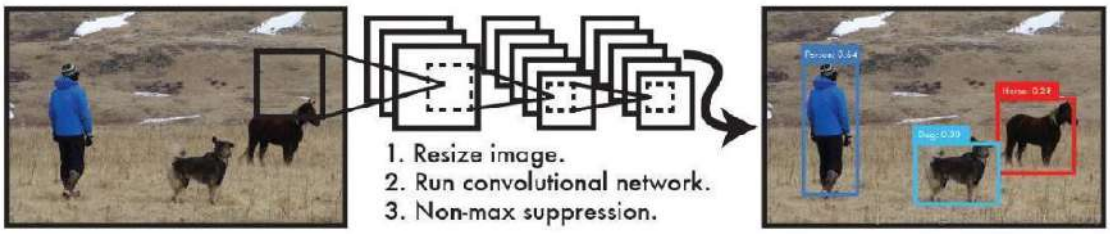

Figura 3.10: Sistema de detección de YOLO
Fuente: Redmon y cols.

La tecnología a lo largo del tiempo ha sido usada para la detección de patrones y objetos, el aprendizaje máquina a ayudado en gran medida a la detección de estos mismos logrando actualmente mejor desempeño que cualquier otro algoritmo. Al trabajar con inteligencia artificial hay un gran desafío en cuanto a tiempo computacional para desarrollar detección en tiempo real, por lo que se han diseñado nuevas estructuras como la red You Only Look Once (YOLO) para la detección objetos en imágenes (Shengyu y cols., 2019). La complejidad de la arquitectura de YOLO se muestra en la Figura 3.11.

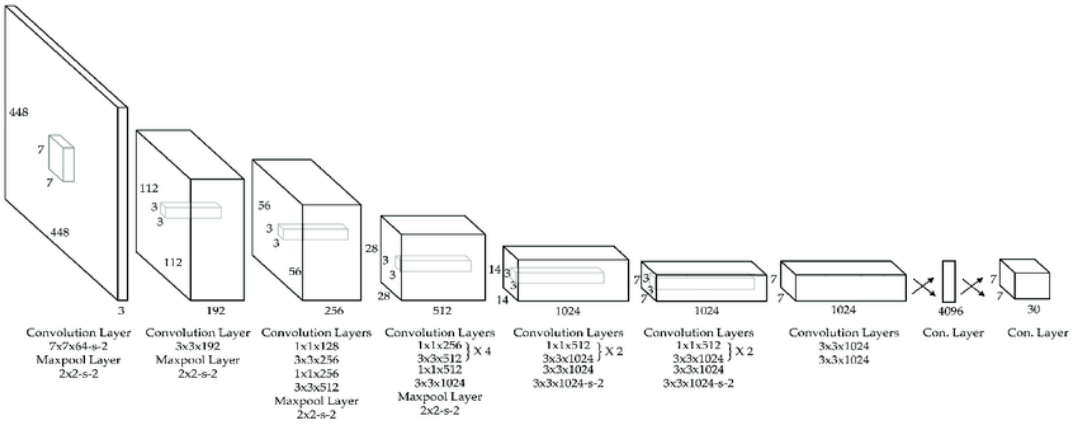

Figura 3.11: Arquitectura del sistema YOLO
Fuente: Redmon y cols.

Pero no solo eso, sino desarrollar modificaciones en la arquitectura de la red para hacerla más rápida creando así el algoritmo Fast YOLO, que se puede aplicar en tiempo real para la detección de objetos en video (Shengyu y cols., 2019).

La arquitectura original de YOLO consta de 27 capas neuronales convolucionales y 24 capas convolucionales y ha sido modificada para realizar modelos más ligeros y más rápidos como Tiny YOLO, que es un modelo que consta de 13 capas neuronales convolucionales y 7 capas convolucionales (Hendry y Chen, 2019).

YOLO se comporta como un sistema multiclase donde cada una de ellas tiene una probabilidad normalizada (Redmon y cols., 2016). Internamente consta de una red neuronal convolucional que se encarga de obtener un mapa de probabilidades que indica las diferentes clases que se pueden encontrar en la figura 3.12.

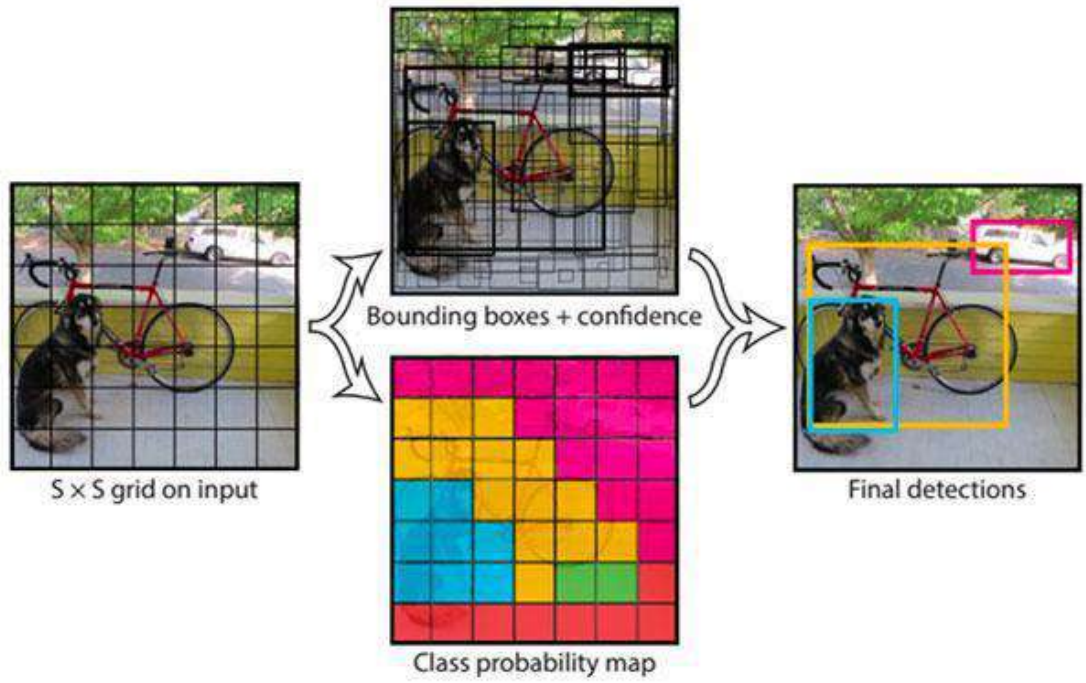

Figura 3.12: Mapa de probabilidades del algoritmo YOLO
Fuente: Redmon y cols.

Con el mapa de probabilidades se obtienen las probabilidades de clase y posteriormente su probabilidad de caja o recuadro que encierra el objeto indicando la confianza de detección de objeto. En las primeras pruebas del algoritmo YOLO se encontraron varios inconvenientes. Entre los inconvenientes encontrados fueron hallados utilizando el error cuadrático medio (MSE), la red neuronal artificial no lograba resultados con el desempeño esperado.

Posteriormente se procedió a ajustar la función de pérdidas de la red aumentando el error en las cajas de detección y reduciendo el valor de pérdidas en la precisión de detección de las clases logrando mejores resultados (Redmon y cols., 2016). Como resultado de esto hace detecciones muy precisas en cuanto a ubicación del objeto pero no con respecto a la precisión de detección de la clase a reconocer.

El algoritmo YOLO al trascender el tiempo se ha vuelto más robusto como en su versión 2 llamada YOLO 9000 siendo capaz de predecir hasta 9000 clases de objetos, se logró a través de una normalización de Batch previa a la entrada de la red neuronal convolucional ahorrando costo computacional a la red, aumentando la resolución de píxeles de entrada que pasó de 224x244 a 488x488 y en cuanto a la estructura de la red neuronal se utilizaron una red fully convolutional sin utilizar redes neuronales clásicas para la clasificación de clases (Redmon y Farhadi, 2017).

Al intentar usar una pérdida focal de la imagen. La precisión media MAP bajo unos 2 puntos. YOLO v3 ya puede ser robusto para el problema de pérdida focal está tratando de resolver porque tiene predicciones de objetividad separadas y predicciones de clase condicional (Redmon y Farhadi, 2018).

Para el entrenamiento del algoritmo YOLO es necesario ingresar tamaños predefinidos de lo recuadros que encierran los aisladores eléctricos dentro de las imágenes del *dataset* (Redmon y cols., 2016).Para hallar el tamaño correcto para cada una de las cajas predefinidas, para este proceso es recomendado usar el algoritmo Kmeans para hallar los tamaños de recuadro más representativos dentro del *dataset*.

3.5. La industria de la energía eléctrica

Hace 20 años atrás la inspección de líneas eléctricas de alta tensión se realizaban usando imágenes de alta resolución tomadas desde un helicóptero que sobrevolaba las torres eléctricas de alta tensión y realizaban una evaluación detallada, en un proceso conocido como gestión de riesgos basado en condiciones, con el fin de calcular los índices de salud para varias partes de la infraestructura eléctrica en general (Earp y cols., 2011). Como se ve en la Figura 3.13.

Figura 3.13: Adquisición del vídeo de inspección en infraestructuras eléctricas hace 20 años atrás
Fuente: Earp y cols.

Las empresas de distribución de electricidad de todo el mundo tienen el requisito de inspeccionar rutinariamente sus líneas aéreas, esto con el fin de mantener la red en condiciones seguras y evitar daños a sus trabajadores o miembros del público, mantener la seguridad del suministro de energía eléctrica y evitar interrupciones del servicio al cliente.(Earp y cols., 2011)

Uno de los procesos más importantes en la inspección de líneas de transmisión eléctrica es la detección de fallas en aisladores eléctricos de alta tensión (J. Han y cols., 2019; Li y cols., 2018). Imágenes detalladas en el proceso de inspección fueron tomadas para brindar la visualización de aisladores eléctricos captados en la zona rural a las afueras de Bogotá (Colombia), como se muestra en la Figura 3.14.

Figura 3.14: Infraestructura eléctrica de alta tensión desde una vista superior
Fuente: Pontificia Universidad Javeriana

Como el mantenimiento de las redes eléctricas de alta tensión es muy costoso, muchas empresas de servicios públicos se enfrentan al desafío de administrar una base de activos compleja y antigua, razón por la cual las técnicas de inspección oportunas y la recopilación precisa de datos (Earp y cols., 2011). La condición de los activos y el mantenimiento predictivo de las infraestructuras son la clave para administrar estas redes de la manera más eficiente y rentable posible.

Los aisladores de alto voltaje proporcionan soporte mecánico para los conductores de las líneas de transmisión y evitar el flujo de corriente de los conductores a la tierra (Pernebayeva y cols., 2017). Su clasificación puede basarse en su resistividad eléctrica, el servicio mecánico relacionado a su flexibilidad y ambiente al que esta expuesto.

Algoritmos de inteligencia artificial han realizado la detección de fallas en aisladores con las imágenes tomadas por drones con resultados no concluyentes (J. Han y cols., 2019; Li y cols., 2018). En las pruebas realizadas para detectar los aisladores eléctricos en redes de alta tensión se encontraban errores de detección tales como detectar en el final o comienzo de una estructura un defecto cuando realmente no había ninguno. Este tipo de error se muestra en la Figura 3.15.

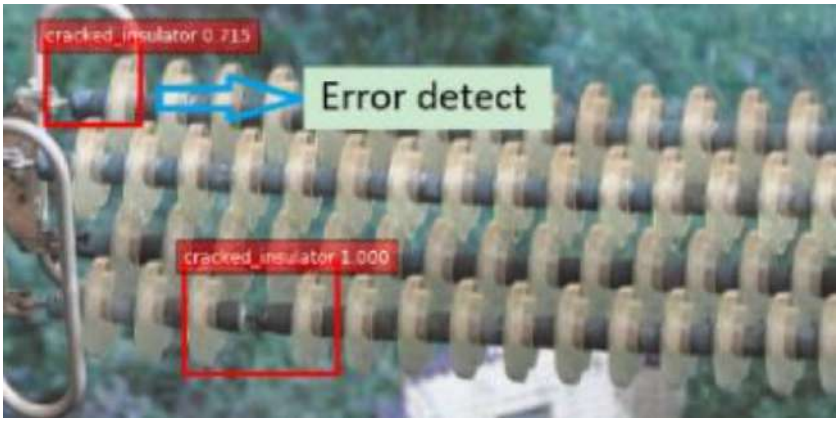

Figura 3.15: Sistema de inteligencia artificial detecta una falla erróneamente en uno de los aisladores de las líneas eléctricas de alta tensión

Fuente: Li y cols.

En caso contrario de detectar un falso positivo haciendo referencia a una detección equivocada de la detección de una falla donde el sistema de aisladores se encuentra en buen estado (Berk y Elzarka, 2019). También existe un caso contrario, el caso contrario se llama falso negativo haciendo referencia a la falta en la detección de una falla dentro del sistema de aisladores eléctricos de alta tensión. Este caso se muestra en la Figura 3.16.

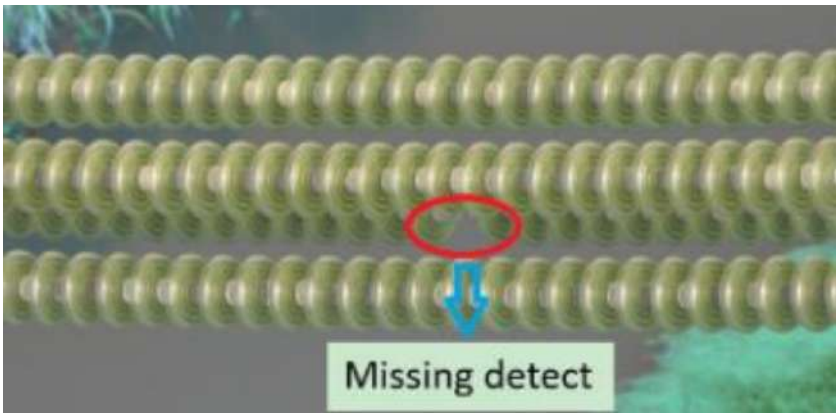

Figura 3.16: Sistema de inteligencia artificial no reconoce la falla en uno de los aisladores de las líneas eléctricas de alta tensión

Fuente: Li y cols.

4. Metodología

4.1. Inspección de líneas eléctricas de alta tensión

Una línea de alta tensión es un medio físico en el cual es posible realizar una transmisión de energía eléctrica a grandes distancias (B y cols., 2018). Las líneas de transmisión eléctricas en infraestructuras están compuestas por un material conductor por lo general cables de acero, aluminio y cobre.

Las redes eléctricas de alta tensión necesitan estar constantemente inspeccionadas por compañías dedicadas a la industria de la energía eléctrica para la supervisión de un correcto y seguro funcionamiento de las mismas. Para estas inspecciones existe cierta dificultad por el difícil acceso a la visualización de las infraestructuras eléctricas, además del alto riesgo que conlleva acceder a las alturas en las que se encuentran (Eibar y cols., 2018).

Décadas atrás diferentes tipos de dispositivos se han usado para la inspección de infraestructuras eléctricas de alta tensión, pero actualmente para realizar este tipo de se han optado por el uso de drones para realizar la inspección de las líneas de manera automática como se muestra en la figura 4.1 (Sampedro Pérez y cols., 2014).

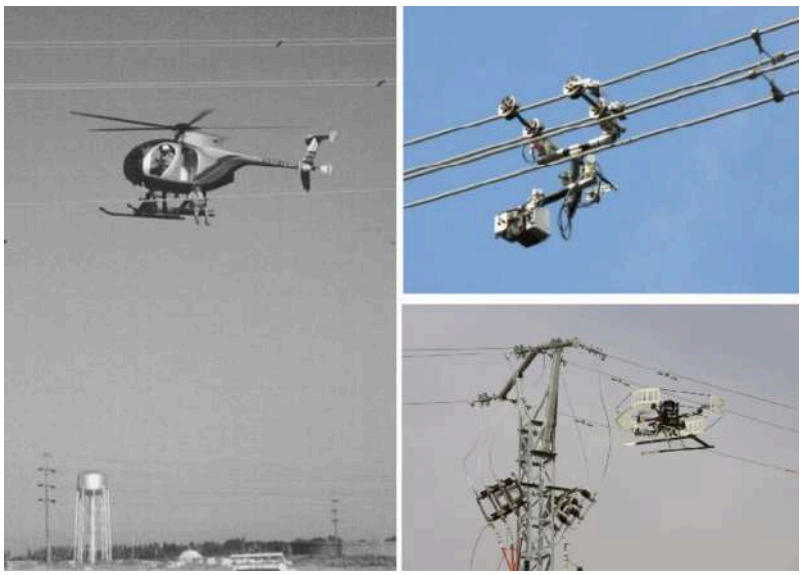

Figura 4.1: Métodos usados para la inspección de infraestructuras eléctricas de alta tensión en las ultimas décadas

Fuente: Sampedro Pérez y cols.

Los drones graban vídeos durante un largo período de tiempo dependiendo de la duración de la batería de cada dron. Los videos de larga duración representan un problema al realizar procesos de inspección, ya que los expertos realizan el proceso de separación en secciones de video importantes para la inspección, un proceso que le puede tomar horas. Para resolver este problema, se plantea el desarrollo un algoritmo de segmentación de información para el proceso de inspección de infraestructura eléctrica de alto voltaje.

El algoritmo planteado utiliza un sistema de inteligencia artificial basados en redes neuronales artificiales. Este sistema de inteligencia artificial reconocerá líneas eléctricas y torres eléctricas. Separando los videos de larga duración capturados por un dron en secciones de video más cortas. Cada sección de video mostrará un tipo diferente de infraestructura eléctrica.

Esta separación de videos ahorra tiempo a las empresas dedicadas al sector de distribución de energía eléctrica al transformar la inspección de la infraestructura eléctrica a una inspección de dos clases diferentes de estructuras: líneas eléctricas y torres eléctricas.

Para la segmentación de los vídeos de inspección se usarán algoritmos de IA enfocados al uso de redes neuronales artificiales convolucionales para la detección de las torres eléctricas de alta tensión a una distancia cercana.

Este seria el primer paso para indicar cuando empezar o terminar la captura de vídeos de segmentación desde el comienzo del trayecto de una torre a otra y en su defecto los vídeos que contengan la inspección de la infraestructura eléctrica captada por el dron.

Se tomarán imágenes utilizando un vehículo aéreo no tripulado (dron) sobrevolando las infraestructuras eléctricas de alta tensión en la zona rural de la ciudad de Bogotá (Colombia), que haría parte de la etapa 1 del proyecto como se muestra en la Figura 4.2.

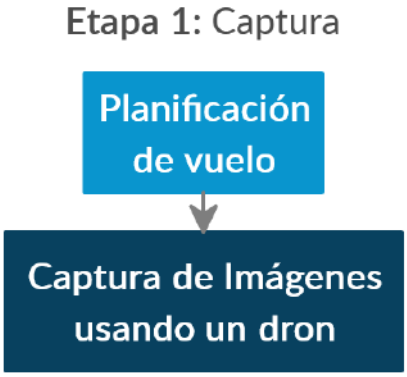

Figura 4.2: Etapa 1 para la inspección de redes eléctricas de alta tensión
Fuente: Autor

Después se procederá a realizar la generación de datos para el entrenamiento de redes neuronales. El conjunto de de datos usado para el entrenamiento de sistemas de inteligencia artificial también es conocido como dataset.

El dataset consistirá en dos clases fundamentales, una de estas clases contendrá todas las imágenes captadas en el proceso de inspección de la torre estando el dron a una distancia muy cercana de ella y en la clase contraria se tomaran imágenes donde el dron pueda estar visualizando torres de alta tensión a una distancia muy lejana, imágenes sobre las líneas de energía eléctrica y las vistas captadas hacia distancias lejanas de la infraestructura eléctrica.

Usando el dataset creado se realizará el entrenamiento de diferentes arquitecturas de sistemas de inteligencia artificial para medir el desempeño de cada una de las arquitecturas implementadas y usar el mejor modelo entrenado.

Una vez entrenado el mejor modelo se presenta la etapa 2 del proyecto, se organiza toda la serie de videos resultantes en una carpeta que contengan los videos de inspección. La carpeta de inspección se vera comprendida en 2 subcarpetas que conteniendo las diferentes clases de estructuras: líneas eléctricas y torres eléctricas. Esta etapa 2 se muestra en la Figura 4.3.

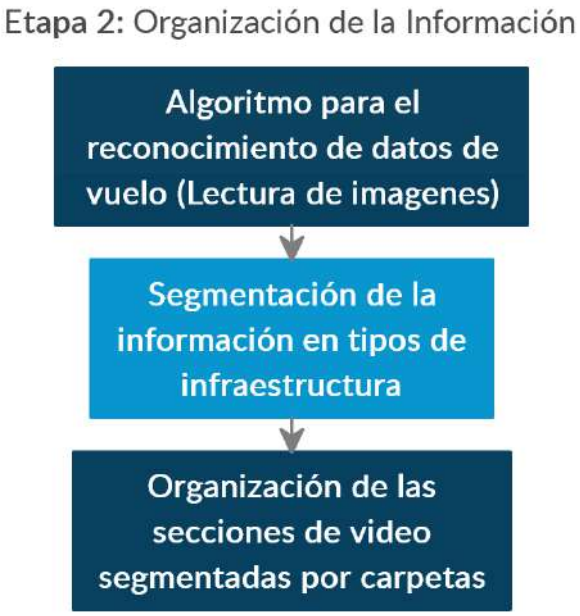

Figura 4.3: Etapa 2 para la inspección de redes eléctricas de alta tensión
Fuente: Autor

Los aisladores eléctricos representan un papel importante en el proceso de inspección de redes eléctricas de alta tensión. Los aisladores eléctricos de alta tensión están ampliamente desplegados sobre las infraestructuras eléctricas de alta tensión (Pernebayeva y cols., 2017).

4.2. Detección de aisladores eléctricos usando inteligencia artificial

Los aisladores eléctricos se encargan de separar las líneas de alta tensión con la red a tierra, evitando arcos eléctricos y pérdidas de energía (Pernebayeva y cols., 2017). Estos aisladores son los que principalmente deben de ser inspeccionados para comprobar el estado de deterioro en el que se encuentran.

Por lo cual se plantea el uso del sistema de detección YOLO (You Only Look Once) para la detección de los diferentes aisladores encontrados en cada infraestructura eléctrica, tomando la imagen exacta de solo los aisladores extrayéndose del vídeo para una visualización cercana de los mismos. La metodología de este algoritmo se muestra en la Figura 4.4.

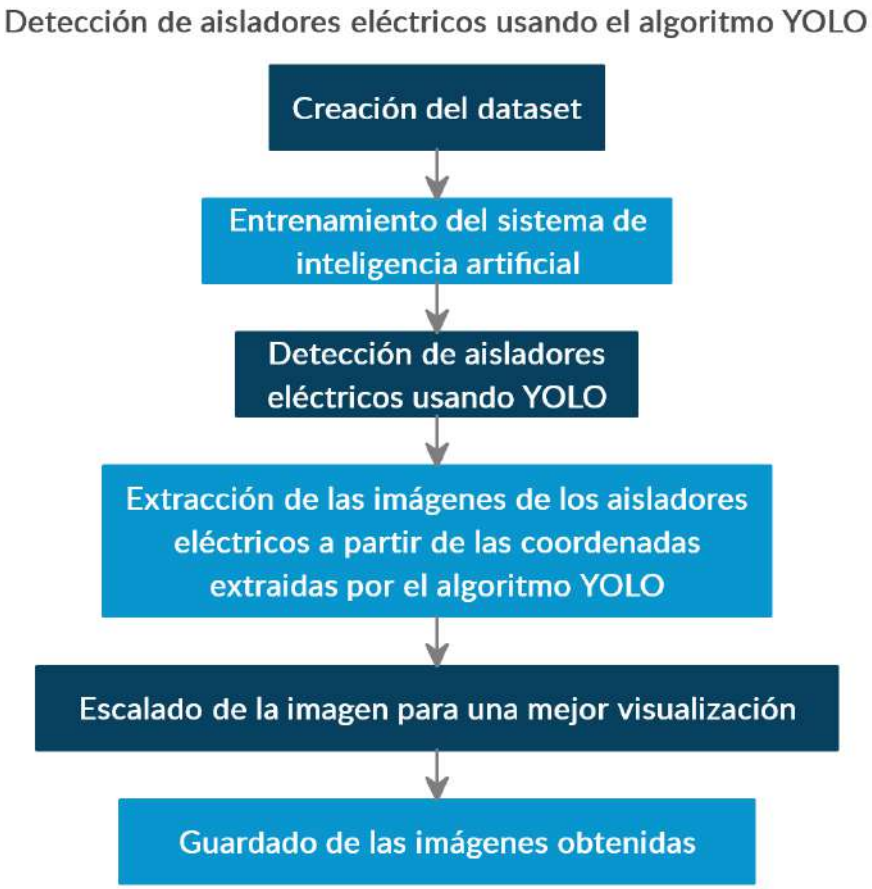

Figura 4.4: Metodología en la detección de aisladores eléctricos usando el algoritmo YOLO
Fuente: Autor

Para el entrenamiento del algoritmo YOLO es necesario contar con un dataset que contenga las imágenes con los objetos a detectar etiquetados junto con las cajas o recuadros que encierran a los objetos a detectar (Redmon y Farhadi, 2017). El proceso de etiquetado de etiquetado de imágenes se hace de manera manual y por lo general suele llevar mucho o poco tiempo dependiendo de la cantidad de datos que se quieran ingresar al dataset.

Para una optimización en el tiempo de generación del dataset de entrenamiento se propone el uso de un dataset de uso libre para la detección de aisladores eléctricos usando redes neuronales artificiales tomadas sobre infraestructuras eléctricas en china (Tao y cols., 2018), adicionalmente al dataset anterior se le pueden anexar muestras tomadas de las capturas realizadas anteriormente en la ciudad de Bogotá (Colombia) con el fin de contener la mayor cantidad de datos posibles para el dataset del sistema de detección YOLO.

Adicionalmente del dataset brindado por Tao y cols., se usarán las imágenes obtenidas de los vuelos del dron sobre las infraestructuras eléctricas en la zona rural de la ciudad de Bogotá (Colombia). A partir de las imágenes tomadas se realiza el etiquetado de los aisladores eléctricos con su respectivos software de etiquetado.

Después de realizar el proceso de etiquetado de los aisladores se implementaran los entrenamientos correspondientes al algoritmo de detección YOLO. El entrenamiento de YOLO se realizara con diferentes arquitecturas de sistemas de inteligencia artificial para medir el desempeño de cada una de las arquitecturas implementadas y usar el mejor modelo entrenado.

Posteriormente a ser entrenado el modelo, se implementará sobre videos de inspección de torres eléctricas para realizara el proceso de extracción de las imágenes que contienen aisladores eléctricos. Esto con el fin de escalar y guardar las imágenes donde se aprecie la visualización de estos aisladores desde una vista más cercana para realizar un proceso más detallado en la inspección de esta serie de elementos por parte de operarios de redes eléctricas certificados para realizar este tipo de inspección.

5. Desarrollo de los algoritmos de IA

5.1. Segmentación de vídeos de inspección sobre infraestructuras eléctricas

Se plantea el desarrollo un algoritmo de segmentación de información para el proceso de inspección de infraestructura eléctrica de alto voltaje. El algoritmo planteado utiliza un sistema de inteligencia artificial basados en redes neuronales artificiales. Este sistema de inteligencia artificial reconocerá líneas eléctricas y torres eléctricas.

El algoritmo separa los videos de larga duración capturados por un dron en secciones de video más cortas donde cada sección de video muestra un tipo diferente de infraestructura eléctrica. Esta separación de videos ahorra tiempo a las empresas dedicadas al sector de distribución de energía eléctrica al transformar la inspección de la infraestructura eléctrica a una inspección de dos clases diferentes de estructuras: líneas eléctricas y torres eléctricas.

El algoritmo organizará toda la serie de videos resultantes en una carpeta que contengan los videos de inspección. La carpeta de inspección se vera comprendida en 2 subcarpetas que conteniendo las diferentes clases de estructuras: líneas eléctricas y torres eléctricas.

5.1.1. Conjunto de datos

Se tomaron alrededor de 6 vídeos de inspección usando un dron que sobrevoló las infraestructuras de alta tensión en la zona rural de la ciudad de Bogotá (Colombia), cada vídeo entre 3 y 10 minutos de vuelo.

El conjunto de datos o *dataset* consiste en dos clases fundamentales: Lineas eléctricas y torres eléctricas. La clase de torres eléctricas contendrá todas las imágenes captadas en el proceso de inspección de la torre estando el dron a una distancia muy cercana de ella. En el caso de la clase de lineas eléctricas se tomaron imágenes donde el dron pueda estar visualizando torres de alta tensión a una distancia muy lejana, imágenes sobre las líneas de energía eléctrica y las vistas captadas hacia distancias lejanas de la infraestructura eléctrica.

Para el etiquetado de las clases se utilizaron valores numéricos en el caso de la inspección torres eléctricas se etiquetó con un valor numérico de 1. En el caso de la detección de lineas eléctricas de alta tensión se le asigno el valor numérico de 0.

De los vídeos de inspección se extrajeron un poco más de 22000 capturas de imagen, de las cuales se usaron 11000 imágenes para la detección de la infraestructura cercana en inspección y 11000 imágenes para la detección de torres de alta tensión a una distancia muy lejana, imágenes sobre las lineas de energía eléctrica y las vistas captadas hacia distancias lejanas de la infraestructura eléctrica como se muestra en la Figura 5.1.

Figura 5.1: (a) Inspección sobre la infraestructura eléctrica (b) Inspección sobre la líneas eléctricas de alta tensión y vista lejana de torres eléctricas
Fuente: Autor

5.1.2. Preprocesamiento de la información entrante a la CNN

Para el entrenamiento de las redes neuronales convolucionales con respecto a las etiquetas del *dataset* se convirtieron los valores de números decimales que los clasificaban a valores *One Hot* o también llamados valores categóricos. como se muestra en la Tabla 5.1.

Tipo de inspección	**Decimal**	**Binario**	**One Hot**
Sobre la infraestructura eléctrica (Torres eléctricas)	1	01	[0. 1.]
Fuera de la infraestructura eléctrica (Líneas eléctricas)	0	00	[1. 0.]

Tabla 5.1: Comparación entre el sistema numérico decimal, binario y One Hot

Este sistema permite el cálculo de operaciones matemáticas de una manera más fácil y reduce costos computacionales (Ghorbanzadeh y cols., 2015), a su vez se convierte los datos numéricos decimales en un vector de probabilidad de pertenencia a una de las dos clases del *dataset* como se ve en la tabla anterior.

Esta etapa de preprocesamiento sobre los *dataset* de entrenamiento, validación y testeo. Este preprocesamiento se realiza antes del entrenamiento de las redes neuronales como método de optimización a la arquitectura de redes neuronales a implementar.

Las imágenes de las capturas realizadas por el vehículo aéreo no tripulado (dron), se extrajeron imágenes de una resolución de 1088×612 píxeles.

En el entrenamiento de redes neuronales convolucionales para la clasificación de imágenes es muy común que los *datasets* contengan imágenes escaladas a medidas mas pequeñas para ahorrar costos computacionales (Bairouk, 2019). Las imágenes tomadas por los videos fueron escaladas a un tamaño de 320x240 píxeles como entrada a una red neuronal para ahorrar costos computacionales y mantener cierta proporción entre las medidas reales. Como se muestra en la Figura 5.2.

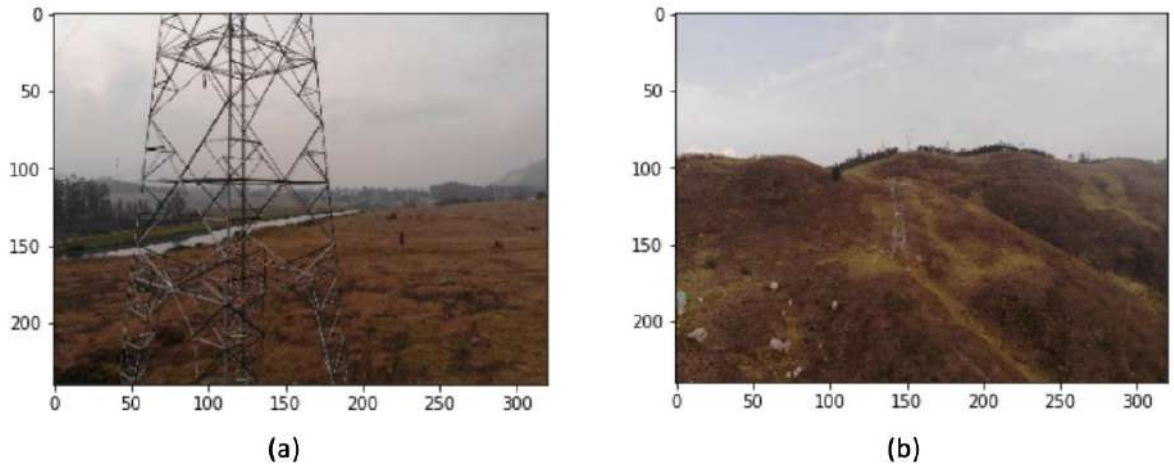

Figura 5.2: (a) Inspección sobre la infraestructura eléctrica a 320x240 píxeles (b) Inspección sobre las líneas eléctricas de alta tensión y vista lejana de torres eléctricas a 320x240 píxeles
Fuente: Autor

Ya obtenido el etiquetado de clases en el *dataset* se realizo una separación de datos de manera aleatoria para crear 3 *datasets* de información ordenados para el entrenamiento, validación y testeo de la red neuronal convolucional a entrenar, donde el *dataset* de entrenamiento le corresponde un 70% de la data, al *dataset* de validación un 15% y al *dataset* de testeo le corresponde el otro 15% de información restante del *dataset*.

El *dataset* de entrenamiento contiene 15399 imágenes de infraestructuras eléctricas almacenadas de manera aleatoria. La sección del *dataset* de validación contiene 3301 imágenes de infraestructuras eléctricas almacenadas de manera aleatoria. La sección del *dataset* de testeo contiene 3300 imágenes de infraestructuras eléctricas almacenadas de manera aleatoria.

Como paso final el *dataset* con todas la imágenes, etiquetas y paquetes se guarda en un archivo .hdf5 pesando alrededor de 5GB de información en el disco duro. Adicionalmente para visualizar las imágenes captadas por el dron en vuelo sobre las infraestructuras eléctricas en algunas de sus tomas se pueden encontrar en el anexo I.

5.1.3. Entrenamiento de las arquitecturas de redes neuronales artificiales

En este apartado se expresan como están constituidas las arquitecturas de redes neuronales artificiales. Además de presentar los parámetros e hiper-parámetros de entrenamiento.

Las imágenes ingresadas a la red tienen una entrada de 320x240 producto del preprocesamiento de las imágenes del *dataset*. Se implemento un *learning rate* o tasa de aprendizaje de 2e-5. *Batch size* o número de imágenes de entrada de 10. Número de épocas de 10. Se usó una función de costos categórica con entropía cruzada como una función de pérdida y se utilizó *Adam* como método de optimización.

5.1.3.1. DenseNet 121

En código 5.1 se presenta el algoritmo que constituye la arquitectura capa por capa usando la DenseNet 121.

Código 5.1: Entrenamiento de la arquitectura de redes neuronales convolucionales DenseNet 121

```
#Inputs
inputs = Input(shape=img_shape, name='images')

#DenseNet Model
output = densenet.DenseNet121(include_top=False,weights=None,
                              input_shape=img_shape,
                              classes = num_classes)(inputs)


#AveragePooling2D
output = AveragePooling2D(pool_size=(2, 2), strides=None,
                          padding='valid',name='AvgPooling')(output)

#Flattened
output = Flatten(name='Flatten')(output)

#Dropout
output = Dropout(0.2,name='Dropout')(output)

#ReLU layer
output = Dense(10, activation = 'relu',name='ReLU')(output)

#Dense layer
output = Dense(num_classes, activation='softmax',name='softmax')(output)

#Checkpoint_path

# Create checkpoint callback
model_checkpoint = ModelCheckpoint(filepath="/content/drive/My Drive/Newdensenet/densenet.h5",
                                   monitor='val_loss', save_best_only=True)

#Model
modelo = Model(inputs=inputs, outputs=output)

ADAM = Adam(lr=lrate)
modelo.compile(loss='categorical_crossentropy',optimizer=ADAM,
               metrics=['categorical_accuracy'])

#Summary
modelo.summary()
```

En la tabla 5.2 se presenta como esta constituida la arquitectura neuronal DenseNet. La forma de salida representa la dimensionalidad de la salida a cada una de las capas implementadas en la red.

Tipo de capa	Forma de salida	Parámetros
InputLayer	[(None, 240, 320, 3)]	0
Densenet121	(None, 7, 10, 1024)	7037504
AvgPooling	(None, 3, 5, 1024)	0
Flatten	(None, 15360)	0
Dropout	(None, 15360)	0
ReLU (Dense)	(None, 10)	153610
softmax (Dense)	(None, 2)	22

Tabla 5.2: Sumario de la arquitectura DenseNet 121

5.1.3.2. Inception V3

En codigo 5.2 se presenta el algoritmo que constituye la arquitectura capa por capa usando la Inception V3.

Código 5.2: Entrenamiento de la arquitectura de redes neuronales convolucionales Inception V3

```
#Inputs
inputs = Input(shape=img_shape, name='images')

#InceptionV3 Model
output = inception_v3.InceptionV3(include_top=False,weights=None,
                                  input_shape=img_shape,
                                  classes = num_classes)(inputs)


#AveragePooling2D
output = AveragePooling2D(pool_size=(2, 2), strides=None,
                          padding='valid',name='AvgPooling')(output)

#Flattened
output = Flatten(name='Flatten')(output)

#Dropout
output = Dropout(0.2,name='Dropout')(output)

#ReLU layer
output = Dense(10, activation = 'relu',name='ReLU')(output)

#Dense layer
output = Dense(num_classes, activation='softmax',name='softmax')(output)

#Checkpoint_path

# Create checkpoint callback
model_checkpoint = ModelCheckpoint(filepath="/content/drive/My Drive/Newinception/inceptionv3.h5",
```

```
                                    monitor='val_loss', save_best_only=True)

#Model
modelo = Model(inputs=inputs, outputs=output)

ADAM = Adam(lr=lrate)
modelo.compile(loss='categorical_crossentropy',optimizer=ADAM,
               metrics=['categorical_accuracy'])

#Summary
modelo.summary()
```

En la tabla 5.3 se presenta como esta constituida la arquitectura neuronal InceptionV3. La forma de salida representa la dimensionalidad de la salida a cada una de las capas implementadas en la red.

Tipo de capa	Forma de salida	Parámetros
InputLayer	[(None, 240, 320, 3)]	0
InceptionV3	(None, 6, 8, 2048)	21802784
AvgPooling	(None, 3, 4, 2048)	0
Flatten	(None, 24576)	0
Dropout	(None, 24576)	0
ReLU (Dense)	(None, 10)	245770
softmax (Dense)	(None, 2)	22

Tabla 5.3: Sumario de la arquitectura Inception V3

5.1.3.3. ResNet 50

En codigo 5.3 se presenta el algoritmo que constituye la arquitectura capa por capa usando la ResNet 50.

Código 5.3: Entrenamiento de la arquitectura de redes neuronales convolucionales ResNet 50

```
#Inputs
inputs = Input(shape=img_shape, name='images')

#ResNet Model
output = resnet.ResNet50(include_top=False,weights=None,
                                    input_shape=img_shape,
                                    classes = num_classes)(inputs)


#AveragePooling2D
output = AveragePooling2D(pool_size=(2, 2), strides=None,
                          padding='valid',name='AvgPooling')(output)

#Flattened
output = Flatten(name='Flatten')(output)

#Dropout
```

```
output = Dropout(0.2,name='Dropout')(output)

#ReLU layer
output = Dense(10, activation = 'relu',name='ReLU')(output)

#Dense layer
output = Dense(num_classes, activation='softmax',name='softmax')(output)

#Checkpoint_path

# Create checkpoint callback
model_checkpoint = ModelCheckpoint(filepath="/content/drive/My Drive/Newresnet/resnet50.h5",
                                   monitor='val_loss', save_best_only=True)

#Model
modelo = Model(inputs=inputs, outputs=output)

ADAM = Adam(lr=lrate)
modelo.compile(loss='categorical_crossentropy',optimizer=ADAM,
               metrics=['categorical_accuracy'])

#Summary
modelo.summary()
```

En la tabla 5.3 se presenta como esta constituida la arquitectura neuronal ResNet50. La forma de salida representa la dimensionalidad de la salida a cada una de las capas implementadas en la red.

Tipo de capa	Forma de salida	Parámetros
InputLayer	[(None, 240, 320, 3)]	0
InceptionV3	(None, 8, 10, 2048)	23587712
AvgPooling	(None, 4, 5, 2048)	0
Flatten	(None, 40960)	0
Dropout	(None, 40960)	0
ReLU (Dense)	(None, 10)	409610
softmax (Dense)	(None, 2)	22

Tabla 5.4: Sumario de la arquitectura ResNet 50

5.1.4. Algoritmo de segmentación

El algoritmo de detección y segmentación de videos de corta larga duración a videos de corta duración de los tipos de infraestructuras eléctricas encontradas funciona de la siguiente manera secuencialmente.

Se inserta un video en el algoritmo. El algoritmo intenta leer el primer fotograma del video insertado. Si el algoritmo puede leer el cuadro que pasa al otro proceso, pero no sucede, el siguiente paso sería tratar de guardar el video de inspección en una ruta en caso de que un video se haya grabado previamente. Cuando se reconoce el marco, pasa por el sistema de detección mediante el aprendizaje automático.

El sistema de detección basado en el aprendizaje automático se lleva a cabo para clasificar cuando el dron vuela sobre las líneas eléctricas o una torre eléctrica. El sistema de aprendizaje automático utiliza la arquitectura mejor capacitada teniendo en cuenta los modelos elegidos.

El algoritmo pregunta si el resultado del sistema de detección es la clase de línea eléctrica o la clase de torre eléctrica. Si el resultado del sistema de detección es la clase de línea eléctrica, el algoritmo comienza a registrar los cuadros sobre las líneas eléctricas.

Si el resultado del sistema de detección es la clase de torre eléctrica, el algoritmo comienza a registrar los marcos sobre las torres eléctricas. En caso de detectar una clase diferente mientras graba un video, el video de grabación se detiene y comienza un nuevo video con la clase que fue reconocida.

Si se leyeron todos los cuadros, el algoritmo guarda el último video tomado y finaliza el código. La dinámica de funcionamiento del algoritmo completo se muestra en la figura 5.3.

Nota: al decir que se intenta una acción dentro del diagrama hace referencia al comando *Try* que posee el lenguaje de programación python, el cual intenta realizar una acción y si la acción no es lógicamente posible o errónea, no interfiere en la ejecución del resto del código del algoritmo.

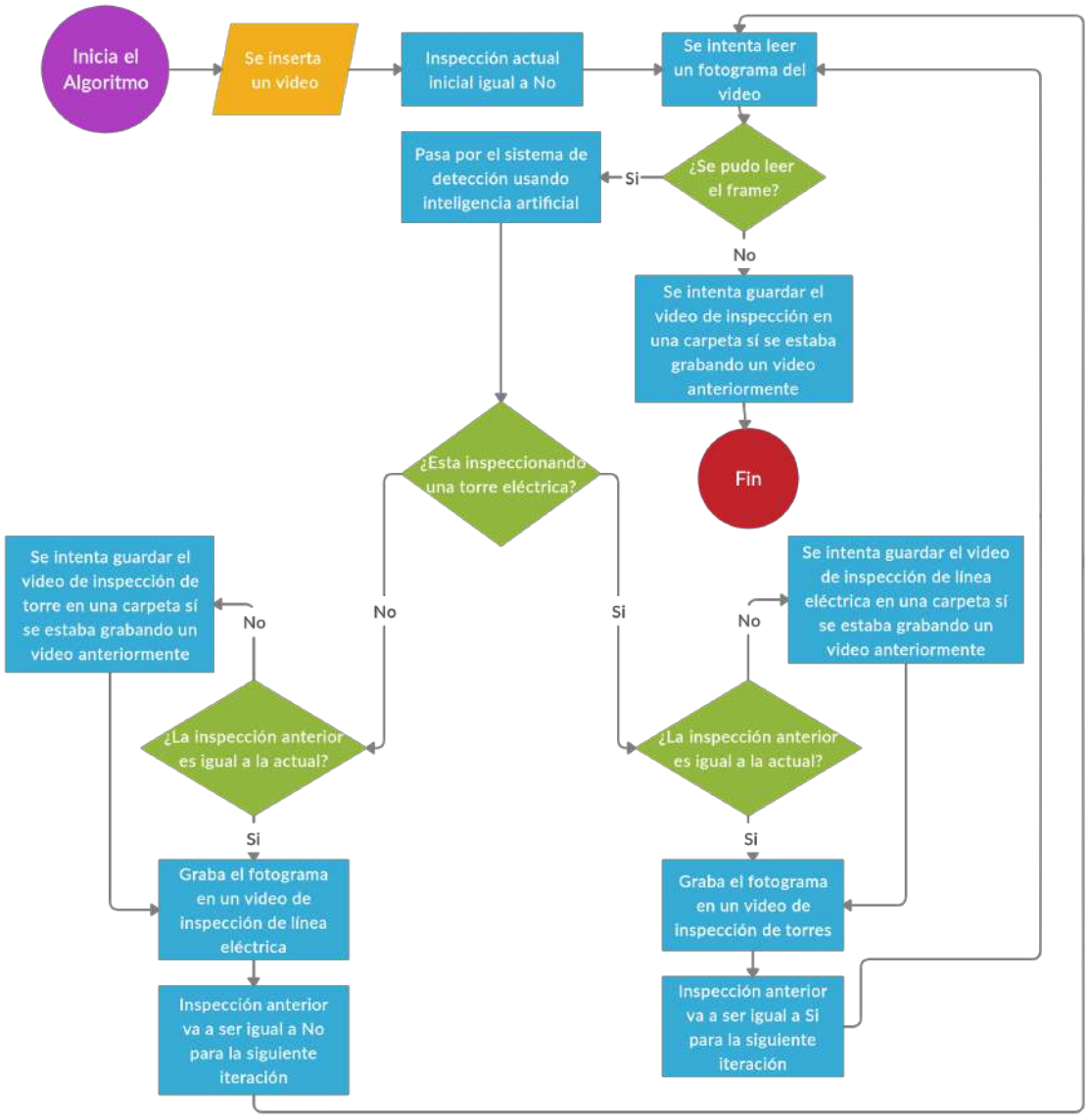

Figura 5.3: Diagrama de flujo de datos del algoritmo
Fuente: Autor

5.2. Detección de aisladores eléctricos en infraestructuras de alta tensión

Para la detección de aisladores en infraestructuras eléctricas de alta tensión fue utilizado el algoritmo conocido como YOLO, con el fin de realizar la detección de cada aislador y extraer la imagen del aislador eléctrico con una visualización más cercana para el proceso de inspección de la infraestructura eléctrica mas detalladamente.

5.2.1. Conjunto de datos

Los aisladores eléctricos se encargan de separar las líneas de alta tensión con la red a tierra, evitando arcos eléctricos y pérdidas de energía (Pernebayeva y cols., 2017). Estos aisladores son los que principalmente deben de ser inspeccionados para comprobar el estado de deterioro en el que se encuentran, por lo que se pretende utilizar el sistema de detección YOLO (*You Only Look Once*) para la detección de los diferentes aisladores de cada infraestructura eléctrica tomando la imagen exacta de solo los aisladores extrayéndose del vídeo para una visualización cercana de los mismos.

Para el entrenamiento del algoritmo YOLO es necesario contar con un *dataset* que contenga las imágenes con los objetos a detectar etiquetados junto con las cajas o recuadros que encierran a los objetos a detectar (Redmon y Farhadi, 2017), este proceso de etiquetado se hace de manera manual y por lo general suele llevar mucho o poco tiempo dependiendo de la cantidad de datos que se quieran ingresar al *dataset*. Como se muestra en la Figura 5.4.

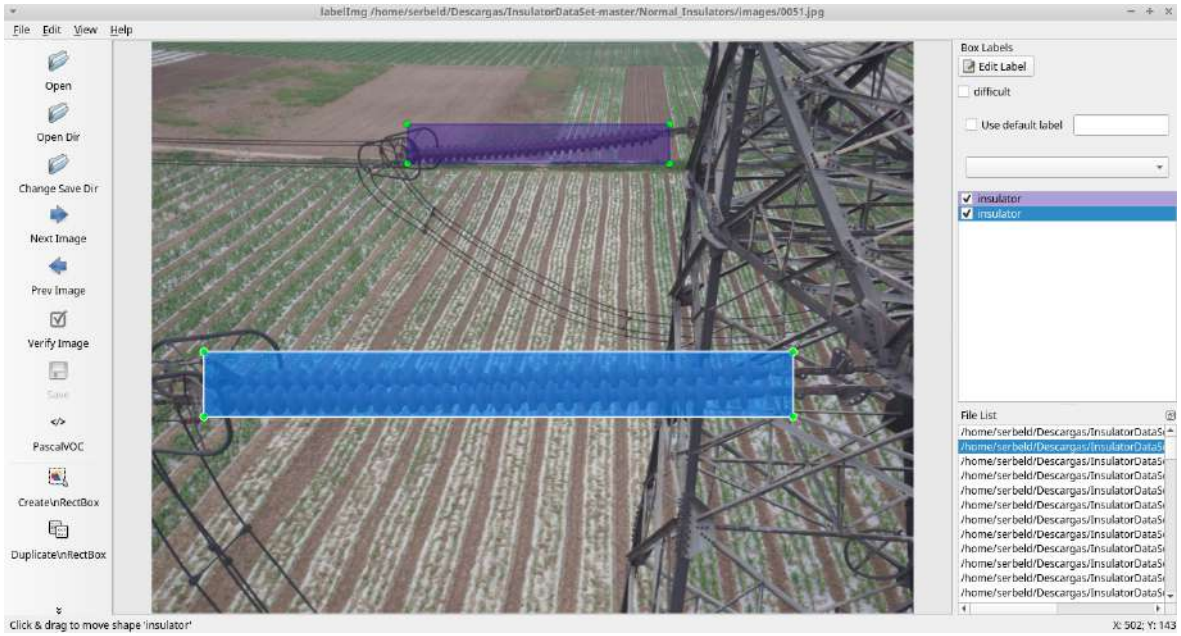

Figura 5.4: Visualización del dataset para la detección con el sistema YOLO con imágenes etiquetadas de aisladores eléctricos usando el software LabelImg

Fuente: Autor

Para una optimización en el tiempo de implementación del algoritmo de detección se propone el uso de un *dataset* de uso libre para la detección de aisladores eléctricos usando redes neuronales artificiales tomadas sobre infraestructuras eléctricas en china (Tao y cols., 2018), las cuales tuvieron que ser modificadas para agregar los aisladores verticales que no estaban correctamente etiquetados como se puede ver en la figura 5.5.

Figura 5.5: Visualización del dataset para la detección de aisladores eléctricos verticales usando el software LabelImg

Fuente: Autor

El software labelImg es un software *Open Source* o de código abierto que permite etiquetar objetos dentro de imágenes. El software guarda los datos de los objetos etiquetados en un archivo .Json en lenguaje de etiquetas. El archivo .Json contiene la ubicación de los puntos que conforman cada uno de los recuadros que encierran los objetos encontrados en la imagen, las clases de objetos que contiene la imagen y el nombre del archivo de la imagen que fue etiquetada.

Adicionalmente al *dataset* anterior se le anexaron muestras tomadas de las capturas realizadas anteriormente en la ciudad de Bogotá (Colombia), este proceso se realizo un etiquetado manualmente usando el software LabelImg con el fin de contener la mayor cantidad de datos posibles para el *dataset* de entrenamiento para el sistema de detección YOLO, como se puede apreciar en la figura 5.6 y 5.7.

Para más visualizaciones de las imágenes a las que fueron aplicadas una modificación en las etiquetas para la detección de aisladores eléctricos y etiquetado del *dataset* tomado en la ciudad de Bogotá se pueden encontrar en el anexo 2.

Figura 5.6: Visualización de imágenes del dataset de infraestructuras eléctricas tomada en la ciudad de Bogotá para la detección de aisladores eléctricos usando el software LabelImg
Fuente: Autor

Figura 5.7: Visualización de imágenes del dataset de infraestructuras eléctricas tomada en la ciudad de Bogotá para la detección de aisladores eléctricos usando el software LabelImg
Fuente: Autor

5.2.2. Entrenamiento del algoritmo YOLO

Para el entrenamiento del algoritmo YOLO se estudian las arquitecturas convolucionales que permitieran al algoritmo la mayor optimización de la red como se ve en la sección 3.3.2 y 3.4 del marco teórico. Entre ellas la arquitectura Full YOLO, InceptionV3 y MobileNet.

El *Backend* hace referencia a las diferentes arquitecturas de redes neuronales convoluciones usadas para la extracción de características generalmente en imágenes, características como colores y formas (Albawi y cols., 2017). Todos los *backends* fueron entrenados con los parámetros e hiperparámetros mostrados en la tabla 5.5.

Parámetros e hiperparámetros	Valores
Tamaño del lote por iteración (Batch size)	4
Número de épocas	40
Tamaños de recuadro predefinidos	5
Optimizador	Adam
Tamaño de entrada a la red	416x416x3
Tasa de aprendizaje	1e-03 y 1e-04

Tabla 5.5: Parámetros e hiperparámetros usados en el entrenamiento del sistema de inteligencia artificial

5.2.2.1. Full YOLO, MobileNet e InceptionV3

El algoritmo **Full YOLO** consiste en una de las arquitecturas de redes neuronales convolucionales propuestas para la detección de objetos dentro de un espacio de trabajo, junto con la detección de múltiples clases de objetos a detectar. En la figura 3.4 y 3.5 de la sección del marco teórico se encuentra un esquema de la arquitectura con los bloques convolucionales que la conforman. **MobileNet** es una arquitectura creada con poca complejidad en vista de su aplicación en dispositivos móviles (Bianco y cols., 2018). **InceptionV3** hace parte de las arquitecturas más óptimas para la extracción de características usando redes neuronales convolucionales (Bianco y cols., 2018).

En el anexo D.4, D.5 y D.6 se presentan las funciones que contienen los bloques convolucionales de las arquitecturas Full YOLO, MobileNet e InceptionV3 respectivamente desarrolladas por Allan Zelener con el uso de las librerías Tensorflow y Keras.

6. Resultados

6.1. Segmentación de vídeo con torres eléctricas de alta tensión

A continuación se presentan las gráficas de desempeño en el entrenamiento de las arquitecturas de redes neuronales artificiales usadas para la detección de torres eléctricas de alta tensión.

6.1.1. DenseNet121

La gráfica encontrada en la Figura 6.1 representa el ajuste de la precisión de entrenamiento y validación conforme la red neuronal convolucional iba siendo entrenada usando la arquitectura DenseNet121.

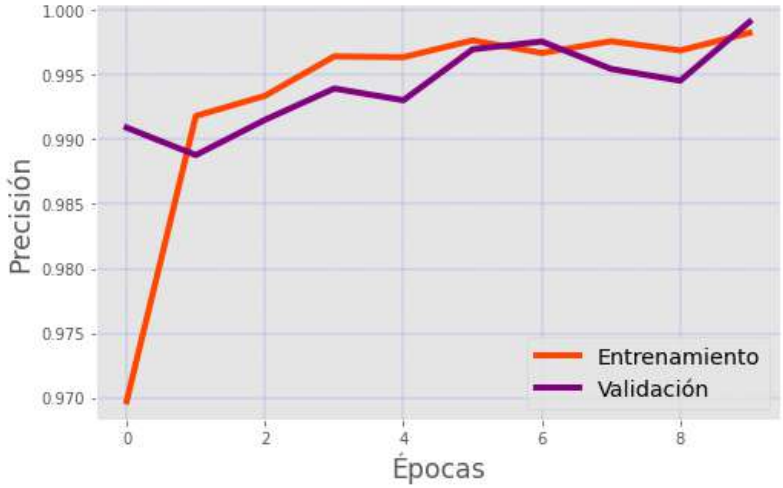

Figura 6.1: Gráfica de precisión vs épocas de la arquitectura DenseNet121
Fuente: Autor

La gráfica encontrada en la Figura 6.2 representa el ajuste de los costos o perdidas de entrenamiento y validación conforme la red neuronal convolucional iba siendo entrenada usando la arquitectura DenseNet121.

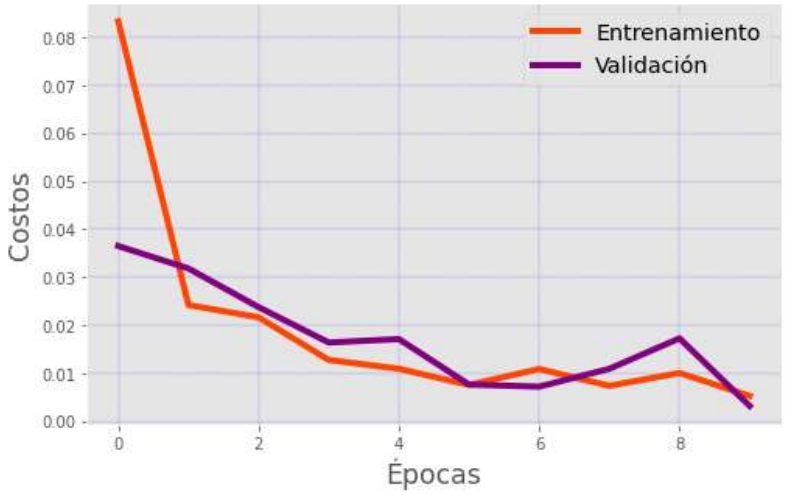

Figura 6.2: Gráfica de costos vs épocas de la arquitectura DenseNet121
Fuente: Autor

6.1.2. Resnet50

La gráfica encontrada en la Figura 6.3 representa el ajuste de la precisión de entrenamiento y validación conforme la red neuronal convolucional iba siendo entrenada usando la arquitectura ResNet50.

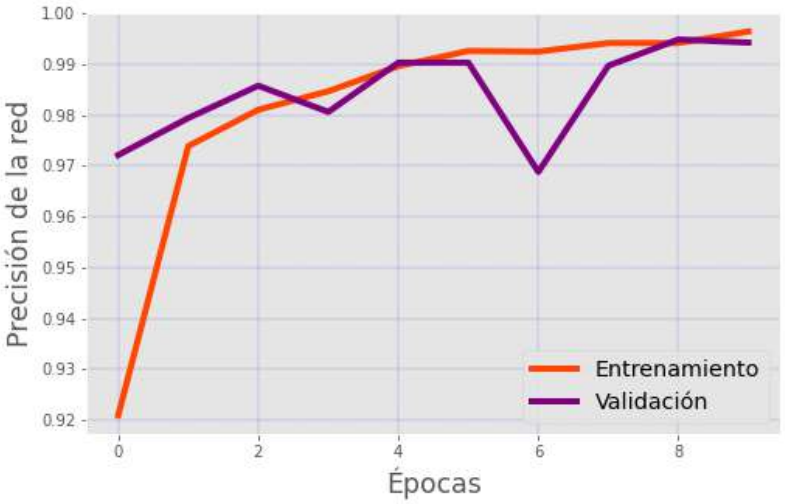

Figura 6.3: Gráfica de precisión vs épocas de la arquitectura Resnet50
Fuente: Autor

La gráfica encontrada en la Figura 6.4 representa el ajuste de los costos o perdidas de entrenamiento y validación conforme la red neuronal convolucional iba siendo entrenada usando la arquitectura ResNet50.

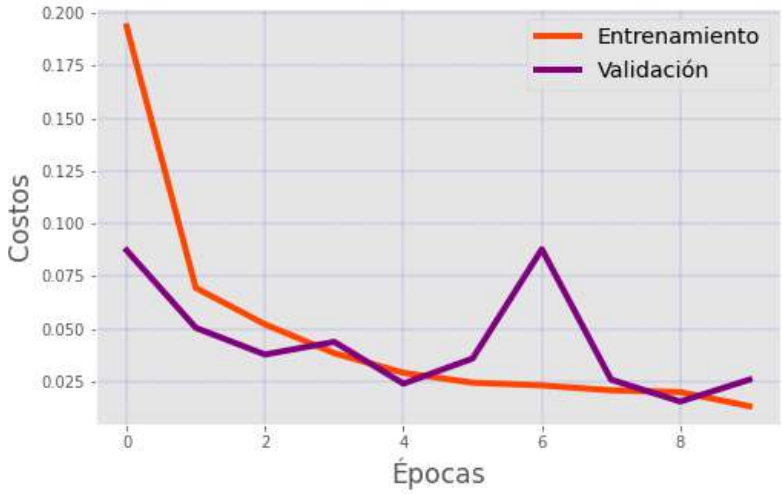

Figura 6.4: Gráfica de costos vs épocas de la arquitectura Resnet50
Fuente: Autor

6.1.3. InceptionV3

La gráfica encontrada en la Figura 6.5 representa el ajuste de la precisión de entrenamiento y validación conforme la red neuronal convolucional iba siendo entrenada usando la arquitectura InceptionV3.

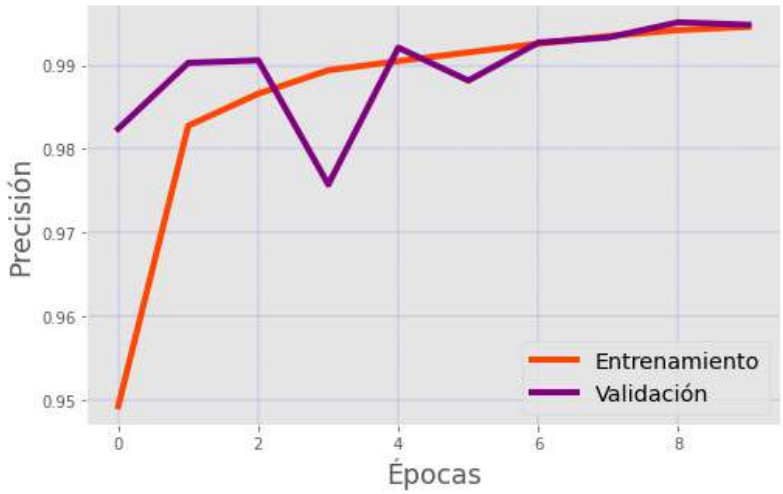

Figura 6.5: Gráfica de precisión vs épocas de la arquitectura InceptionV3
Fuente: Autor

La gráfica encontrada en la Figura 6.6 representa el ajuste de los costos o perdidas de entrenamiento y validación conforme la red neuronal convolucional iba siendo entrenada usando la arquitectura InceptionV3.

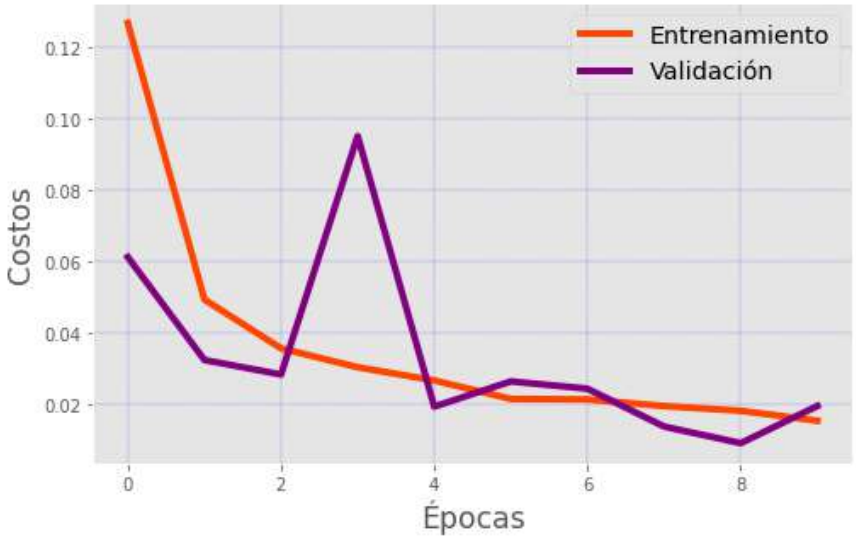

Figura 6.6: Gráfica de costos vs épocas de la arquitectura InceptionV3
Fuente: Autor

Los siguientes datos fueron tomados del *dataset* creado para testeo donde se especifican los desempeños de cada arquitectura entrenada.

Modelo	Precisión	Costos	Especificidad	Sensibilidad
DenseNet 121	99.97%	0.0015	99.94%	100%
Resnet 50	99.64%	0.0101	99.59%	99.69%
Inception V3	99.52%	0.0099	99.17%	99.88%

Tabla 6.1: Estadísticas de desempeño en la detección de torres eléctricas de alta tensión

Donde la especificidad indica el porcentaje de precisión en la detección de las líneas eléctricas de alta tensión y la sensibilidad indica el porcentaje de precisión en la detección de las torres eléctricas de alta tensión a una distancia cercana.

Implementando el mejor algoritmo de detección, en este caso la arquitectura DenseNet 121 se obtienen resultados acordes al desempeño establecido en un segmento de vídeo captado por el dron sobre las lineas eléctricas de alta tensión.

Al comparar las predicciones realizadas por la arquitectura DenseNet121 y un operador humano las detecciones son relativamente iguales. Se uso un muestreo de 466 fotogramas extraídos de un video de un dron sobre las lineas eléctricas de alta tensión.

Las clases están separadas en 2, donde el sistema de detección introduce un cero (0) o un uno (1) a su salida. La clase cero (0) hace referencia a la detección a la detección de lineas eléctricas y la clase uno (1) hace referencia a la detección a la detección de torres eléctricas como se muestra en la Figura 6.7.

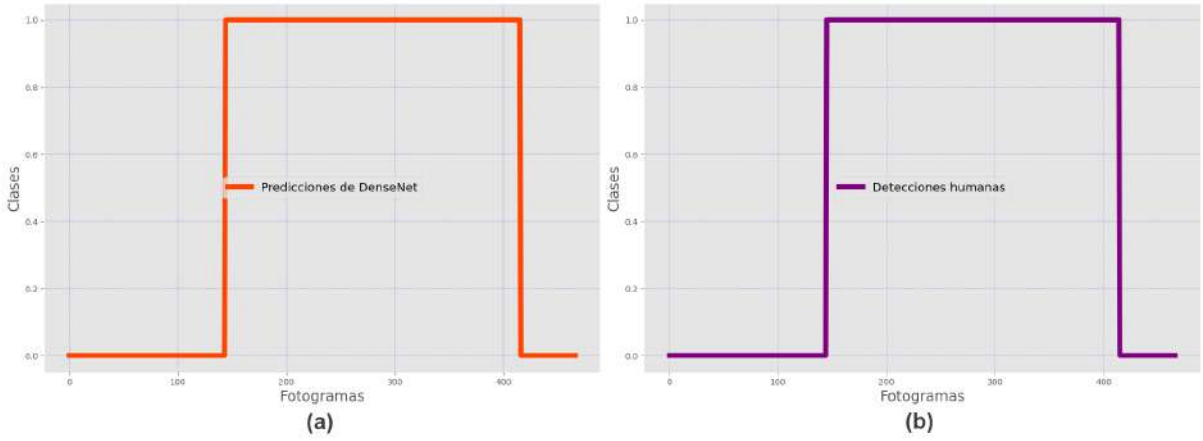

Figura 6.7: (a) Predicciones de la DenseNet 121 (b) Detecciones realizadas por un operador humano
Fuente: Autor

Prosiguiendo con las pruebas, se planteo que pasaría en el peor de los casos usando una arquitectura con menor desempeño para la detección como lo seria InceptionV3 y se puede denotar en la Figura 6.8.

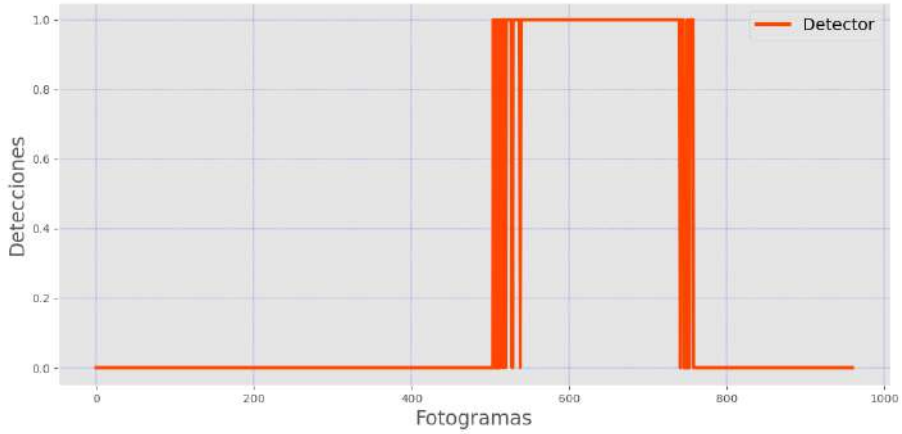

Figura 6.8: Gráfica de detección en vídeo captado por el dron con la arquitectura InceptionV3
Fuente: Autor

Cuando el dron empieza a entrar o salir dentro de la zona de visualización cercana a una torre eléctrica de alta tensión se obtienen detecciones incorrectas, resultado de no poder lograr un 100% precisión en la detección ante cualquier caso posible, tomando este caso se dice que siempre van a existir detecciones erróneas entre el lapso de transición en la detección de la infraestructura generando ruido de alta frecuencia en la salida del detector.

Para resolver esta problemática se realizo un muestreo de datos equivalente a 30 fotogramas lo que seria equivalente a un muestreo de 1 segundo en la detección, para aplicarle un intervalo modal tomando la mayor frecuencia entre los datos de muestreo, al obtener valores binarios en la detección la resultante es una señal filtrada con las detecciones correctas para la segmentación del vídeo. Como se muestra en la Figura 6.9.

Figura 6.9: Gráfica de detección en vídeo captado por el dron con la arquitectura InceptionV3 con un filtro de intervalo modal

Fuente: Autor

Este intervalo modal usado como filtro en la detección se implementó en la arquitectura de la DenseNet 121 para hacer el sistema de inspección mucho más robusto y lograr detecciones constantes en el proceso de segmentación de los vídeos de inspección sobre infraestructuras eléctricas.

El sistema de detección de infraestructura eléctrica fue usado en múltiples plataformas o sistemas operativos para comprobar su correcto funcionamiento en los sistemas operativos de la distribución Linux y Windows. Como se ven en las Figuras 6.10 y 6.11.

Figura 6.10: Sistema de detección de infraestructura eléctrica usando el sistema operativo Linux
Fuente: Autor

Adicionalmente en la ventana de vídeo de demostración del sistema de detección en la ubicación superior izquierda se puede encontrar resultados de predicción y tiempo de ejecución por fotograma.

Figura 6.11: Sistema de detección de infraestructura eléctrica usando el sistema operativo Windows
Fuente: Autor

6.1.4. Ejecución de algoritmo de segmentación de videos usando interfaz gráfica

Para hacer el proceso de separación de videos para un operador humano, se realizo una interfaz gráfica sencilla y amigable con el usuario, desarrollada con la librería de desarrollo Tkinter. La visualización inicial de la interfaz con el usuario se visualiza en la Figura 6.12.

Figura 6.12: Interfaz gráfica
Fuente: Autor

En la interfaz gráfica el botón de "Insert video" se usa para buscar la ruta donde se encuentra ubicado el video como se muestra en la Figura 6.13.

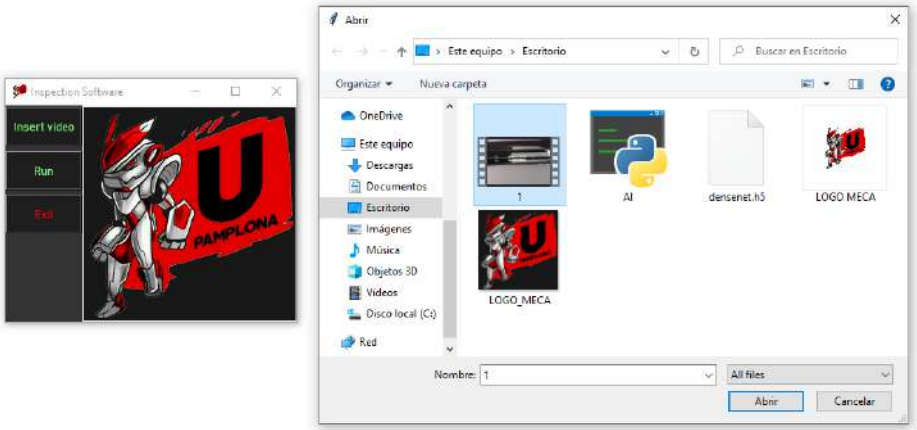

Figura 6.13: Interfaz gráfica usando el botón "Insert video"
Fuente: Autor

Ya insertado el video se procede usar el botón de ”Run”. El botón de ”Run” es el botón encargado de ejecutar todo el algoritmo de separación, donde el algoritmo se emplea sobre el video insertado.

Una vez pulsado el botón ”Run” se crea automáticamente una carpeta llamada ”Video”, la cual contendrá los videos separados con el algoritmo como se ve en la Figura 6.14.

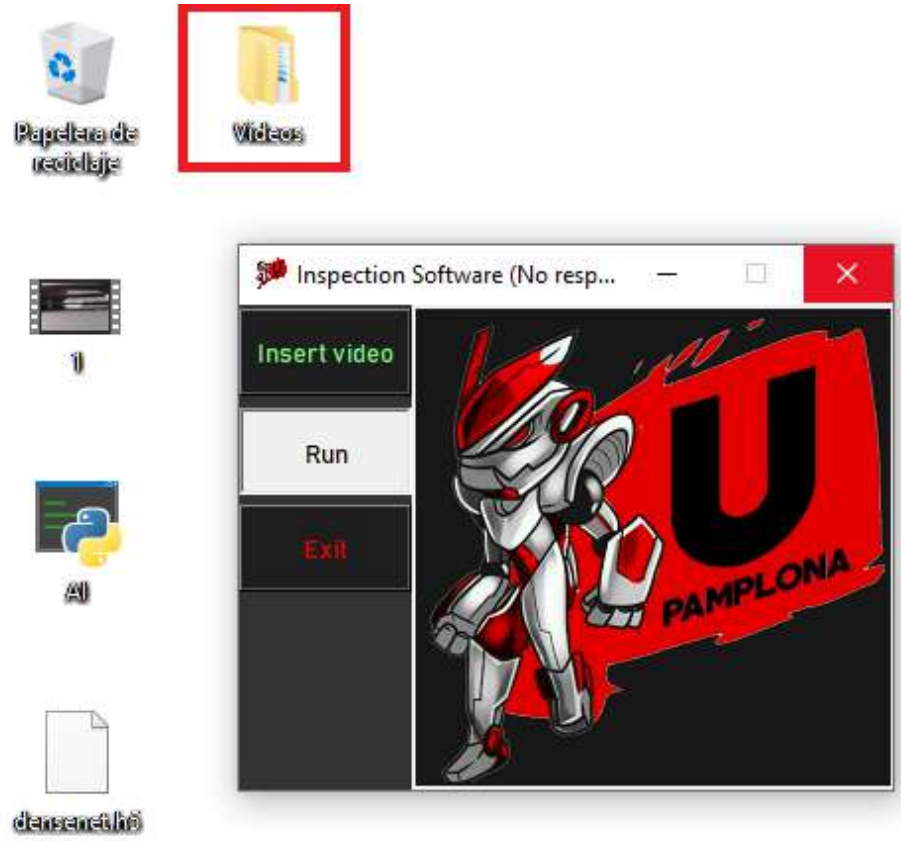

Figura 6.14: Interfaz gráfica usando el botón ”Run”
Fuente: Autor

Una vez terminada la ejecución del algoritmo la interfaz gráfica indica que el algoritmo ha realizado correctamente el proceso de separación de videos con un mensaje de ”Done!” como se muestra en la Figura 6.15.

Figura 6.15: Interfaz gráfica mostrando la correcta ejecución del algoritmo
Fuente: Autor

Mientras tanto en la carpeta de "Videos" se generarán dos carpetas. Cada una de las carpetas generadas corresponde al tipo de infraestructura detectada dentro del video como se muestra en la Figura 6.16.

Figura 6.16: Carpetas que contienen los videos de inspección sobre líneas y torres eléctricas
Fuente: Autor

6.2. Sistema de detección de aisladores eléctricos

En esta sección se presentan los resultados con respecto al sistema de detección de aisladores eléctricos usando el algoritmo de detección YOLO.

El IOU o intersección sobre la unión indica el grado de acoplamiento que existe entre los recuadros generados y los objetos detectados dentro del mapa de probabilidades mencionado en la sección 3.4.

Para medir el parámetro de velocidad de ejecución, el algoritmo se utilizó la tasa de detecciones en fotogramas por unidad de segundo, también conocida como FPS por sus siglas en inglés (Frames Per Second). Los resultados de entrenamiento de la red encuentra en las tablas 6.2 y 6.3.

Arquitectura	Precisión Media	IOU	FPS
Full YOLO	74.24%	53.67%	0.42
InceptionV3	NA	NA	NA
MobileNet	NA	NA	NA

Tabla 6.2: Resultados de los entrenamientos de los diferentes backends usando una tasa de aprendizaje de 1e-03

El resultado de NA hace referencia a que no se consiguieron resultados concluyentes, debido a que las arquitecturas no detectaron ningún aislador eléctrico dentro de las imágenes encontradas en el *dataset* de testeo.

Arquitectura	Precisión Media	IOU	FPS
Full YOLO	96.97%	54.78%	0.48
InceptionV3	95.25%	54.39%	0.54
MobileNet	97.19%	58.91%	1.03

Tabla 6.3: Resultados de los entrenamientos de los diferentes backends usando una tasa de aprendizaje de 1e-04

6.2.1. Gráficas de Características Operativas del Receptor (ROC)

Una forma común de evaluar los resultados de los experimentos de Machine Learning es usar *Recall* y *Precision*. Estas medidas llevan el nombre de su origen en la recuperación de información y presentan sesgos específicos. Los sesgos permiten medir el rendimiento al manejar correctamente los ejemplos negativos, propagan las prevalencias y sesgos marginales subyacentes y no tienen en cuenta el rendimiento del nivel de probabilidad (Powers y Ailab, 2011).

En las ciencias médicas, el análisis de las características operativas del receptor (ROC) se tomó prestado del procesamiento de señales para convertirse en un estándar para la evaluación en sistemas de detección, comparando la tasa positiva verdadera y la tasa positiva falsa. En las ciencias del comportamiento, la especificidad y la sensibilidad son las más usadas (Powers y Ailab, 2011).

Para la detección de los aisladores eléctricos usando el sistema de inteligencia artificial YOLO, se consiguieron los resultados mostrados en la siguiente tabla relacionados al desempeño de la sistema de detección. La curva de características operativas del receptor nos permite saber acerca del rendimiento de un sistema de detección en una serie de umbrales de datos (Muschelli, 2019). La curva de las características operativas del receptor se puede resumir en un solo valor normalizado conocido como el área bajo la curva o AUC.

El AUC toma un valor entre 0 y 1, donde el valor de 1 indica que todas las detecciones realizadas fueron correctas. El desempeño del sistema de inteligencia artificial usando la gráfica de desempeño de las características operativas del receptor se muestra en la Figura 6.17. Donde a su esquina inferior derecha se encuentran los valores AUC para cada arquitectura implementada en el *dataset* de testeo.

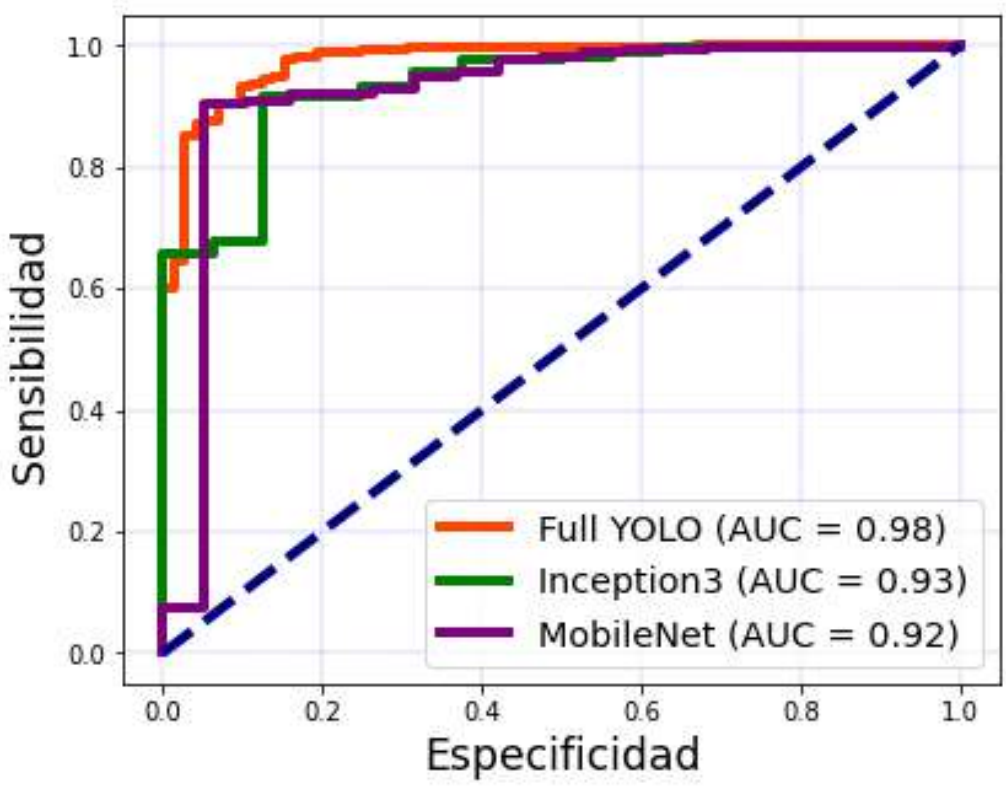

Figura 6.17: Curva de sensibilidad vs especificidad usando una tasa de aprendizaje de 1e-04
Fuente: Autor

La curva de sensibilidad vs especificidad muestra el crecimiento en la detección de falsos positivos con respecto a los verdaderos positivos en una escala normalizada mostrando un mejor desempeño en la arquitectura Full YOLO tomando en cuenta el parámetro AUC.

Aunque el desempeño de la arquitectura Full YOLO muestre un resultado creciente conforme a las detecciones realizadas en la Figura 6.17, en cuanto a valores numéricos en su precisión presenta un comportamiento diferente. Como se muestra en la tabla 6.3 una alta precisión en la detección con la arquitectura MobileNet y se afirma con la gráfica 6.18, donde se muestra el desempeño de cada una de sus arquitecturas con respecto a su precisión evaluado en el *dataset* de testeo.

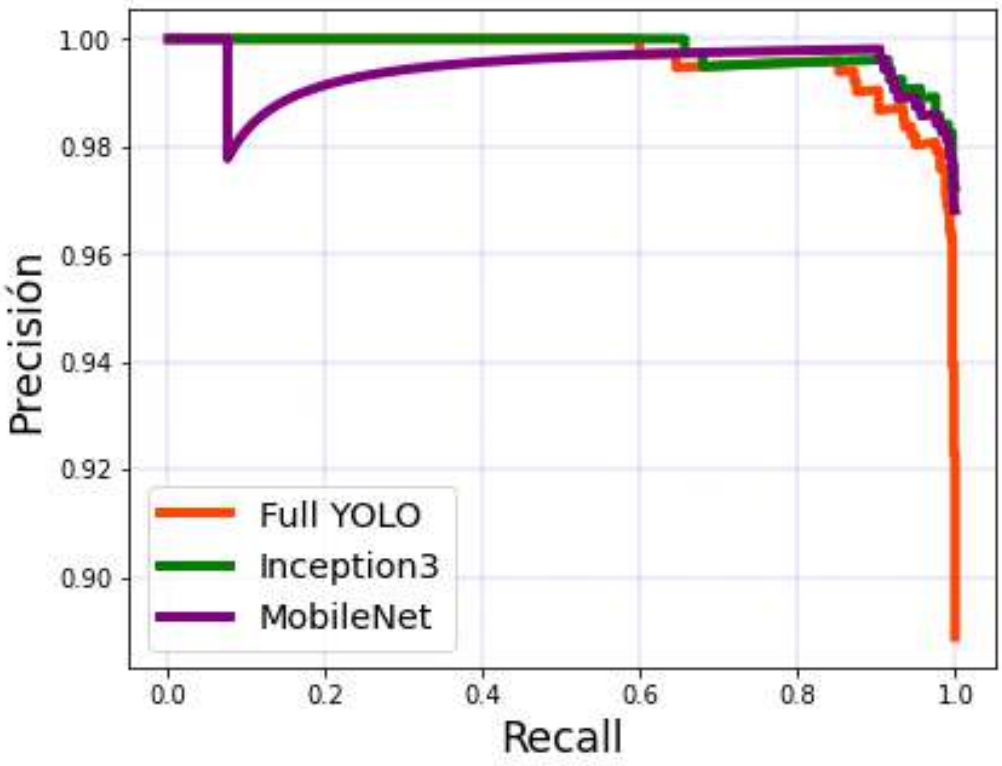

Figura 6.18: Curva de precisión vs recall usando una tasa de aprendizaje de 1e-04
Fuente: Autor

La curva de precisión vs recall presenta la caída en la precisión en la detección de aisladores eléctricos usando la arquitectura Full YOLO a diferencia de las arquitecturas InceptionV3 y MobileNet.

6.3. Detecciones de aisladores con la arquitectura MobileNet

Ya implementado el algoritmo de YOLO sobre uno de los videos de detección se pueden ver en la Figura 6.19 donde el porcentaje mostrado a en imagen representa las probabilidades de ser un aislador eléctrico. En Anexo III se pueden ver más implementaciones.

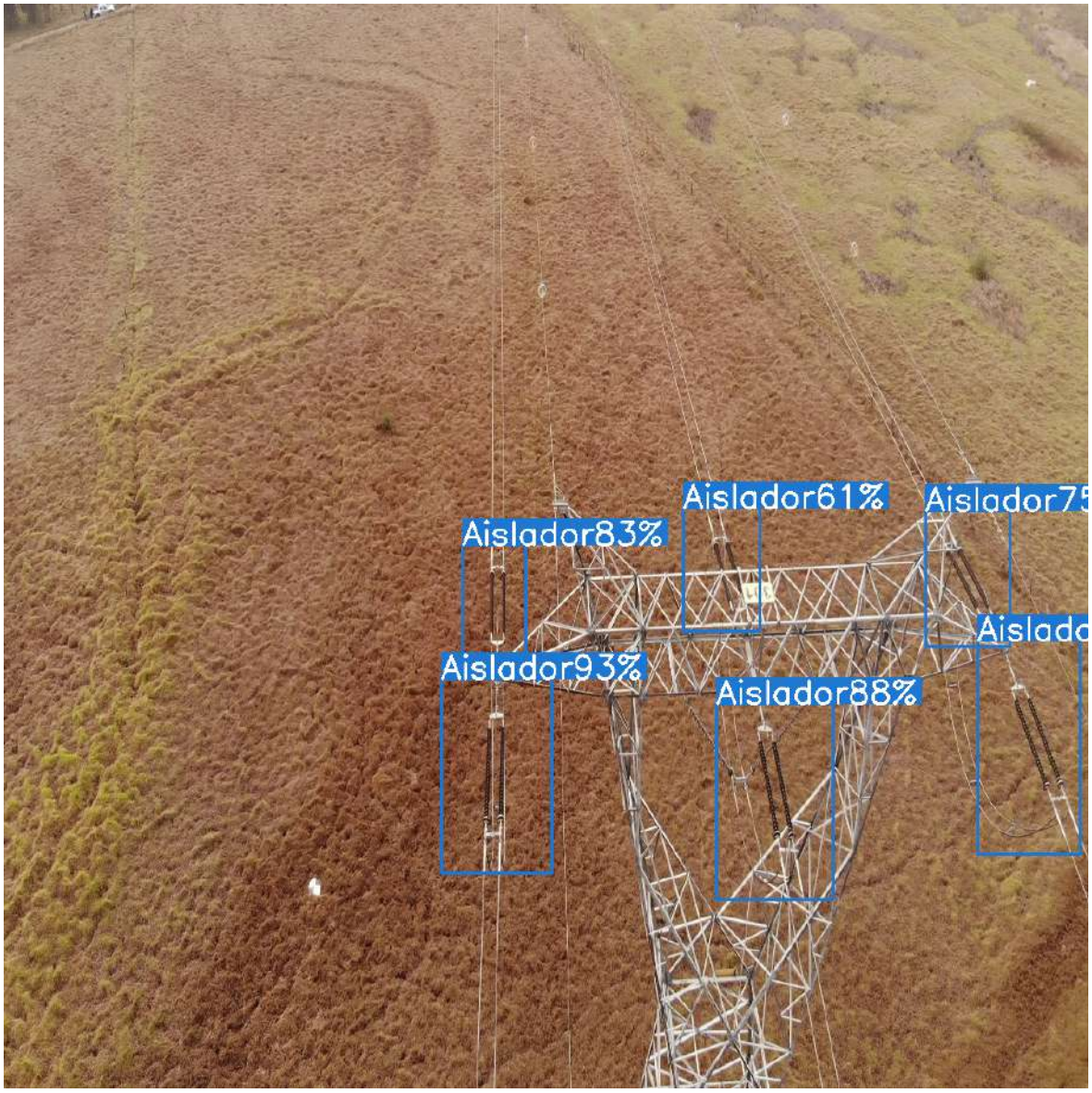

Figura 6.19: Aplicación del algoritmo de detección de aisladores eléctricos
Fuente: Autor

Realizadas detecciones se procede a tomar una imagen con las detecciones y extraer las imágenes correspondientes a los aisladores eléctricos como se ve en la Figura 6.20.

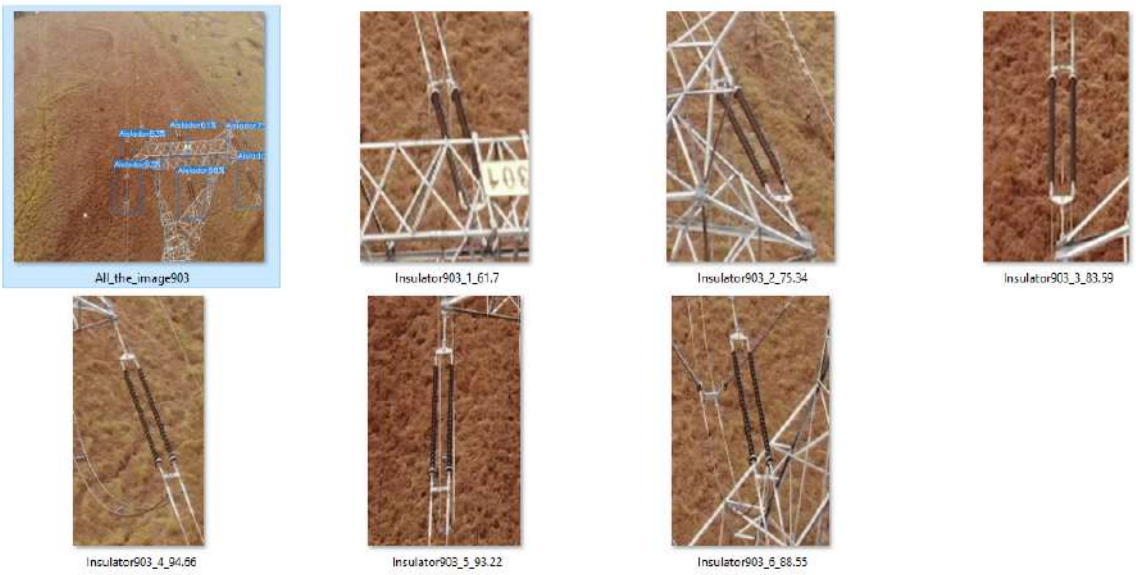

Figura 6.20: Archivos generados por el algoritmo de detección
Fuente: Autor

Para hacer la correcta visualización de los aisladores eléctricos se utilizo un escalado hasta de 2 veces el tamaño de la imagen usando un interpolador cúbico por su excelente desempeño en cuanto al redimensionamiento de imágenes a tal punto de ser utilizado como base en los software de edición de imágenes y videos (D. Han, 2013). El resultado de la extracción de los aisladores eléctricos en imágenes se puede ver en la Figura 6.21.

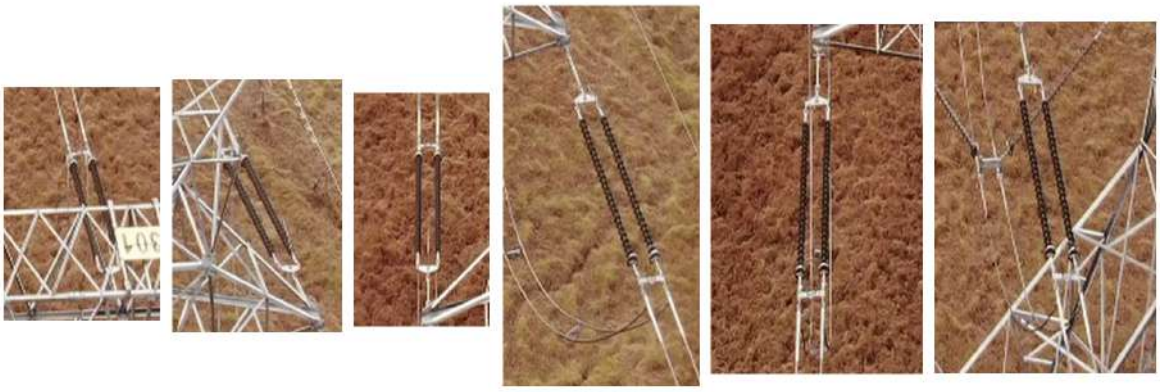

Figura 6.21: Imágenes extraídas por el algoritmo de detección
Fuente: Autor

7. Conclusiones

La investigación llevada a cabo presenta los algoritmos de inteligencia artificial como una gran ayuda para facilitar procesos de inspección en líneas de transmisión en el sector de la industria de las energías eléctricas.

Se generó un conjunto de datos completamente equilibrado basado en el número de cuadros que se podían extraer de los videos y con él una asignación de clase para el proceso de detección de torres eléctricas a una distancia de inspección cercana.

Se determino la efectividad de los algoritmos de inteligencia artificial para el proceso de segmentación de videos capturados por drones. Se logró extraer con éxito los videos de tipos de infraestructuras eléctricas más relevantes para realizar un proceso de inspección detallado.

Cuando el sistema de detección de lineas y torres eléctricas se ejecuta en dos sistemas operativos diferentes, existe una diferencia entre el tiempo de ejecución del algoritmo por cuadro detectado en el video. Se encuentra que el algoritmo se ejecuta más rápido en el sistema operativo Linux que en el sistema operativo Windows con una diferencia cercana a los 100 milisegundos.

A pesar de tener la menor complejidad computacional, DenseNet demostró ser la mejor arquitectura de red neuronal convolucional por sus excelentes resultados en términos de especificidad y sensibilidad en comparación con los otros modelos propuestos para la detección de lineas y torres eléctricas.

Algoritmos de inteligencia artificial creados hasta el momento no logran los resultados concluyentes para realizar un proceso de inspección en aisladores eléctricos de manera automática (J. Han y cols., 2019; Li y cols., 2018). Por otro lado, algoritmos de inteligencia artificial sí pueden llegar a ayudar en el proceso de inspección de líneas de transmisión eléctricas a supervisores de las compañías encargadas de la distribución de la energía eléctrica.

La arquitectura de redes convolucionales MobileNet logró un excelente desempeño en la extracción de características para la detección de aisladores eléctricos en redes de transmisión eléctrica. Se extrajeron imágenes de aisladores eléctricos usando un algoritmo de inteligencia artificial permitiendo una visualización cercana de los elementos eléctricos contenidos en imágenes de líneas de transmisión eléctrica. La extracción de las imágenes de los aisladores eléctricos permite al supervisor realizar un fácil proceso de inspección de las líneas de transmisión eléctrica.

Implementaciones a futuro...

Algoritmos de inteligencia artificial están siendo recientemente investigados para obtener imágenes con super resolución lo que implicaría una mejor visualización en los elementos de una imagen (Ledig y cols., 2017). Uno de los objetivos a futuro seria implementar algoritmos de inteligencia artificial para lograr obtener imágenes con super resolución para una visualización más profesional de los elementos que componen las líneas de transmisión eléctricas. Una comparación de estos métodos de escalado se pueden visualizar en la Figura 7.1.

Figura 7.1: De izquierda a derecha: interpolación bicúbica, red residual profunda optimizada para MSE, red adversa generativa residual profunda optimizada para una pérdida más sensible a la percepción humana , imagen original de recursos humanos. (Un escalado hasta 4 veces el tamaño original)

Fuente: Ledig y cols.

Recientemente se han realizado detecciones de aisladores eléctricos defectuosos usando algoritmos de inteligencia artificial. Los algoritmos de inteligencia artificial presentan errores en la detección de defectos en aisladores eléctricos de alta tensión como se muestra en la imagen a la derecha de la Figura 7.2 (Li y cols., 2018). Se postula continuar con la investigación de redes neuronales convolucionales para realizar sistemas de inteligencia artificial que logren una mayor precisión en la detección fallas en aisladores eléctricos de alta tensión.

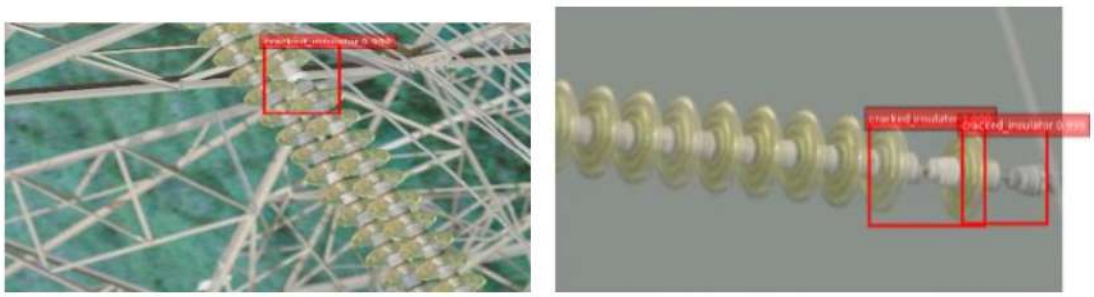

Figura 7.2: Ejemplos de detecciones realizadas por sistemas de inteligencia artificial

Fuente: Li y cols.

Bibliografía

Abiodun, O., Jantan, A., Omolara, O., Dada, K., Mohamed, N., y Arshad, H. (2018, 11). State-of-the-art in artificial neural network applications: A survey. *Heliyon*, *4*, e00938. doi: 10.1016/j.heliyon.2018.e00938

Albawi, S., Abed Mohammed, T., y ALZAWI, S. (2017, 08). Understanding of a convolutional neural network.. doi: 10.1109/ICEngTechnol.2017.8308186

Alex, K., Ilya, S., y Hg, E. (2012, 01). Imagenet classification with deep convolutional neural networks. *Proceedings of NIPS, IEEE, Neural Information Processing System Foundation*, 1097-1105.

Al-Imam, A., Motyka, M., y Jędrzejko, M. (2020, 05). Conflicting opinions in connection with digital superintelligence. *International Journal of Artificial Intelligence*, *9*, 336-348. doi: 10.11591/ijai.v9.i2.pp336-348

Alom, M. Z., Taha, T., Yakopcic, C., Westberg, S., Sidike, P., Nasrin, M., … Asari, V. (2019, 03). A state-of-the-art survey on deep learning theory and architectures. *Electronics*, *8*, 292. doi: 10.3390/electronics8030292

B, J., Periasamy, P., y Chenniappan, V. (2018). *Electric power transmission and distribution - a teacher's guide.*

Bairouk, A. (2019, 11). Improving convolutional neural network accuracy using gabor filter and progressive resizing..

Berk, R., y Elzarka, A. (2019, 10). Almost politically acceptable criminal justice risk assessment. *arXiv.*

Bertoni, F., Citti, G., y Sarti, A. (2019, 11). Lgn-cnn: a biologically inspired cnn architecture. *arXiv.*

Bianco, S., Cadène, R., Celona, L., y Napoletano, P. (2018, 10). Benchmark analysis of representative deep neural network architectures. *IEEE Access*, *6*, 64270-64277. doi: 10 .1109/ACCESS.2018.2877890

Cong, I., Choi, S., y Lukin, M. (2019, 12). Quantum convolutional neural networks. *Nature Physics*, *15*, 1-6. doi: 10.1038/s41567-019-0648-8

Dumoulin, V., y Visin, F. (2016, 03). A guide to convolution arithmetic for deep learning. *arXiv.*

Earp, G., Eyre-Walker, R., Ellam, A., y Thomas, A. (2011). Advanced aerial inspection and asset management of electricity towers. En *2011 ieee pes 12th international conference on transmission and distribution construction, operation and live-line maintenance (esmo)* (p. 1-7).

Eibar, C., Gavilane, J., Pérez, M., Aldás, ., Proaño, P., y Basantes, D. (2018, 01). Design and simulation of a mobile robot, for the inspection of electrical lines. *KnE Engineering*, *1*, 141. doi: 10.18502/keg.v1i2.1491

Feng, P., Lin, Y., Guan, J., Dong, Y., He, G., Xia, Z., y Shi, H. (2019, 07). Embranchment cnn based local climate zone classification using sar and multispectral remote sensing data. En (p. 6344-6347). doi: 10.1109/IGARSS.2019.8898703

Ghorbanzadeh, P., Taleshmekaeil, D., Shaddeli, A., y Kianpour, N. (2015, 01). Using one hot residue..

Han, D. (2013, 03). Comparison of commonly used image interpolation methods. *Proceedings of the 2nd International Conference on Computer Science and Electronics Engineering*. doi: 10.2991/iccsee.2013.391

Han, J., Yang, Z., Zhang, Q., Chen, C., Li, H., Lai, S., … Chen, R. (2019, 05). A method of insulator faults detection in aerial images for high-voltage transmission lines inspection. *Applied Sciences*, *9*, 2009. doi: 10.3390/app9102009

He, K., Zhang, X., Ren, S., y Sun, J. (2015, 12). Deep residual learning for image recognition. *arXiv*, *7*.

He, K., Zhang, X., Ren, S., y Sun, J. (2016, June). Deep residual learning for image recognition. En *The ieee conference on computer vision and pattern recognition (cvpr).*

Hendry, y Chen, R.-C. (2019, 05). Automatic license plate recognition via sliding-window darknet-yolo deep learning. *Image and Vision Computing*, *87*. doi: 10.1016/j.imavis.2019.04.007

Konstantinidis, D., Argyriou, V., Stathaki, T., y Nikos, G. (2019, 11). A modular cnn-based building detector for remote sensing images. *Computer Networks*, *168*, 107034. doi: 10.1016/j.comnet.2019.107034

Ledig, C., Theis, L., Huszar, F., Caballero, J., Cunningham, A., Acosta, A., … Shi, W. (2017, 07). Photo-realistic single image super-resolution using a generative adversarial network. En (p. 105-114). doi: 10.1109/CVPR.2017.19

Li, S., Zhou, H., Wang, G., Zhu, X., Kong, L., y Hu, Z. (2018, 08). Cracked insulator detection based on r-fcn. *Journal of Physics: Conference Series*, *1069*, 012147. doi: 10.1088/1742-6596/1069/1/012147

Lowe, B., y Kulkarni, A. (2015, 02). Multispectral image analysis using random forest. *International Journal on Soft Computing*, *6*, 1-14. doi: 10.5121/ijsc.2015.6101

Martin T. Hagan, M. H. B. O. D. J., Howard B. Demuth. (2014). Biological inspiration. En *Neural network design* (cap. 1.1.8).

Morales Castro, E. . J. Z. R., Arturo Ramírez Reyes. (2020, Enero/Junio). Comportamiento del mercado de divisas: una aplicación de redes neuronales artificiales. *YACHANA*, *9*.

Muschelli, J. (2019, 12). Roc and auc with a binary predictor: a potentially misleading metric. *Journal of Classification*. doi: 10.1007/s00357-019-09345-1

Pernebayeva, D., Bagheri, M., y James, A. (2017). High voltage insulator surface evaluation using image processing. En *2017 international symposium on electrical insulating materials (iseim)* (Vol. 2, p. 520-523).

Powers, D., y Ailab. (2011, 01). Evaluation: From precision, recall and f-measure to roc, informedness, markedness correlation. *J. Mach. Learn. Technol*, *2*, 2229-3981. doi: 10 .9735/2229-3981

Redmon, J., Divvala, S., Girshick, R., y Farhadi, A. (2016, 06). You only look once: Unified, real-time object detection. En (p. 779-788). doi: 10.1109/CVPR.2016.91

Redmon, J., y Farhadi, A. (2017, 07). Yolo9000: Better, faster, stronger. En (p. 6517-6525). doi: 10.1109/CVPR.2017.690

Redmon, J., y Farhadi, A. (2018, 04). Yolov3: An incremental improvement. *arXiv*.

Saeedan, F., Weber, N., Goesele, M., y Roth, S. (2018, 06). Detail-preserving pooling in deep networks. En (p. 9108-9116). doi: 10.1109/CVPR.2018.00949

Sainath, T., Mohamed, A.-r., Kingsbury, B., y Ramabhadran, B. (2013, 05). Deep convolutional neural networks for lvcsr. En (p. 8614-8618). doi: 10.1109/ICASSP.2013.6639347

Sampedro Pérez, C., Martinez, C., Chauhan, A., y Campoy, P. (2014, 07). A supervised approach to electric tower detection and classification for power line inspection.. doi: 10.1109/IJCNN.2014.6889836

Shengyu, L., Wang, B., Wang, H., Chen, L., Linjian, M., Zhang, y Xiaoyan. (2019, 07). A real-time object detection algorithm for video. *Computers and Electrical Engineering*, *77*, 398-408. doi: 10.1016/j.compeleceng.2019.05.009

Simonyan, K., y Zisserman, A. (2014, 09). Very deep convolutional networks for large-scale image recognition. *arXiv 1409.1556*.

Sohn, K., Jung, D., Lee, H., y Hero, A. (2011, 11). Efficient learning of sparse, distributed, convolutional feature representations for object recognition. En (p. 2643-2650). doi: 10 .1109/ICCV.2011.6126554

Sugata, T., y Yang, C. (2017, 11). Leaf app: Leaf recognition with deep convolutional neural networks. *IOP Conference Series: Materials Science and Engineering*, *273*, 012004. doi: 10.1088/1757-899X/245/1/012004

Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., … Rabinovich, A. (2014, 09). Going deeper with convolutions.

Szegedy, C., Vanhoucke, V., Ioffe, S., Shlens, J., y Wojna, Z. (2016, 06). Rethinking the inception architecture for computer vision.. doi: 10.1109/CVPR.2016.308

Tao, X., Zhang, D., Wang, Z., Liu, X., Zhang, H., y Xu, D. (2018). Detection of power line insulator defects using aerial images analyzed with convolutional neural networks. *IEEE Transactions on Systems, Man, and Cybernetics: Systems*.

Zeiler, M., y Fergus, R. (2013, 01). Visualizing and understanding convolutional networks. *European Conference on Computer Vision(ECCV)*, *8689*, 818-833.

Nota: El método de citación usado fue basado en las normas APA. Las referencias en texto se puede visualizar como: (AutorPrincipal y cols., AñoDePublicación). Donde cols hace referencia a colegas, refiriéndose al resto de autores del material bibliográfico.

Lista de Acrónimos y Abreviaturas

ANN	Redes Neuronales artificiales - *Artificial Neural Networks.*
CNN	*Convolutional neural networkr.*
CNNs	Redes Neuronales Convolucionales - *Convolutional neural networkrs.*
FP	Falsos Positivos - *False Positive.*
IA	Inteligencia Artificial.
ROC	Características operativas del receptor - *Receiver Operating Characteristic.*
TP	Verdaderos Positivos - *True Positive.*
YOLO	*You Only Look Once.*

A. Anexo I: Imágenes tomadas por drones sobre líneas eléctricas de alta tensión

Imágenes de lineas de alta tensión tomadas usando drones a través del semillero de investigación SIRP

Figura A.1: Imágenes de lineas de alta tensión tomadas usando drones para visualización de torres eléctricas

Fuente: Semillero de investigación en sistemas inteligentes, robótica y percepción

Figura A.2: Imágenes de lineas de alta tensión tomadas usando drones para visualización de torres eléctricas desde una vista superior

Fuente: Semillero de investigación en sistemas inteligentes, robótica y percepción

Figura A.3: Imágenes de lineas de alta tensión tomadas usando drones para visualización de torres eléctricas de manera lejana

Fuente: Semillero de investigación en sistemas inteligentes, robótica y percepción

B. Anexo II: Etiquetado de aisladores eléctricos usando el software LabelImg

Visualizaciones de las imágenes a las que fueron aplicadas una modificación en las etiquetas para la detección de aisladores eléctricos y etiquetado del dataset tomado en la ciudad de Bogotá se pueden encontrar en esta sección. En las siguientes figuas se encuentran las diferentes vistas para la visualización de imágenes del dataset para la detección de aisladores eléctricos verticales usando el software LabelImg

Figura B.1: Visualización de imágenes del dataset para la detección de aisladores eléctricos verticales usando el software LabelImg (Vista 1)

Fuente: Autor

Figura B.2: Visualización de imágenes del dataset para la detección de aisladores eléctricos verticales usando el software LabelImg (Vista 2)

Fuente: Autor

Figura B.3: Visualización de imágenes del dataset para la detección de aisladores eléctricos verticales usando el software LabelImg (Vista 3)

Fuente: Autor

Figura B.4: Visualización de imágenes del dataset para la detección de aisladores eléctricos verticales usando el software LabelImg (Vista 4)

Fuente: Autor

Figura B.5: Visualización de imágenes del dataset para la detección de aisladores eléctricos verticales usando el software LabelImg (Vista 5)

Fuente: Autor

Figura B.6: Visualización de imágenes del dataset para la detección de aisladores eléctricos verticales usando el software LabelImg (Vista 6)

Fuente: Autor

Figura B.7: Visualización de la imagen 1 del dataset de infraestructuras eléctricas tomada en la ciudad de Bogotá para la detección de aisladores eléctricos usando el software LabelImg

Fuente: Autor

Figura B.8: Visualización de la imagen 2 del dataset de infraestructuras eléctricas tomada en la ciudad de Bogotá para la detección de aisladores eléctricos usando el software LabelImg
Fuente: Autor

Figura B.9: Visualización de la imagen 3 del dataset de infraestructuras eléctricas tomada en la ciudad de Bogotá para la detección de aisladores eléctricos usando el software LabelImg
Fuente: Autor

C. Anexo III: Visualización de las detecciones realizadas por el algoritmo YOLO

Visualizaciones de la aplicación del algoritmo de inteligencia artificial para la extracción de aisladores eléctricos sobre imágenes de infraestructuras eléctricas se pueden encontrar en esta sección. Las siguientes figuras presentan las diferentes detecciones en la aplicación del algoritmo de detección de aisladores eléctricos.

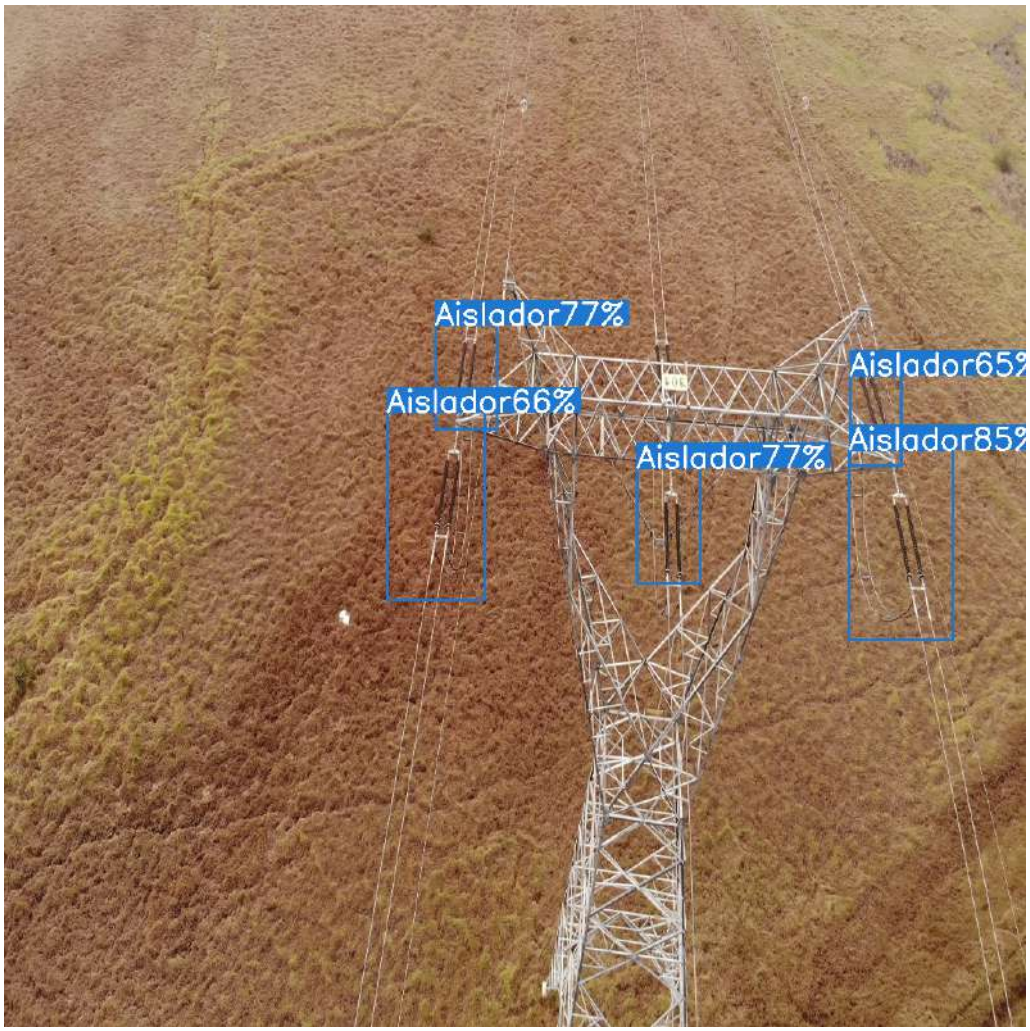

Figura C.1: Aplicación del algoritmo de detección de aisladores eléctricos (Detección 1)
Fuente: Autor

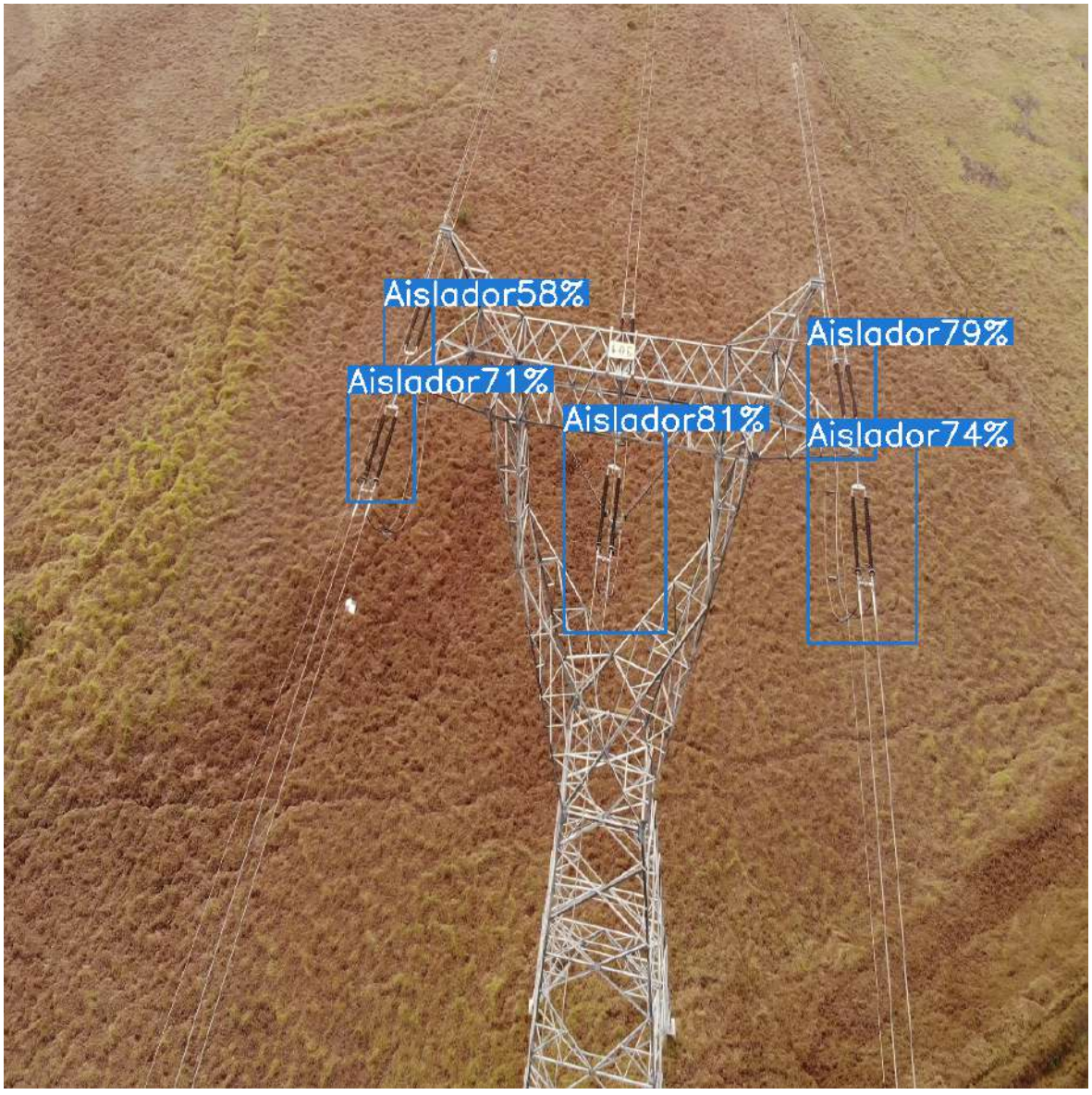

Figura C.2: Aplicación del algoritmo de detección de aisladores eléctricos (Detección 2)
Fuente: Autor

Figura C.3: Aplicación del algoritmo de detección de aisladores eléctricos (Detección 3)
Fuente: Autor

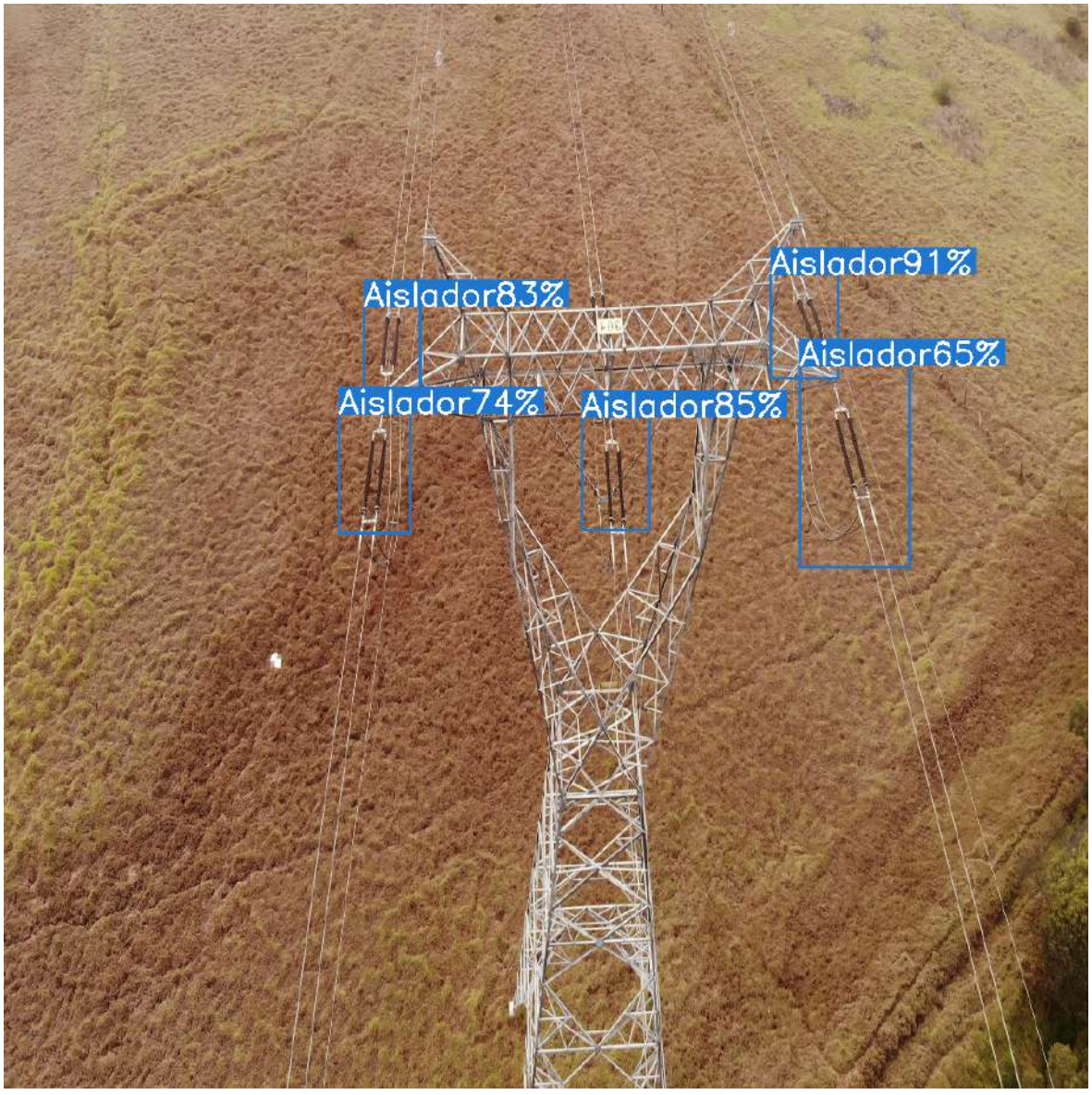

Figura C.4: Aplicación del algoritmo de detección de aisladores eléctricos (Detección 4)
Fuente: Autor

D. Anexo IV: Códigos implementados para la inspección de líneas eléctricas de alta tensión

Este capítulo de anexos presenta las secciones mas relevantes de los códigos implementados para la inspección de líneas eléctricas de alta tensión.

D.1. Conjuntos de datos en el algoritmo de redes neuronales convolucionales

En el código D.1 se muestra el algoritmo para extraer las imágenes de los videos para crear el conjunto de datos.

Código D.1: Código para extraer las imágenes de los videos para crear el conjunto de datos en el algoritmo de redes neuronales convolucionales

```
import cv2
import vlc
import time

formato = '.jpg'
name = 0
tipo = 'WithoutInspecting_'

for i in range(0,6): #6 videos

    cap = cv2.VideoCapture(str(i)+'.MP4')

    path = 'Prueba1/'
    [rec, camara] = cap.read()

    while(rec == 1):
        [rec, camara] = cap.read()
        name = name + 1

        if (rec ==1):
            file = str(tipo) + str(name) + formato
            camara = cv2.resize(camara, None, fx = 0.40, fy= 0.40)

            cv2.imshow('Camara',camara)

            # Guarda la imagen tomada
            cv2.imwrite(path + file, camara)

            if name%100 == 0:
                print("Imagen numero: " + str(name))

        if cv2.waitKey(1) & 0xFF == ord('q'):
            break

```

```
        if cv2.waitKey(1) & 0xFF == ord('c'):
            print('Inspecting_thetower')
            path = 'Inspecting_thetower/'
            tipo = 'Inspecting_'
            plays = vlc.MediaPlayer("Beep_Short.mp3")
            plays.play()
            time.sleep(1)

        if cv2.waitKey(1) & 0xFF == ord('v'):
            print('WithoutInspecting_lines')
            path = 'WithoutInspecting_lines/'
            tipo = 'WithoutInspecting_'
            plays = vlc.MediaPlayer("Beep_Short.mp3")
            plays.play()
            time.sleep(1)

    plays = vlc.MediaPlayer("Beep_Short.mp3")
    plays.play()
    time.sleep(1)

    cap.release()
    cv2.destroyAllWindows()

plays = vlc.MediaPlayer("Beep_Short.mp3")
plays.play()
time.sleep(2)
```

En el código D.2 se muestra el algoritmo creado para crear el conjunto de datos en un archivo .hdf5 para el entrenamiento de redes neuronales artificiales.

Código D.2: Código para generar el conjuntos de datos en el algoritmo de redes neuronales convolucionales

```
from random import shuffle
import glob
import h5py
import numpy as np
import cv2

shuffle_data = True
hdf5_path = 'Inspection_320x240.hdf5'
train_path = 'Train/*'

addrs = glob.glob(train_path)

classes = []
labels = []
files = []
labelsNumbers = {}
indexlabel = 0

for subfolder in addrs:
    classes.append(subfolder)
    labelsNumbers[subfolder.split('_')[1]] = indexlabel
    indexlabel += 1
    files.append(glob.glob(subfolder+'/*'))

print(labelsNumbers)

addrs = [item for sublist in files for item in sublist]

for f in addrs:
    l = f.split('_')[0] #Nombre de Clase
    if l == 'Train/WithoutInspecting':
        h = 0
```

```
    if l == 'Train/Inspecting':
        h = 1
    labels.append(h)

if shuffle_data:
    c = list(zip(addrs, labels))
    shuffle(c)
    addrs, labels = zip(*c)

# Divide the data into 70% train, 15% validation, and 15% test
train_addrs = addrs[0:int(0.7 * len(addrs))]
train_labels = labels[0:int(0.7 * len(labels))]
val_addrs = addrs[int(0.7 * len(addrs)):int(0.85 * len(addrs))]
val_labels = labels[int(0.7 * len(addrs)):int(0.85 * len(addrs))]
test_addrs = addrs[int(0.85 * len(addrs)):]
test_labels = labels[int(0.85 * len(labels)):]

data_order = 'tf'

if data_order == 'tf':
    train_shape = (len(train_addrs), 240, 320, 3)
    val_shape = (len(val_addrs), 240, 320, 3)
    test_shape = (len(test_addrs), 240, 320, 3)

# open a hdf5 file and create earrays
hdf5_file = h5py.File(hdf5_path, mode='w')
hdf5_file.create_dataset("train_img", train_shape, np.uint8)
hdf5_file.create_dataset("val_img", val_shape, np.uint8)
hdf5_file.create_dataset("test_img", test_shape, np.uint8)
hdf5_file.create_dataset("train_labels", (len(train_addrs),), np.uint8)
hdf5_file["train_labels"][...] = train_labels
hdf5_file.create_dataset("val_labels", (len(val_addrs),), np.uint8)
hdf5_file["val_labels"][...] = val_labels
hdf5_file.create_dataset("test_labels", (len(test_addrs),), np.uint8)
hdf5_file["test_labels"][...] = test_labels

# loop over train addresses
for i in range(len(train_addrs)):
    # print how many images are saved every 500 images
    if i % 500 == 0 and i > 1:
        print('Train data: ' + str(i) + '/' + str(len(train_addrs)))

    # cv2 load images as BGR, convert it to RGB
    addr = train_addrs[i]
    img = cv2.imread(addr)
    img = cv2.resize(img, (320,240), interpolation=cv2.INTER_CUBIC)
    img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB) # cv2 load images as BGR, convert it to RGB
    try:
        img = img
        hdf5_file["train_img"][i, ...] = img[None]
    except Exception as e:
        pass

# loop over validation addresses
for i in range(len(val_addrs)):
    # print how many images are saved every 500 images
    if i % 500 == 0 and i > 1:
        print('Validation data: ' + str(i) + '/' + str(len(val_addrs)))

    # cv2 load images as BGR, convert it to RGB
    addr = val_addrs[i]
    img = cv2.imread(addr)
    img = cv2.resize(img, (320,240), interpolation=cv2.INTER_CUBIC)
    img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB) # cv2 load images as BGR, convert it to RGB
```

```
    try:
        img = img
        hdf5_file["val_img"][i, ...] = img[None]
    except Exception as e:
        pass

# loop over test addresses
for i in range(len(test_addrs)):
    # print how many images are saved every 500 images
    if i % 500 == 0 and i > 1:
        print('Test data: ' + str(i) + '/' + str(len(test_addrs)))

    # cv2 load images as BGR, convert it to RGB
    addr = test_addrs[i]
    img = cv2.imread(addr)
    img = cv2.resize(img, (320,240), interpolation=cv2.INTER_CUBIC)
    img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB) # cv2 load images as BGR, convert it to RGB
    try:
        img = img
        hdf5_file["test_img"][i, ...] = img[None]
    except Exception as e:
        pass

# save the mean and close the hdf5 file
cv2.destroyAllWindows()
hdf5_file.close()
```

D.2. Entrenamiento de algoritmos de inteligencia artificial

En el código D.3 se muestra el algoritmo de inteligencia artificial para la detección de infraestructura eléctrica usando la arquitectura DenseNet. Este código fue desarrollado sobre la plataforma llamada Google Colab para usar un sistema de GPU gratuito para el entrenamiento de redes neuronales artificiales.

Código D.3: Algoritmo de inteligencia artificial para la detección de infraestructura eléctrica usando la arquitectura DenseNet

```
!pip install h5py
import h5py

from google.colab import drive,files
drive.mount('/content/drive/')
import sys
sys.path.append('/content/drive/My Drive/Serbeld_Drone/DATASET/Dataset_320x240_and_Code')

hdf5_path = '/content/drive/My Drive/Serbeld_Drone/DATASET/Dataset_320x240_and_Code/Inspection_320x240.hdf5'

dataset = h5py.File(hdf5_path, "r")

!pip install tensorflow==1.15.0
!pip install keras==2.3.1

# Parameters
batch_size = 10
num_classes = 2
num_epochs = 10
lrate = 2e-5

import tensorflow as tf
import numpy as np
```

```
import os
from tensorflow.keras.utils import to_categorical
from tensorflow.keras import backend as k
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Dense,Dropout,Flatten,Input,AveragePooling2D
from tensorflow.keras.callbacks import TensorBoard, ModelCheckpoint
from tensorflow.keras.optimizers import SGD,Adam
from tensorflow.keras.applications import densenet
import matplotlib.pylab as plt

# Size
filas,columnas = 240,320
img_shape = (filas, columnas, 3)

#train
train_img = dataset["train_img"]

xt = np.array(train_img)
yt = np.array(dataset["train_labels"])

#test
xtest = np.array(dataset["test_img"])
ytest = np.array(dataset["test_labels"])

#Validation
xval = np.array(dataset["val_img"])
yval = np.array(dataset["val_labels"])

#Categorical values or OneHot
yt = to_categorical(yt,num_classes)
ytest = to_categorical(ytest,num_classes)
yval = to_categorical(yval,num_classes)

from keras.callbacks import ModelCheckpoint

#Inputs
inputs = Input(shape=img_shape, name='images')

#Nasnet Model
output = densenet.DenseNet121(include_top=False,weights=None,
                                    input_shape=img_shape,
                                    classes = num_classes)(inputs)


#AveragePooling2D
output = AveragePooling2D(pool_size=(2, 2), strides=None,
                          padding='valid',name='AvgPooling')(output)

#Flattened
output = Flatten(name='Flatten')(output)

#Dropout
output = Dropout(0.2,name='Dropout')(output)

#ReLU layer
output = Dense(10, activation = 'relu',name='ReLU')(output)

#Dense layer
output = Dense(num_classes, activation='softmax',name='softmax')(output)

#Checkpoint_path

# Create checkpoint callback
model_checkpoint = ModelCheckpoint(filepath="/content/drive/My Drive/Newdensenet/densenet.h5",
```

```
                                    monitor='val_loss', save_best_only=True)

#Model
modelo = Model(inputs=inputs, outputs=output)

ADAM = Adam(lr=lrate)
modelo.compile(loss='categorical_crossentropy',optimizer=ADAM,
               metrics=['categorical_accuracy'])

#Summary
modelo.summary()

#Training Model

stad = modelo.fit({'images': xt}, {'softmax': yt}, batch_size=batch_size,
          epochs=num_epochs, validation_data=(xval, yval), callbacks = [model_checkpoint])

modelo.save('/content/drive/My Drive/Newdensenet/densenet.h5')
```

algoritmo de segmentación de videos usando interfaz gráfica

D.3. Algoritmo de segmentación de videos usando interfaz gráfica

En el código D.4 se muestra el algoritmo de segmentación de videos usando interfaz gráfica. Para este código se uso la librería Tkinter como herramienta para generar una interfaz gráfica amigable con el usuario.

Código D.4: Algoritmo de segmentación de videos usando interfaz gráficas

```
import cv2
import numpy as np
import tkinter as tk
from easygui import *
from tensorflow.keras.models import load_model
from scipy import stats
import os

#Load the best model trained
model = load_model("densenet.h5")

root = tk.Tk()
root.title('Inspection Software')
root.iconbitmap('LOGO MECA.ico')
root.geometry("330x245")

def open_video():
    try:
        global name_file
    except:
        print()

    name_file = fileopenbox()

    try:
        global inp_img
        print(name_file)
    except:
        print(name_file)

def run_file():

```

```
    try:
        global salir
        salir = 0
    except:
        salir = 0
    try:
        cap = cv2.VideoCapture(name_file)

        [rec, camara] = cap.read()

        video = camara
        camara = video

        videofps = cap.get(cv2.CAP_PROP_FPS)
        four = cv2.VideoWriter_fourcc(*'MP4V')

        numero_frames = int(cap.get(cv2.CAP_PROP_FRAME_COUNT))
        activate_detection = 0
        detection = []

        filter2 = []
        filter1 = []

        frame_actual = 0
        recordingvideo = 1
        activate = 0
        cont = 0

        while (frame_actual < (numero_frames - 35)):

            activate_detection = 1
            # Toma parámetros de captura de la cámara
            [rec, camara] = cap.read()
            video = camara
            camara2 = cv2.resize(camara, (320, 240),
            interpolation=cv2.INTER_CUBIC)
            camara2 = cv2.cvtColor(camara2,
                  cv2.COLOR_BGR2RGB) # cv2 load images as BGR convert it to RGB

            if (frame_actual % 8) == 0:
                # ImagenInput
                inputoimage = camara2
                x = inputoimage.reshape((-1, 240, 320, 3))
                prediction = np.round(model.predict(x))
                filter1.append(round(prediction[0, 0]))
                filter2.append(round(prediction[0, 1]))

            if len(filter1) == 5:

                if (stats.mode(filter1)[0]) == [1.0]:
                    if activate_detection == 1:
                        detection.append(0)
                    activate = 0

                if (stats.mode(filter2)[0]) == [1.0]:
                    if activate_detection == 1:
                        detection.append(1)
                    activate = 1

                filter1 = []
                filter2 = []

            if recordingvideo == 1 and activate == 1:
                cont = cont + 1
```

```
97                os.makedirs('Videos/Torres_eléctricas/', exist_ok=True)
98                out_drone = cv2.VideoWriter(('Videos/Torres_eléctricas/' +
99                'Torre_' + str(cont) + '.mp4'),
100                     four, videofps, (int(video.shape[1]), int(video.shape[0])))
101               recordingvideo = 0
102               out_drone_line.release()
103               out_drone_line = None
104
105           if activate == 1:
106               out_drone.write(video)
107
108           if activate == 0:
109               try:
110                   out_drone.release()
111                   out_drone = None
112                   os.makedirs('Videos/Líneas_eléctricas/', exist_ok=True)
113                   out_drone_line = cv2.VideoWriter(('Videos/Líneas_eléctricas/'
114                   + 'Línea_eléctrica_' + str(cont) + '.mp4'),
115                         four, videofps, (int(video.shape[1]),
116                         int(video.shape[0])))
117                   recordingvideo = 1
118               except:
119                   recordingvideo = 1
120
121               try:
122                   out_drone_line.write(video)
123               except:
124                   os.makedirs('Videos/Líneas_eléctricas/', exist_ok=True)
125                   out_drone_line = cv2.VideoWriter(('Videos/Líneas_eléctricas/'
126                   + 'Línea_eléctrica_' + str(cont) + '.mp4'),
127                       four, videofps, (int(video.shape[1]),
128                       int(video.shape[0])))
129
130           if salir == 1:
131               break
132
133           frame_actual = frame_actual + 1
134
135       try:
136           out_drone.release()
137
138       except:
139           try:
140               out_drone_line.release()
141           except:
142               try:
143                   out_drone.release()
144
145               except:
146                   print('Nothing')
147
148       cap.release()
149
150       cv2.destroyAllWindows()
151
152       Label_porcentaje = tk.Label(root, text="Done!",
153       bg="#333333", fg="#80FF80", font="Bahnschrift 10")
154       Label_porcentaje.grid(row=3, column=0)
155
156   except:
157       Label_porcentaje = tk.Label(root, text='Insert video',
158       bg="#333333", fg="#E40000", font="Bahnschrift 10")
159       Label_porcentaje.grid(row=3, column=0)
160
```

```
#set window color
root['bg']= "#333333"

logo = tk.PhotoImage(file="LOGO_MECA.png")
fondo = tk.Label(root, image=logo)

button_open = tk.Button(root,
    text="Insert video", height=2, width=11, bg="#1E1E1E", fg="#80FF80",
    font="Bahnschrift 10", command=open_video)

button_Detect = tk.Button(root,
    text="Run", height=2, width=11, bg="#1E1E1E", fg="#80FF80",
     font="Bahnschrift 10", command=run_file)

button_exit = tk.Button(root,
    text="Exit", height=2, width=11, bg="#1E1E1E", fg="#E40000",
     font="Bahnschrift 10", command=root.quit)

button_open.grid(row=0, column=0)
button_Detect.grid(row=1, column=0)
button_exit.grid(row=2, column=0)
fondo.grid(row=0, column=1, rowspan=20)

root.mainloop()
```

D.4. Convoluciones de Full YOLO

En el código D.5 se presenta la función que contiene los bloques convolucionales de la arquitectura Full YOLO implementada por Allan Zelener con el uso de las librerías Tensorflow y Keras

Código D.5: Convoluciones de Full YOLO

```

class FullYoloFeature(BaseFeatureExtractor):

    def ___init___(self, input_size):
        input_image = Input(shape=(input_size, input_size, 3))

        # the function to implement the orgnization layer (thanks to github.com/allanzelener/YAD2K)
        def space_to_depth_x2(x):
            return tf.space_to_depth(x, block_size=2)

        # Layer 1
        x = Conv2D(32, (3,3), strides=(1,1), padding='same', name='conv_1', use_bias=False)(input_image↩
            ↪ )
        x = BatchNormalization(name='norm_1')(x)
        x = LeakyReLU(alpha=0.1)(x)
        x = MaxPooling2D(pool_size=(2, 2))(x)

        # Layer 2
        x = Conv2D(64, (3,3), strides=(1,1), padding='same', name='conv_2', use_bias=False)(x)
        x = BatchNormalization(name='norm_2')(x)
        x = LeakyReLU(alpha=0.1)(x)
        x = MaxPooling2D(pool_size=(2, 2))(x)

        # Layer 3
        x = Conv2D(128, (3,3), strides=(1,1), padding='same', name='conv_3', use_bias=False)(x)
        x = BatchNormalization(name='norm_3')(x)
        x = LeakyReLU(alpha=0.1)(x)

```

```
        # Layer 4
        x = Conv2D(64, (1,1), strides=(1,1), padding='same', name='conv_4', use_bias=False)(x)
        x = BatchNormalization(name='norm_4')(x)
        x = LeakyReLU(alpha=0.1)(x)

        # Layer 5
        x = Conv2D(128, (3,3), strides=(1,1), padding='same', name='conv_5', use_bias=False)(x)
        x = BatchNormalization(name='norm_5')(x)
        x = LeakyReLU(alpha=0.1)(x)
        x = MaxPooling2D(pool_size=(2, 2))(x)

        # Layer 6
        x = Conv2D(256, (3,3), strides=(1,1), padding='same', name='conv_6', use_bias=False)(x)
        x = BatchNormalization(name='norm_6')(x)
        x = LeakyReLU(alpha=0.1)(x)

        # Layer 7
        x = Conv2D(128, (1,1), strides=(1,1), padding='same', name='conv_7', use_bias=False)(x)
        x = BatchNormalization(name='norm_7')(x)
        x = LeakyReLU(alpha=0.1)(x)

        # Layer 8
        x = Conv2D(256, (3,3), strides=(1,1), padding='same', name='conv_8', use_bias=False)(x)
        x = BatchNormalization(name='norm_8')(x)
        x = LeakyReLU(alpha=0.1)(x)
        x = MaxPooling2D(pool_size=(2, 2))(x)

        # Layer 9
        x = Conv2D(512, (3,3), strides=(1,1), padding='same', name='conv_9', use_bias=False)(x)
        x = BatchNormalization(name='norm_9')(x)
        x = LeakyReLU(alpha=0.1)(x)

        # Layer 10
        x = Conv2D(256, (1,1), strides=(1,1), padding='same', name='conv_10', use_bias=False)(x)
        x = BatchNormalization(name='norm_10')(x)
        x = LeakyReLU(alpha=0.1)(x)

        # Layer 11
        x = Conv2D(512, (3,3), strides=(1,1), padding='same', name='conv_11', use_bias=False)(x)
        x = BatchNormalization(name='norm_11')(x)
        x = LeakyReLU(alpha=0.1)(x)

        # Layer 12
        x = Conv2D(256, (1,1), strides=(1,1), padding='same', name='conv_12', use_bias=False)(x)
        x = BatchNormalization(name='norm_12')(x)
        x = LeakyReLU(alpha=0.1)(x)

        # Layer 13
        x = Conv2D(512, (3,3), strides=(1,1), padding='same', name='conv_13', use_bias=False)(x)
        x = BatchNormalization(name='norm_13')(x)
        x = LeakyReLU(alpha=0.1)(x)

        skip_connection = x

        x = MaxPooling2D(pool_size=(2, 2))(x)

        # Layer 14
        x = Conv2D(1024, (3,3), strides=(1,1), padding='same', name='conv_14', use_bias=False)(x)
        x = BatchNormalization(name='norm_14')(x)
        x = LeakyReLU(alpha=0.1)(x)

        # Layer 15
        x = Conv2D(512, (1,1), strides=(1,1), padding='same', name='conv_15', use_bias=False)(x)
        x = BatchNormalization(name='norm_15')(x)
```

```
92          x = LeakyReLU(alpha=0.1)(x)
93
94          # Layer 16
95          x = Conv2D(1024, (3,3), strides=(1,1), padding='same', name='conv_16', use_bias=False)(x)
96          x = BatchNormalization(name='norm_16')(x)
97          x = LeakyReLU(alpha=0.1)(x)
98
99          # Layer 17
100         x = Conv2D(512, (1,1), strides=(1,1), padding='same', name='conv_17', use_bias=False)(x)
101         x = BatchNormalization(name='norm_17')(x)
102         x = LeakyReLU(alpha=0.1)(x)
103
104         # Layer 18
105         x = Conv2D(1024, (3,3), strides=(1,1), padding='same', name='conv_18', use_bias=False)(x)
106         x = BatchNormalization(name='norm_18')(x)
107         x = LeakyReLU(alpha=0.1)(x)
108
109         # Layer 19
110         x = Conv2D(1024, (3,3), strides=(1,1), padding='same', name='conv_19', use_bias=False)(x)
111         x = BatchNormalization(name='norm_19')(x)
112         x = LeakyReLU(alpha=0.1)(x)
113
114         # Layer 20
115         x = Conv2D(1024, (3,3), strides=(1,1), padding='same', name='conv_20', use_bias=False)(x)
116         x = BatchNormalization(name='norm_20')(x)
117         x = LeakyReLU(alpha=0.1)(x)
118
119         # Layer 21
120         skip_connection = Conv2D(64, (1,1), strides=(1,1), padding='same', name='conv_21', use_bias=↩
                ↪ False)(skip_connection)
121         skip_connection = BatchNormalization(name='norm_21')(skip_connection)
122         skip_connection = LeakyReLU(alpha=0.1)(skip_connection)
123         skip_connection = Lambda(space_to_depth_x2)(skip_connection)
124
125         x = concatenate([skip_connection, x])
126
127         # Layer 22
128         x = Conv2D(1024, (3,3), strides=(1,1), padding='same', name='conv_22', use_bias=False)(x)
129         x = BatchNormalization(name='norm_22')(x)
130         x = LeakyReLU(alpha=0.1)(x)
131
132         self.feature_extractor = Model(input_image, x)
133         self.feature_extractor.load_weights(FULL_YOLO_BACKEND_PATH)
134
135     def normalize(self, image):
136         return image / 255.
```

D.5. Convoluciones de MobileNet

En el código D.6 se presenta la función que contiene los bloques convolucionales de la arquitectura MobileNet implementada por con el uso de las librerías Tensorflow y Keras.

Código D.6: Convoluciones de MobileNet

```
1
2class MobileNetFeature(BaseFeatureExtractor):
3
4    def ___init___(self, input_size):
5        input_image = Input(shape=(input_size, input_size, 3))
6
7        mobilenet = MobileNet(input_shape=(224,224,3), include_top=False)
8        mobilenet.load_weights(MOBILENET_BACKEND_PATH)
9
10       x = mobilenet(input_image)
11
12       self.feature_extractor = Model(input_image, x)
13
14   def normalize(self, image):
15       image = image / 255.
16       image = image - 0.5
17       image = image * 2.
18
19       return image
```

D.6. Convoluciones de InceptionV3

En el código D.7 se presenta la función que contiene los bloques convolucionales de la arquitectura InceptionV3 implementada con el uso de las librerías Tensorflow y Keras.

Código D.7: Convoluciones de InceptionV3

```
1
2class Inception3Feature(BaseFeatureExtractor):
3
4    def ___init___(self, input_size):
5        input_image = Input(shape=(input_size, input_size, 3))
6
7        inception = InceptionV3(input_shape=(input_size,input_size,3), include_top=False)
8        inception.load_weights(INCEPTION3_BACKEND_PATH)
9
10       x = inception(input_image)
11
12       self.feature_extractor = Model(input_image, x)
13
14   def normalize(self, image):
15       image = image / 255.
16       image = image - 0.5
17       image = image * 2.
18
19       return image
```